Universidade Federal da Paraíba

Programa de Pós-Graduação em Informática

Exact and heuristic approaches for single machine scheduling with inventory constraints and sequence-dependent setup times

Rafael Sobral de Morais

João Pessoa - PB

2026

Universidade Federal da Paraíba

Programa de Pós-Graduação em Informática

Rafael Sobral de Morais

Exact and heuristic approaches for single machine scheduling with inventory constraints and sequence-dependent setup times

Dissertação submetida ao Programa de Pós-Graduação em Informática do Centro de Informática da Universidade Federal da Paraíba, como requisito parcial para obtenção do grau de Mestre em Informática.

Orientador: Anand Subramanian

Coorientador: Teobaldo Leite Bulhões Júnior

João Pessoa - PB

2026

Catalogação na publicação
Seção de Catalogação e Classificação

M828e Morais, Rafael Sobral de.
Exact and heuristic approaches for single machine scheduling with inventory constraints and sequence-dependent setup times / Rafael Sobral de Morais. - João Pessoa, 2026.
75 f. : il.

Orientação: Anand Subramanian.
Coorientação: Teobaldo Leite Bulhões Júnior.
Dissertação (Mestrado) - UFPB/CI.

1. Problema de sequenciamento. 2. Setups. 3. Métodos heurísticos. 4. Inventário. I. Subramanian, Anand. II. Bulhões Júnior, Teobaldo Leite. III. Título.

UFPB/BC CDU 004(043)

Elaborado por GRACILENE BARBOSA FIGUEIREDO - CRB-15/794

UNIVERSIDADE FEDERAL DA PARAÍBA
CENTRO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA

Ata da Sessão Pública de Defesa de Dissertação de Mestrado de Rafael Sobral de Morais, candidato ao título de Mestre em Informática na área de Ciências da Computação, realizada em 30 de janeiro de 2026.

Aos trinta dias do mês de janeiro do ano de dois mil e vinte e seis, às 16h, no Laboratório de Engenharia de Sistemas e Robótica (LASER), reuniram-se os membros da Banca Examinadora constituída para julgar o Trabalho Final do discente **RAFAEL SOBRAL DE MORAIS**, vinculado a Universidade Federal da Paraíba sob a matrícula nº 20241006388, candidato ao grau de Mestre em Informática, na área de "Ciências da Computação", na linha de pesquisa "Metodologia e Técnicas de Computação" do Programa de Pós-Graduação em Informática. A comissão examinadora foi composta pelos professores Anand Subramanian, orientador e presidente da banca; Bruno Petrato Bruck, examinador interno ao programa; Teobaldo Leite Bulhões Junior, examinador interno ao programa; e Yuri Laio Teixeira Veras Silva, examinador externo à Instituição. Dando início aos trabalhos, o presidente cumprimentou os presentes, comunicou a finalidade da reunião e passou a palavra ao candidato para que fizesse, oralmente, a exposição do trabalho de dissertação intitulado "**Abordagens exatas e heurísticas para o problema de escalonamento em máquina única com restrições de inventário e tempos de setup dependentes da sequência**". Concluída a exposição, o candidato foi arguido pela Banca Examinadora, que emitiu o seguinte parecer: "***aprovado***". Do ocorrido, eu, Gean Paulo P. M. de Barros, secretário, lavrei a presente ata que vai assinada por mim e pelos membros da Banca Examinadora. João Pessoa, 30 de janeiro de 2026.

Gean Paulo P. M. de Barros
Secretário - SIAPE 2326476

gov.br Documento assinado digitalmente
ANAND SUBRAMANIAN
Data: 02/02/2026 13:17:58-0300
Verifique em https://validar.iti.gov.br

Prof. Dr. Anand Subramanian
Orientador (PPGI)

gov.br Documento assinado digitalmente
TEOBALDO LEITE BULHOES JUNIOR
Data: 30/01/2026 18:03:55-0300
Verifique em https://validar.iti.gov.br

Prof. Dr. Teobaldo Leite Bulhões Junior
Coorientador (PPGI)

gov.br Documento assinado digitalmente
BRUNO PETRATO BRUCK
Data: 30/01/2026 17:58:24-0300
Verifique em https://validar.iti.gov.br

Prof. Dr. Bruno Petrato Bruck
Examinador interno ao Programa (PPGI)

gov.br Documento assinado digitalmente
YURI LAIO TEIXEIRA VERAS SILVA
Data: 30/01/2026 17:34:34-0300
Verifique em https://validar.iti.gov.br

Prof. Dr. Yuri Laio Teixeira Veras Silva
Examinador externo à Instituição (UFCG)

Abstract

This work addresses the single machine scheduling problem with release dates, sequence-dependent setup times, and inventory constraints, $1|r_j, s_{ij}, inv|C_{\max}$. Four mixed-integer programming formulations are proposed: position-indexed, arc-indexed, arc-inventory-indexed, and arc-time-indexed. Computational experiments were performed, and the results show that the arc-indexed formulation provides the best performance among the models.

Building on this formulation, a branch-and-cut algorithm incorporating inventory capacity cuts is developed, along with a branch-cut-and-price algorithm that extends an existing branch-and-price framework for scheduling problems. Among the exact methods, the branch-cut-and-price approach solves the largest number of instances to optimality, while the branch-and-cut method attains the lowest average gaps.

Two heuristic methods based on the Iterated Local Search framework are also proposed, using ruin-and-recreate and SISRs-inspired perturbation strategies. Both heuristics consistently generate feasible solutions, with the ruin-and-recreate variant achieving particularly competitive performance.

Keywords: Scheduling, setups, inventory.

Resumo

Este trabalho aborda o problema de sequenciamento em uma única máquina com datas de liberação, tempos de preparação dependentes da sequência e restrições de inventário, $1|r_j, s_{ij}, inv|C_{\max}$. Quatro formulações de programação inteira mista são propostas: indexada por posição, indexada por arcos, indexada por arcos–inventário e indexada por arcos–tempo. Experimentos computacionais foram realizados e os resultados mostram que a formulação indexada por arcos apresenta o melhor desempenho entre os modelos.

Com base nessa formulação, é desenvolvido um algoritmo branch-and-cut que incorpora cortes de capacidade de inventário, bem como um algoritmo branch-cut-and-price que estende uma estrutura de branch-and-price já existente para problemas de sequenciamento. Dentre os métodos exatos, o branch-cut-and-price soluciona o maior número de instâncias na otimalidade, enquanto o branch-and-cut obtém os menores gaps médios.

Além disso, são propostos dois métodos heurísticos baseados no framework de Busca Local Iterada, utilizando estratégias de perturbação do tipo ruin-and-recreate e inspiradas no algoritmo SISRs. Ambas as heurísticas geram soluções viáveis de forma consistente, sendo que a variante com ruin-and-recreate apresenta desempenho particularmente competitivo.

Palavras-chave: *Scheduling*, *Setups*, Inventário.

Acknowledgments

I gratefully acknowledge CAPES for the financial support provided through a master's scholarship.

I would like to express my sincere gratitude to my advisor, Prof. Dr. Anand Subramanian, and my co-advisor, Prof. Dr. Teobaldo Bulhões, for their invaluable guidance, continuous support, and insightful contributions throughout this work, as well as for their mentorship since my undergraduate studies.

I am also deeply thankful to the members of the committee for their availability, understanding, and for their constructive comments and valuable suggestions.

I would like to thank all members of LOG for their partnership, collaboration, and stimulating academic environment since my undergraduate years.

My sincere thanks go to my friends for their constant encouragement and support throughout this journey.

I am especially grateful to my girlfriend, Laura, for being by my side, offering unwavering encouragement, motivation, and understanding.

Finally, I express my deepest gratitude to my parents, Luis Carlos and Maria José, for their unconditional love, dedication, and sacrifices in providing me with the opportunities and support necessary to achieve my goals.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Preliminaries

Scheduling problems represent decision-making processes that arise in a wide range of contexts, especially in manufacturing and service environments. This class of problems involves allocating resources to jobs over time, with the goal of optimizing one or more performance objectives. (Pinedo, 2022).

According to Lawler et al. (1993), scheduling and sequencing theory stands out in the field of operations research due to the virtually unlimited number of problem variants it encompasses. Thus, the resources, jobs, and objectives can differ substantially depending on the context and the specific characteristics of each organizational setting.

Resources may range from industrial machines and airport runways to construction workers and computer processing units. Jobs, in turn, may include operations on production lines, control of landings and takeoffs, stages of engineering projects, or software executions. Each of these jobs may have distinct priorities, different release dates, varying processing times, and predefined deadlines for completion.

Problems involving a single-machine configuration are among the most relevant to study, since more complex scenarios can often be reduced to single-machine formulations. These problems consist of scheduling a set of n jobs that must be processed on a single available machine, subject to the constraints imposed by the specific context.

One of the constraints commonly found in scheduling problems is the presence of setup times between jobs, which represent the time required to prepare the machine for processing a job. Setup times may be sequence-independent or sequence-dependent. In the former case, the setup time can simply be added to the job's processing time. In the latter, however, the problem becomes more complex, since the setup required for a job varies according to the job processed immediately before it, meaning that the start and completion times are directly affected by the sequencing.

Another common type of constraint involves release dates for jobs, which specify the moment at which each job becomes available for processing in the system. Release dates can be handled in two ways: by allowing the machine to remain idle until a job is released, or by establishing a sequence that ensures each job begins after its release time, without causing idle periods.

Resource constraints also arise in various scenarios in which jobs consume or produce some resource, such as raw materials, energy, semi-finished goods, and others. Related to resource constraints are inventory constraints, where jobs add or remove items from an inventory with limited capacity. In such cases, it must be ensured that during scheduling, the inventory always contains sufficient resources for processing the jobs and that its capacity is never exceeded.

This work introduces the single-machine scheduling problem with release dates, sequence-dependent setups and inventory constraints, defined as $1|r_j, s_{ij}, inv|C_{max}$ according to the notation proposed by Graham et al. (1979). The main goal is to develop and explore different exact and heuristic methods, analyzing how the characteristics of the problem influence the complexity of its solution. To this end, four mathematical formulations are proposed, together with exact approaches based on *Branch-and-Cut* (B&C) and *Branch-Cut-and-Price* (BCP), as well as heuristic methods that combine the *Iterated Local Search* (ILS) framework with ruin-and-recreate strategies.

1.2 Motivation

In a world of growing competitiveness, the use of decision-support tools becomes essential for ensuring strong performance across organizations in various sectors. The efficiency is directly tied to the effective management of resources, and in this

context, job scheduling, as an integral part of the decision-making process, plays a fundamental role in most manufacturing and production environments (Pinedo, 2022). Implementing an efficient scheduling system is a strategic investment capable of generating significant benefits, such as cost reduction, productivity gains, and even improved satisfaction among customers and workers.

According to Briskorn and Pesch (2013), the scheduling problem with release dates and inventory constraints can be observed in the context of a transshipment terminal, where the release dates represent the arrival times of trucks at the terminal. Some trucks arrive loaded with items that must be unloaded in the terminal and require available storage space in the inventory. There are also empty trucks that arrive to be loaded with items from the inventory. In this setting, the sequencing of loading and unloading operations must ensure that the inventory capacity is never exceeded, while also guaranteeing that the loading demand of each truck is always fulfilled.

In many cases, the presence of setup times between jobs is disregarded to simplify problem solving. However, in certain scenarios, setup operations have a considerable impact on performance, and accounting for them in the planning stage can lead to significant gains (Allahverdi et al., 2008).

Given this context, the study of scheduling problems with release dates, inventory constraints, and setup times becomes especially relevant, as such problems encompass several characteristics commonly found across a wide range of industrial and service environments.

An illustrative example of a problem with these characteristics is a small transshipment terminal similar to that described by Briskorn and Pesch (2013). Such a terminal operates only a few days per week and processes a limited number of trucks each day, using a small fleet of forklifts. Trucks transport products in different packaging formats, which directly affect handling requirements. Some trucks carry palletized goods with varying pallet sizes and stacking patterns, requiring changes in forklift configuration and thus incurring setup times. Other trucks transport products in drums, which necessitate equipping the forklifts with drum clamps.

Consequently, the sequence of loading and unloading operations must explicitly account for setup times associated with changes in forklift configuration, while simultaneously respecting truck release dates, the limited availability of forklifts, and the inventory capacity constraints of the terminal.

Figure 1.1: Forklift equipped with a drum clamp. Source: `http://alphaliftuae.com/forklift-attachments/drum-clamp/`

According to Davari et al. (2020), the problem of minimizing the makespan on a single machine with release dates, and inventory constraints is NP-hard. Since the variant with sequence dependent setups is a generalization of this problem, one can conclude that $1|r_j, s_{ij}, inv|C_{max}$ is also NP-hard, meaning that the problem is computationally difficult to solve exactly and the development of efficient solution methods is relevant.

1.3 Objectives

The main objective of this work is the development of solution methods for the single machine scheduling problem with release dates, inventory constraints and sequence dependent setups.

The remaining objectives are listed as follows.

- Conduct a survey of the state of the art on related problems;
- Propose a set of benchmark instances for the proposed problem;
- Formulate different mathematical modeling approaches for the problem;
- Propose exact methods based on *Branch-and-cut* and *Branch-cut-and-price* for the problem under study;
- Compare the performance of different heuristic methods for solving the problem.

1.4 Outline

This work is organized as follows: Chapter 2 presents basic scheduling concepts and a literature review. Chapter 3 describes the proposed formulations and algorithms. Chapter 4 presents the computational experiments and results. Finally, Chapter 5 provides concluding remarks and outlines directions for future research.

Chapter 2

Theoretical framework and literature review

This chapter introduces the main concepts related to scheduling problems, such as the different machine configurations, constraints, attributes and properties, as well as the various objective functions commonly found in the scheduling literature. Subsequently, a literature review is provided, highlighting studies that address variants closely related to the problem under consideration.

2.1 Scheduling notation

In order to simplify the identification of scheduling problems by taking their characteristics into account, Graham et al. (1979) proposed a three field notation $\alpha|\beta|\gamma$, in which α represents the machine configuration, β describes the main constraints and other process characteristics, and γ defines the objective function.

2.1.1 Machines

- Single machine – 1: There is only one machine or resource that is shared by all jobs.
- Identical parallel machines – P_m: In this environment, there are m identical machines that can process jobs in parallel.
- Uniform parallel machines – Q_m: There are m machines in parallel, and each machine has a distinct processing speed that is independent of the jobs.

- Unrelated parallel machines – R_m: There are m machines in parallel, and the processing speed of each machine varies depending on the job to be processed.

2.1.2 Jobs

Jobs are defined by a set of properties that specify processing data and constraints. When a property is directly related to a constraint, it can be included in the β field of the problem notation. Some of the properties found in the literature are listed below:

- Processing time – p_j : The time required to process the job. In some problems, this time may depend on the sequence or the machine.
- Release date – r_j : The earliest time at which the processing of the job can begin.
- Due date – d_j : The scheduled completion date of the job. Early or late completion may incur a penalty.
- Deadline – $\bar{d}_j$: The strict latest date for completing the job, which must be met.
- Penalty – w_j : Weight indicating the priority of the job. It can be used to penalize early or late completion.
- Resource demand – q_j : The amount of resource consumed by the job. If the jobs produces the resource, the value is negative.

2.1.3 Constraints and attributes

- Sequence dependent setup times – s_{ij}: Before processing a job, the machine must be set up, and the time required for this procedure varies depending on the job to be processed and the immediately preceding job.
- Machine-dependent setup times – s_{jk}: In parallel machine environments, the setup time for a job may vary depending on the machine on which it will be processed.
- Non-renewable resources – NR: Jobs consume a limited resource that is not replenished during the production horizon.

- Resource constraints – *res*: Jobs consume a certain resource that may be supplied at a constant rate over time or only at specific points in time.
- Inventory constraints – *inv*: Jobs consume or produce a resource that is stored in an inventory with limited capacity.
- Precedence – *prec*: There are precedence relationships between jobs, in which a job cannot be processed before one or more other jobs.
- Machine availability – a_k: In problems with parallel machines, not all machines may be available at the same time. In this case, each machine k becomes available from time a_k.

2.1.4 Objectives

- Makespan – C_{max}: The goal is to minimize the total completion time of production. C_{max} represents the time at which the last job is finished, equivalent to the largest completion time C_j.
- Weighted completion times – $\sum w_j C_j$: In this case, the completion time of each job is associated with a weight w_j, indicating a preference for certain jobs to be finished before others.
- Maximum Lateness – L_{max}: The goal is to minimize the maximum lateness in the system, where the lateness of a job is given by $L_j = C_j - d_j$.
- Weighted Tardiness – $\sum w_j T_j$: The objective is to minimize the sum of weighted tardiness, where $T_j = \max\{C_j - d_j, 0\}$.
- Earliness – E_j: In some situations, early completion of a job is penalized, which is given by $E_j = \max\{d_j - C_j, 0\}$.
- Number of late jobs – $\sum U_j$: The objective is to minimize the total number of late jobs, where $U_j = 1$ if $C_j - d_j \geq 0$, and $U_j = 0$ otherwise.

2.2 Related work

2.2.1 Problems with release dates and setup times

In one of the earliest studies to address release dates together with setup times, Bianco et al. (1988) introduced a mixed-integer linear programming (MILP) formulation and a branch-and-bound (B&B) algorithm for a problem equivalent to

minimizing the makespan on a single machine, assuming zero processing times and sequence-dependent setup times. The authors tested their approach on instance sets of sizes $n = 10$, $n = 15$ e $n = 20$, and found that the performance of the algorithm is heavily affected by the ratio between the smallest and largest setup times.

Ovacikt and Uzsoy (1994) proposed a set of benchmark instances and a decomposition heuristic, referred to as the *Rolling Horizons Procedure* (RHP), for the problem of minimizing maximum lateness on a single machine $(1|r_j, s_{ij}|L_{\max})$. Their approach was shown to provide high-quality solutions within reasonable computational times. Subsequently, the same problem was addressed by Shin et al. (2002), who developed a *tabu search* algorithm capable of generating solutions more quickly and with superior quality when compared to the RHP.

A mathematical formulation and a heuristic algorithm for the problem of minimizing weighted tardiness on a single machine $(1|r_j, s_{ij}|\sum w_j T_j)$ were introduced in the work of Chang et al. (2004). The proposed heuristic proved to be effective for the problem, having a complexity of $\mathcal{O}(n^3)$ and achieving the optimal solution in more than 80% of the tested instances.

The minimization of total tardiness $(1|r_j, s_{ij}|\sum T_j)$ was addressed by Nogueira et al. (2022), who proposed a hybrid heuristic that combines *variable neighborhood search* (VNS) and Lagrangian relaxation. The performance of this algorithm was compared with several methods found in the literature, including ILS, tabu search, and a *hybrid evolutionary algorithm* (HEA). The proposed hybrid algorithm achieved the smallest average gaps and found the optimal solution for the vast majority of instances with known optima.

Chou et al. (2008) used a constraint programming model and a B&B to solve the problem of minimizing the weighted completion times on a single machine $(1|r_j, s_{ij}|\sum w_j C_j)$ for instances with up to 40 jobs. In addition, two heuristics were proposed to handle larger instances. The *best index dispatch* (BID) heuristic, derived from the B&B method, proved to be more effective than other tested approaches in generating high-quality solutions for instances ranging from 5 to 500 jobs.

The problem of minimizing the makespan on a single machine with non-zero processing times, release dates and sequence dependent setups $(1|r_j, s_{ij}|C_{max})$ was first addressed in the work of Montoya-Torres et al. (2012), who proposed a random insertion heuristic. This heuristic was later outperformed by the *beam search* (BS) method introduced by Velez-Gallego et al. (2016), who also presented a MILP for-

mulation with arc-indexed decision variables, capable of solving only a few instances with up to 50 jobs within a one-hour time limit. The same problem was subsequently tackled by Fan et al. (2019), who proposed the *Variable Block Insertion Heuristic* (VBIH). When compared against the BS method of Velez-Gallego et al. (2016) and the iterated greedy algorithm of Ruiz and Stützle (2007), the VBIH demonstrated competitive performance, producing high-quality solutions for instances with up to 75 jobs.

Morais et al. (2024) put forward a branch-and-price (BP) algorithm and a hybrid heuristic that combines ILS and BS to solve the $1|r_j, s_{ij}|C_{max}$ problem. In this work, the authors developed an efficient move-evaluation strategy within the local search phase, which significantly reduced the computational complexity and running time of the algorithm. When compared with methods available in the literature, the proposed ILS-BS approach proved markedly superior, achieving the smallest gaps relative to the best-known solutions across all groups of instances.

Problems involving release dates and setup times are also found in other machine environments. Ying and Cheng (2010) used an IG heuristic to tackle the problem of minimizing maximum lateness on identical parallel machines ($P|r_j, s_{jk}|L_{\max}$). Their method proved more effective than the RHPs proposed by Ovacikt and Uzsoy (1994), obtaining 16.78% of the optimal solutions and improving the solution quality for more than half of the benchmark instances. The IG approach was later enhanced by Lin et al. (2011), who incorporated a *simulated annealing* technique, achieving 22% of the optimal solutions and identifying new lower bounds, with only a slight increase in average runtime.

Driessel and Mönch (2011) addressed the problem of minimizing weighted tardiness with precedence constraints ($P|r_j, s_{jk}, \text{prec}|w_jT_j$), presenting several variants of VNS that employed four types of neighborhood structures: single-machine insertion, single-machine swap, inter-machine insertion, and inter-machine swap. The proposed VNS variants were compared with a genetic algorithm (GA), and the best-performing VNS version achieved an average cost 1.31% lower than that of the GA, while requiring a similar computational time. Moreover, it was observed that increasing either the population size or the number of iterations in the GA did not yield significant improvements, whereas expanding the number of neighborhoods in the VNS led to higher-quality solutions.

In the context of uniform parallel machines, the problem of minimizing the sum of

earliness and tardiness costs ($Q|r_j, s_{ijk}|\sum e_j E_j + t_j T_j$) was studied by Balakrishnan et al. (1999), who presented a MILP formulation. Using this formulation together with Benders decomposition, the authors were able to solve instances with up to 12 jobs and 3 machines. Subsequently, Ümit Bilge et al. (2004) presented different tabu search strategies for the problem of minimizing total tardiness ($Q|r_j, s_{ijk}|\sum T_j$). The authors observed that the intensification-oriented strategy produced lower-cost solutions when compared to the diversification-oriented strategy.

The same problem was addressed by Kim et al. (2007), who again employed tabu search, but with different methods for generating initial solutions and distinct candidate list strategies. In addition, the authors proposed algorithms based on simulated annealing. The results showed that, in terms of solution quality, the tabu search that used an expanded candidate list and allowed insertion and swap moves both within and across machines outperformed the other methods, including the tabu search of Ümit Bilge et al. (2004), achieving improvements of up to 9% on instances with tighter due dates.

Tabu search was also the method proposed by Logendran et al. (2007) for the problem of minimizing weighted tardiness on unrelated parallel machines with dynamic availability. The authors concluded that using a tabu list of fixed size together with a short-term memory search strategy is advisable for small instances, whereas for larger instances the best results are obtained with a variable-size tabu list and long-term memory. A similar problem, but without machine availability constraints, denoted by ($R|r_j, s_{ijk}|\sum w_j T_j$), was addressed by Lin and Hsieh (2014), who proposed a mixed-integer programming model and a hybrid population-based metaheuristic (IHM). This method was compared with the tabu search of Logendran et al. (2007) and with an ant colony optimization algorithm (ACO). The results showed only marginal differences between the solutions produced by IHM and tabu search, but on average the IHM obtained solutions of higher quality than the other heuristics, both for small instances (3 machines and 12 jobs) and for large instances (10 machines and 100 jobs).

Diana et al. (2018) presented an ILS-VND metaheuristic for the problem $R|r_j, s_{ijk}|\sum w_j T_j$, which was compared with the IHM proposed by Lin and Hsieh (2014). In the VND phase, neighborhoods based on swap and insertion moves, both within and between machines, are explored, and diversification is achieved through a perturbation stage that performs two random inter-machine moves. Among the

16 groups of instances evaluated, the ILS-VND produced the best average solution in 14 groups, always with lower computational time when compared with the IHM.

Table 2.1: Related work on scheduling with release dates and setups

Reference	α	β	γ	**Approach**
Bianco et al. (1988)	1	r_j, s_{ij}, p_j=0	C_{max}	MILP, B&B
Ovacikt and Uzsoy (1994)	1	r_j, s_{ij}	L_{max}	RHP
Balakrishnan et al. (1999)	Q_m	r_j, s_{ijk}	$\sum T_j$	MIP, Benders
Shin et al. (2002)	1	r_j, s_{ij}	L_{max}	Tabu search
Chang et al. (2004)	1	r_j, s_{ij}	$\sum w_j T_j$	MIP, heuristic
Nogueira et al. (2022)	1	r_j, s_{ij}	$\sum T_j$	VNS + Lagrangian Relaxation
Chou et al. (2008)	1	r_j, s_{ij}	$\sum w_j C_j$	CP, B&B, BID
Montoya-Torres et al. (2012)	1	r_j, s_{ij}	C_{max}	Random Insertion
Velez-Gallego et al. (2016)	1	r_j, s_{ij}	C_{max}	MILP, BS
Fan et al. (2019)	1	r_j, s_{ij}	C_{max}	VBIH
Morais et al. (2024)	1	r_j, s_{ij}	C_{max}	BP, ILS-BS
Ying and Cheng (2010)	P_m	r_j, s_{jk}	L_{max}	IG
Lin et al. (2011)	P_m	r_j, s_{ij}	L_{max}	IG + SA
Driessel and Mönch (2011)	P_m	$r_j, s_{ij}, prec$	$\sum w_j T_j$	VNS
Ümit Bilge et al. (2004)	Q_m	r_j, s_{ijk}	$\sum T_j$	Tabu search
Kim et al. (2007)	Q_m	r_j, s_{ijk}	$\sum T_j$	Tabu search
Logendran et al. (2007)	R_m	$r_j, a_j, s_{ij},$	$\sum w_j T_j$	Tabu search
Lin and Hsieh (2014)	R_m	$r_j, s_{ijk},$	$\sum w_j T_j$	MILP, IHM
Diana et al. (2018)	R_m	$r_j, s_{ijk},$	$\sum w_j T_j$	ILS-VND

2.2.2 Problems with inventory constraints

Despite the practical relevance of inventory considerations and the extensive scheduling literature, only a limited number of studies have examined scheduling variants that incorporate inventory constraints. According to Davari et al. (2020), such constraints generalize non-renewable resource constraints, which involve resources that are consumed during job processing and not restored afterward, such as raw materials, fuel, energy or financial resources.

Carlier and Kan (1982) introduced a polynomially bounded algorithm to solve a class of scheduling problems with non-renewable resources and precedence constraints. The authors state that with the addition of jobs producing resources, the problem becomes significantly more difficult. Slowiński (1984) presented a linear programming formulation for the scheduling problem with parallel machines subject to financial constraints. Toker et al. (1991) established an equivalence between the optimal solution of the flow shop problem with two machines ($F2||C_{max}$) and the solution of $1|\text{NR}|C_{max}$, in which a unit of resource is released at each discrete time period.

Later, the equivalence between the two problems for any regular objective function was demonstrated in the work of Sergey Kovalev and Bécuwe (2024).

The complexity of many single machine problems with non-renewable resources was shown in the work of Gafarov et al. (2011), which considered the $1|\text{NR}|C_{max}$, $1|\text{NR}|\sum T_j$, $1|\text{NR}|\sum L_{max}$ e $1|\text{NR}|\sum U_j$. Györgyi and Kis (2014) put forward exact and approximation schemes for some special cases of the single machine scheduling problem with the objective to minimize the makespan subject to non-renewable resources constraints. This approach was later expanded to the parallel machines context by Györgyi and Kis (2017), in which the authors also considered variants with release dates and maximum lateness minimization.

In the production scheduling context, resource constraints were the subject of the work of Briskorn et al. (2010), which proved the complexity of special cases of $1|inv|\sum w_jC_j$, $1|inv|L_{max}$ e $1|inv|\sum U_j$, considering an unlimited inventory. The same is assumed by Briskorn et al. (2013), who proposed a B&B and a dynamic programming algorithm for the $1|inv|\sum w_jC_j$, being capable of solving instances with up to 20 jobs. Another B&B was presented by Briskorn and Leung (2013) to the problem of minimizing the weighted tardiness. The authors developed many strategies to obtain bounds and applied a method that fixes jobs in sequence positions based on the idea of *earliest due date* (EDD). Using that scheme, the B&B was able to solve instances with 240 jobs.

Briskorn and Pesch (2013) places the single machine scheduling problem with inventory constraints in the context of a transshipment terminal with a limited storage capacity, in which jobs represent the loading and unloading operations that occur on a single dock. Different objective functions were considered, and algorithms based on *variable neighborhood search* (VNS) and *variable neighborhood descent* (VND) were proposed with the latter being capable of producing high-quality solutions at the expense of greater computational effort.

The problem of scheduling loading and unloading operations on a transshipment terminal with a single station was also studied by Ghorbanzadeh et al. (2019), who presented the problem as a single machine scheduling with release dates and inventory constraints, with the objective of minimizing the makespan ($1|r_j, \text{inv}|C_{max}$). A B&B and dynamic programming algorithm were proposed as solution approaches.

Davari et al. (2020) tackled a similar problem, using a MILP with arc-time indexed variables, as well as an alternative formulation in which the indexing is based on

job position. In addition, they proposed an exact guess-and-check (GC) algorithm, which was tested on a benchmark set introduced by the authors containing 900 instances with sizes ranging from 10 to 100 jobs. The same set was later used by Guerra et al. (2024), who proposed an hybrid metaheuristic denoted *Hybrid Population-based Ruin-and-Recreate* (HyPRR), which combines elements of populational algorithms and ruin-and-recreate methods. The HyPRR achieved competitive results when compared to the GC algorithm and genetic algorithms found in the literature, producing high-quality solutions for large instances with reasonable time.

Neumann and Schwindt (2003) addressed the project scheduling problem with temporal and inventory constraints, with applications in the flux sequencing in chemical industries. In this study the authors proposed an efficient B&B capable of solving instances with 100 jobs and multiple resources within less than a minute of runtime.

Schwindt and Trautmann (2000) proposed a B&B for a batch scheduling problem with several constraints, such as labor limitations, safety stock, and capacity-limited inventory.

Bartels and Zimmermann (2009) tackled a scheduling problem in the context of an automotive industry, in which tests in experimental vehicles must be sequenced subject to resources constraints and intervals between jobs. A MILP was implemented to solve instances with up to 20 jobs. In addition, a priority-rule based heuristic was developed to solve larger problems, being effective for instances with up to 600 jobs.

In turn, the literature on problems that consider setups together with inventory or resource constraints is quite limited. Torabi et al. (2013) addressed a multi-objective problem in an unrelated parallel-machine environment with sequence-dependent setups, release dates, and uncertain processing times. The authors proposed a multi-objective particle swarm optimization (MOPSO) algorithm, which demonstrated the ability to generate solutions with greater diversity and quality compared to the traditional particle swarm optimization method.

Motivated by a real-world problem in the steel industry, Herr and Goel (2016) addressed the problem of minimizing total tardiness on a single machine with resource constraints and setup times between jobs belonging to different families. They presented a MILP and an ILS-based heuristic. Within the one-hour time limit, the model was able to solve small instances with 8 and 10 jobs, for which the heuristic also found the optimal solution. However, for larger instances, the model failed to find the optimal solution, whereas the heuristic produced better solutions with

lower computational effort, making it more suitable for practical application. The ILS was later outperformed by the iterated greedy algorithm proposed by Pinheiro and Arroyo (2020), in which, at each iteration, a solution undergoes destruction, repair, and local search until the algorithm reaches a maximum number of iterations without improvement.

Yepes-Borrero et al. (2020) presented a MILP and a metaheuristic based on *greedy randomized adaptive search procedure* (GRASP) for the problem of minimizing the makespan on unrelated parallel machines with resource constraints and sequence dependent setups, differing from other work in the literature by considering that resources are only consumed during setup operations. The MILP only solved very small instances, with up to 6 jobs.

Soares and Carvalho (2022) proposed a *biased random-key genetic algorithm* (BRKGA) for makespan minimization on identical parallel machines with resource constraints and sequence dependent setups ($P_m|s_{ij}, \text{res1}|C_{max}$). In this case the resources refer to the limited molds used during the processing of jobs. The proposed method found the optimal solution in all instances with known optima, and achieved an average reduction of 7,62% in upper bounds values.

The problem of minimizing the weighted completion time subject to the availability of a single resource was addressed by Su and Lin (2022), who developed a B&B based heuristic. In this problem, which can be denoted as $1|\alpha_j, \beta_j|w_jC_j$, each job requires an amount of resource α during its processing, and upon completion, an amount β of the resource is returned to the inventory, which has unlimited capacity.

This dissertation introduces the problem of minimizing the makespan on a single machine, with release dates, sequence dependent setups and inventory constraints, denoted $1|r_j, s_{ij}, \text{inv}|C_{max}$.This problem has not been explicitly addressed in the literature and can be viewed as a combination of the $1|r_j, s_{ij}|C_{max}$ problem studied by Montoya-Torres et al. (2012), Velez-Gallego et al. (2016), Fan et al. (2019), and Morais et al. (2024), and the $1|r_j, \text{inv}|C_{max}$ problem considered by Davari et al. (2020).

Table 2.2: Related work on scheduling with release dates and inventory or resource constraints

Reference	α	β	γ	**Approach**
Briskorn et al. (2013)	1	inv	$\sum w_j C_j$	B&B, PD
Briskorn and Leung (2013)	1	inv	$\sum w_j T_j$	B&B
			$\sum w_j C_j$	VND
Briskorn and Pesch (2013)	1	inv	$\sum w_j T_j$	VNS
			$\sum w_j U_j$	GA
			$\sum w_j C_j$	
Torabi et al. (2013)	R_m	r_j, res, s_{ij}	$\sum w_j T_j$	MOPSO
			$\sum w_j L_j$	
Herr and Goel (2016)	1	res, s_{ij}	$\sum T_j$	MIP, ILS
Ghorbanzadeh et al. (2019)	1	r_j, inv	C_{max}	B&B, PD
Pinheiro and Arroyo (2020)	1	res, s_{ij}	$\sum T_j$	IG
Davari et al. (2020)	1	r_j, inv	C_{max}	MIP, GC
Yepes-Borrero et al. (2020)	R_m	res, s_{ij}	C_{max}	MIP, GRASP
Soares and Carvalho (2022)	P_m	res, s_{ij}	C_{max}	BRKGA
Su and Lin (2022)	1	α_j, β_j	$w_j C_j$	B&B
Guerra et al. (2024)	1	r_j, inv	C_{max}	HyPRR, HGS

Chapter 3

Methodology

3.1 Problem definition

Let $J = \{1, 2, \ldots, n\}$ be the set of n jobs to be processed on a single machine. Each job $j \in J$ has a processing time p_j and becomes available for execution at its release date r_j. Processing job j immediately after job i requires a setup time s_{ij}, while the first job in the sequence requires an initial setup time s_{0j}. Setup times are non-anticipatory, meaning that the setup for job j can only start after its release at time r_j.

The job set J can be partitioned into two subsets: J^-, which contains the jobs that consume the resource, and J^+, which contains the jobs that produce the resource. Processing a job $j \in J^-$ reduces the inventory level by $q_j \leq 0$, whereas processing a job $j \in J^+$ increases it by $q_j \geq 0$. The resources are stored in an inventory with limited capacity I_C. A major challenge of the problem is to ensure that, throughout the sequence, the inventory level never exceeds its maximum capacity and never falls below the amount required to process resource-consuming jobs.

A solution to the problem is represented by a sequence $\pi = (\pi_1, \pi_2, \ldots, \pi_n)$ in which each job $j \in J$ is executed exactly once. The completion time C_j of a job j is defined as follows. If j is the first job in the sequence ($j = \pi_1$), then $C_j = r_j + s_{0j} + p_j$; otherwise, letting i be the job that immediately precedes j, we have $C_j = \max\{C_i, r_j\} + s_{ij} + p_j$.

An example instance of the problem is presented in Table 3.1, where $I_0 = 10$ and $I_C = 12$. In this instance, there are five jobs to be scheduled, with their respective

release dates, processing times, and resource demands. The setup times between jobs are also provided in the table. A solution to this instance is illustrated in the figure 3.1, where the sequence $\pi = (1, 4, 3, 5, 2)$ results in a makespan of $C_{\max} = 21$ and an inventory level of 12 units at its peak.

j	r_j	p_j	q_j
1	1	1	-4
2	3	2	1
3	10	1	5
4	8	2	-2
5	4	2	2

s_{ij}	**1**	**2**	**3**	**4**	**5**
0	2	5	6	5	7
1	-	9	8	1	5
2	4	-	6	5	3
3	1	3	-	8	2
4	10	2	2	-	7
5	8	1	5	7	-

Table 3.1: Instance example

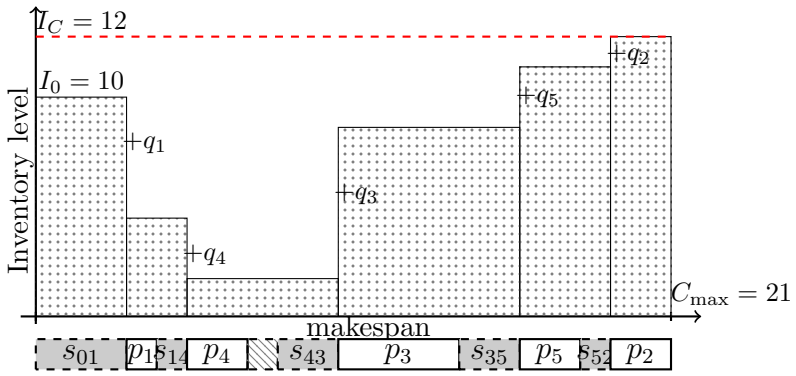

Figure 3.1: Example of a solution $\pi = (1, 4, 3, 5, 2)$

3.2 Mathematical formulations

3.2.1 Position indexed formulation

This formulation adapts the model originally proposed in Davari et al. (2020) for the $1|r_j, inv|C_{\max}$ problem. Binary variables x_{js} are defined to indicate whether job $j \in J$ is processed in position $s \in S$ of the sequence, where $S = \{1, \ldots, n\}$. The inventory level after processing the job assigned to position $s \in S$ is represented by variables y_s. Completion times are modeled through variables C_s, which denote the completion time of the job scheduled in position s.

$$\min \quad C_n \tag{3.1}$$

Subject to:

$$\sum_{s=1}^{n} x_{js} = 1 \qquad \forall j \in J \tag{3.2}$$

$$\sum_{j \in J} x_{js} = 1 \qquad \forall s \in S \tag{3.3}$$

$$C_1 = \sum_{j \in J} (r_j + p_j + s_{0j}) x_{j1} \tag{3.4}$$

$$C_s \geq C_{s-1} + (p_j + s_{ij})(x_{is-1} + x_{js} - 1) \quad \forall s \in S \setminus \{1\}, i \in J, j \in J \setminus \{i\} \tag{3.5}$$

$$C_s \geq (r_j + p_j + s_{ij})(x_{is-1} + x_{js} - 1) \quad \forall s \in S \setminus \{1\}, i \in J, j \in J \setminus \{i\} \tag{3.6}$$

$$y_1 = I_0 + \sum_{j=1}^{n} q_j x_{j1} \tag{3.7}$$

$$y_s = y_{s-1} + \sum_{j=1}^{n} q_j x_{js} \qquad \forall s \in S \setminus \{1\} \tag{3.8}$$

$$C_s \geq 0 \qquad \forall s \in S \tag{3.9}$$

$$x_{js} \in \{0, 1\} \qquad \forall j \in J, s \in S \tag{3.10}$$

$$0 \leq y_s \leq I_C \qquad \forall s \in S \tag{3.11}$$

The objective function (3.1) minimizes the completion time of the job assigned to the last position in the sequence. Constraints (3.2) and (3.3) ensure that each job is assigned to exactly one position, and that each position can accommodate at

most one job. The completion time of the first job in the sequence is computed by constraint (3.4). Constraints (3.5) and (3.6) determine the completion times of all subsequent jobs, with (3.6) ensuring that the release dates are respected. Constraint (3.7) calculates the inventory level after processing the first job, whereas constraints (3.8) compute the inventory levels for the remaining positions. Finally, constraints (3.9)–(3.11) define the domains of the decision variables.

3.2.2 Arc indexed formulation

The proposed model is based on the formulation presented in Velez-Gallego et al. (2016) for the $1|r_j, s_{ij}|C_{max}$ problem, which employs an arc-indexed representation. In this formulation, the decision variables x_{ij} take value 1 if job $j \in J'$ is processed immediately after job $i \in J'$ in the sequence. The set J' is defined as $J' = J \cup \{0\}$, where 0 is an artificial job used to define both the start and end of the schedule. The continuous variables b_i denote the starting time of each job $i \in J$, while continuous variable σ measures the total idle time in the schedule.

In order to adapt the formulation to the problem under study, additional continuous variables y_{ij} are introduced to represent the inventory level on arc (i, j) for $i, j \in J'$. The complete formulation is presented below:

$$\min \quad \sum_{k \in J} p_k + \sum_{i \in J'} \sum_{j \in J} s_{ij} x_{ij} + \sigma \tag{3.12}$$

Subject to:

$$\sum_{i \in J' \setminus \{j\}} x_{ij} = 1 \qquad \forall j \in J' \tag{3.13}$$

$$\sum_{j \in J' \setminus \{i\}} x_{ij} = 1 \qquad \forall i \in J' \tag{3.14}$$

$$b_0 = 0 \tag{3.15}$$

$$\sigma \geq b_k + p_k + \sum_{j \in J' \setminus \{k\}} s_{jk} x_{jk} - \sum_{j \in J} p_j - \sum_{i \in J'} \sum_{j \in J' \setminus \{i\}} s_{ij} x_{ij} \qquad \forall k \in J' \tag{3.16}$$

$$b_i + p_i + \sum_{k \in J' \setminus \{i\}} s_{ki} x_{ki} + M(x_{ij} - 1) - b_j \leq 0 \qquad \forall i \in J', j \in J \setminus \{i\} \tag{3.17}$$

$$r_j - b_j \leq 0 \qquad \forall j \in J \tag{3.18}$$

$$\sum_{k \in J' \setminus \{j\}} y_{jk} - \sum_{i \in J' \setminus \{j\}} y_{ij} = q_j \qquad \forall j \in J \tag{3.19}$$

$$I_0 x_{0j} - y_{0j} = 0 \qquad \forall j \in J \tag{3.20}$$

$$\delta_i x_{ij} - y_{ij} \leq 0 \qquad \forall i, j \in J' \tag{3.21}$$

$$0 \leq y_{ij} \leq I_c \qquad \forall i, j \in J' \tag{3.22}$$

$$x_{ij} \in \{0, 1\} \qquad \forall i, j \in J' \tag{3.23}$$

$$y_{ij} \geq 0 \qquad \forall i, j \in J' \tag{3.24}$$

$$b_j \geq 0 \qquad \forall j \in J' \tag{3.25}$$

$$\sigma \geq 0 \tag{3.26}$$

Equation (3.12) presents the objective function, which minimizes the makespan. This value is composed of three components: the total processing time, the setup times, and the total idle time. Constraints (3.13) and (3.14) ensure that each job has exactly one predecessor and one successor. Constraint (3.15) defines that the dummy job is processed at the beginning of the sequence. Constraints (3.16) compute the total idle time. Constraints (3.17) ensure that the start of a job occurs only after the completion of its predecessor. Constraints (3.18) impose that jobs can only start after their release dates. Constraints (3.19) compute the inventory level on each

arc, while (3.20) computes the inventory level on the initial arc. Constraints (3.21)–(3.22) ensure that inventory limits are respected. Finally, constraints (3.23)–(3.26) define the domain of the variables.

3.2.3 Arc-inventory formulation

In this formulation, binary decision variables x_{ij}^{q} are introduced, taking value 1 if job $j \in J$ is processed immediately after job $i \in J$ and the inventory level after processing job j is $q \in Q_j$. The set Q_j is defined as $Q_j = \{q_j^{\min}, \ldots, q_j^{\max}\}$, where $q_j^{\min} = 0$ and $q_j^{\max} = I_c - q_j$ for $j \in J^-$, and $q_j^{\min} = q_j$ and $q_j^{\max} = I_c$ for $j \in J^+$. This definition ensures that the inventory limits are respected throughout the schedule.

All remaining decision variables remain unchanged with respect to the formulation presented in Section 3.2.2. The complete mathematical formulation is provided below:

$$\min \quad \sum_{k \in J} p_k + \sum_{i \in J'} \sum_{j \in J} \sum_{q=0}^{I_C} s_{ij} x_{ij}^{q} + \sigma \tag{3.27}$$

Subject to:

$$\sum_{i \in J' \setminus \{j\}} \sum_{q=0}^{I_C} x_{ij}^{q} = 1 \qquad \forall j \in J' \tag{3.28}$$

$$\sum_{j \in J' \setminus \{i\}} \sum_{q=0}^{I_C} x_{ij}^{q} = 1 \qquad \forall i \in J \tag{3.29}$$

$$\sum_{j \in J'} x_{0j}^{I_0 + q_j} = 1 \tag{3.30}$$

$$\sum_{j \in J' \setminus \{i\}} x_{ji}^{q} - \sum_{j \in J' \setminus \{i\}} x_{ij}^{q+q_j} = 0 \qquad \forall i \in J, q \in Q_i \tag{3.31}$$

$$b_k + p_k + \sum_{j \in J' \setminus \{k\}} \sum_{q \in Q_k} s_{jk} x_{jk}^q - \sum_{j \in J} p_j - \sum_{i \in J'} \sum_{j \in J' \setminus \{i\}} \sum_{q \in Q_j} s_{ij} x_{ij}^q - \sigma \leq 0 \qquad \forall k \in J' \tag{3.32}$$

$$b_i + p_i + \sum_{k \in J' \setminus \{i\}} \sum_{q \in Q_i} s_{ki} x_{ki}^q + M \left(\sum_{q \in Q_j} (x_{ij}^q) - 1 \right) - b_j \leq 0 \qquad \forall i \in J, j \in J' \tag{3.33}$$

$$x_{ij}^q \in \{0, 1\} \qquad \forall i, j \in J', q \in Q_j \tag{3.34}$$

$$(3.15), (3.18), (3.25), (3.26).$$

Overall, the objective function and constraints retain a structure similar to that of the arc-indexed formulation, with minor modifications to accommodate the newly introduced arc–inventory–indexed variables. In particular, constraints (3.30) define the inventory level after the first job in the sequence, while constraints (3.31) enforce inventory consistency along the remaining arcs.

3.2.4 Arc-time formulation

The formulation introduces the binary decision variables x_{ij}^t, where $x_{ij}^t = 1$ if job $i \in J'$ is processed immediately before job $j \in J'$ and job j starts at time $t \in \{0, 1, \ldots, t_{max} - p_j\}$, otherwise, $x_{ij}^t = 0$. In addition, continuous variables y_t represent the inventory level at time $t \in \{0, 1, \ldots, t_{max}\}$, where t_{max} is an upper bound on the *makespan*. The proposed model is described below:

$$\min \quad C_{max} \tag{3.35}$$

Subject to:

$$\sum_{i \in J' \setminus \{j\}} \sum_{t=p_i}^{t_{max}-p_j} x_{ij}^t = 1 \qquad \forall j \in J' \tag{3.36}$$

$$\sum_{j \in J} \sum_{t=s_{0j}+r_j}^{t_{max}-p_j} x_{0j}^t = 1 \tag{3.37}$$

$$\sum_{\substack{j \in J' \setminus \{i\}, \\ t-p_j \geq 0}} x_{ji}^t - \sum_{\substack{j \in J' \setminus \{i\}, \\ t+p_i+p_j+s_{ij} \leq t_{max}}} x_{ij}^{max(t+p_i+s_{ij},\ \ r_j+s_{ij})} = 0 \qquad \forall i \in J, t \in T \tag{3.38}$$

$$C_{max} \geq \sum_{i \in J \setminus \{j\}} \sum_{t=p_i}^{t_{max}-p_j} (t + p_j) x_{ij}^t \qquad \forall j \in J' \tag{3.39}$$

$$y_0 = I_0 \tag{3.40}$$

$$y_t = y_{t-1} + \sum_{i \in J'} \sum_{j \in J} q_j x_{ij}^t \qquad \forall t = \{1, ..., t_{max}\} \tag{3.41}$$

$$0 \leq y_t \leq I_C \qquad \forall t \in \{0, ..., t_{max}\} \tag{3.42}$$

$$C_{max} \geq 0 \tag{3.43}$$

$$x_{ij}^t \in \{0, 1\} \qquad \forall i \in J', j \in J', t \in T. \tag{3.44}$$

The objective function (3.35) minimizes the *makespan*. Constraints (3.36) ensure that each job in the set is processed exactly once. Constraint (3.37) defines the start of the scheduling process on a single machine, ensuring that it begins only after the release date of the first job and after the initial *setup*.

Constraints (3.38) enforce flow conservation by establishing that if a job $i \in J'$ is completed at time $t \in \{0, \ldots, t_{max}\}$, then the subsequent job $j \in J'$ must start at time $\max(t + p_i + s_{ij},\ r_j + s_{ij})$.

Constraints (3.39) compute the completion times of the jobs. Constraints (3.40) and (3.41) determine the inventory level at each time instant t. Constraints (3.42)

ensure that the minimum and maximum inventory levels are respected. Finally, constraints (3.43) and (3.44) ensure the non-negativity of completion times and define the variable domains, respectively.

3.3 Branch-and-cut algorithm

Let the $G = (J', A)$ be the complete and directed graph with each vertex $j \in J'$ representing a job and A being the set of arcs $\{(i, j) \mid i, j \in J'\}$. The set $\delta^+_{(i)} \subseteq A$ represent the arcs leaving vertex i. If the jobs in a subset $S \subseteq J'$ have a total demand $q(S)$, then at least $\lceil |q(S)|/I_C \rceil$ arcs must enter S. That can be expressed in terms of the following cuts:

$$\sum_{a \in \delta^+(S)} x_a \geq \left\lceil \frac{|q(S)|}{I_C} \right\rceil \qquad S \subseteq J' \tag{3.45}$$

The B&C algorithm adds the capacity cuts (3.45) to the arc indexed formulation presented in section 3.2.2. The cuts are separated only for nodes whose depth does not exceed 10, and the method used is the tabu search procedure suggested by Augerat et al. (1998) and also used by Chemla et al. (2013) and Bulhões et al. (2018).

The procedure is described in Algorithm 1. The separation routine aims at identifying violated cut sets by combining a greedy randomized construction phase with a local improvement phase based on tabu search.

For each value of the parameter *sizeThreshold*, which controls the maximum cardinality of the generated sets, the algorithm explores candidate cut sets rooted at each node of the graph. For a given root node i, the procedure starts with the singleton set $S = \{i\}$ and its corresponding cut value $q(S) = q_i$. The set S is then iteratively expanded by means of the greedy ExpandSet procedure (Algorithm 2). At each expansion step, all nodes $j \notin S$ are evaluated according to their marginal contribution $x(S{:}j)$ to the cut, then the node $j^\star$ with the highest contribution is selected and added to S, updating $q(S)$ accordingly. This process continues until no candidates remain or the size of S reaches the threshold defined by *sizeThreshold*.

Once the construction phase ends, the resulting set S is passed to the improvement phase implemented by ImproveSetTS (Algorithm 3). This phase applies a

tabu search to optimize the violation of the corresponding cut. Starting from the constructed set, the algorithm iteratively explores the neighborhood defined by admissible add and remove moves, selecting at each iteration the move that yields the largest improvement in violation while respecting tabu restrictions. After a fixed number of iterations $T_{\max} = 1000$, the best set S^* encountered during the search is returned.

If the improved set is nonempty and induces a violated inequality, a binary cut-set indicator vector is built and added to the pool of generated cuts. The overall procedure repeats this process for all nodes and threshold values, yielding a diversified collection of violated cut sets. This approach allows the separation algorithm to efficiently explore the solution space and identify high-quality cuts that significantly strengthen the linear relaxation.

Algorithm 1 Separation of Cut Sets

procedure SEPARATECUTSETS
  Initialize list of cut sets $\mathcal{C} \leftarrow \emptyset$
  for sizeThreshold $= 1$ to 5 **do**
    for each node i **do**
      $S \leftarrow \{i\}$; $q(S) \leftarrow q_i$
      $S \leftarrow$ EXPANDSET$(S, q(S), \text{sizeThreshold})$
      if IMPROVESETTS$(S, q(S))$ and $|S| > 0$ **then**
        Build cut-set indicator vector c from S
        Add c to $\mathcal{C}$
  return $\mathcal{C}$

Algorithm 2 ExpandSet procedure

procedure EXPANDSET$(S, q(S), \text{sizeThreshold})$
  while $|S| < \frac{n+1}{\text{sizeThreshold}}$ **do**
    Compute the candidate list $LC \leftarrow \{(j, x(S{:}j)) \mid j \notin S\}$
    if $LC = \emptyset$ **then**
      return S
    Select $j^\star \leftarrow \arg\max_{(j,x(S:j))\in LC} x(S{:}j)$
    $S \leftarrow S \cup \{j^\star\}$
    $q(S) \leftarrow q(S) + q_{j^\star}$
  return S

Algorithm 3 Improve set with tabu search

```
1: procedure ImproveSetTS(S, q(S))
2:     Initialize tabu list TL ← ∅
3:     S* ← S
4:     for t = 1 to T_max do
5:         Generate all admissible add/remove moves
6:         Select move m that maximizes violation improvement
7:         Apply m to obtain new S, update tabu list
8:         if violation(S) > violation(S*) then
9:             S* ← S
10:    return S*
```

3.4 Branch-cut-and-price

A branch-cut-and-price (BCP) algorithm was developed to solve the $1|r_j, s_{ij}, inv|C_{\max}$ problem by extending the branch-and-price (B&P) approach proposed in Morais et al. (2024) for the $1|r_j, s_{ij}|C_{\max}$ problem. This extension incorporates additional inventory-related capacity cuts (3.45), as well as lazy constraints to enforce inventory feasibility during the branch-and-bound process.

Considering the arc–time–indexed formulation defined in Section 3.2.4, the underlying B&P framework uses only the variables and constraints related to the scheduling decisions (3.35–3.39), omitting the inventory variables and constraints. The formulation is rewritten in terms of a directed acyclic graph $G = (V, A)$, where the vertex set V is defined as $V = R \cup O$, with the set $R = \{(j,t) : j \in J,\ t = \min_{i \in J' \setminus \{j\}}(s_{ij}) + r_j + p_j, \ldots, T\}$ containing vertices associated with the completion of job j at time t, where $\bar{s}_j = \min_{i \in J_0 \setminus \{j\}} s_{ij}$ and T is an upper bound on the makespan. The set $O = \{(0,t) : t = 0, \ldots, T\}$ represents the idle state of the machine. When a path reaches a vertex (j,t), job j has already been fully processed, possibly followed by idle time.

The arc sets are defined as follows:

$$
\begin{aligned}
A_1 = \{((i,t),(j,t+s_{ij}+p_j)) : (i,t) \in R,\ (j,t+s_{ij}+p_j) \in R,\\
t \geq r_j,\ i \neq j\},\\
A_2 = \{((0,t),(j,t+s_{0j}+p_j)) : (0,t) \in O,\ (j,t+s_{0j}+p_j) \in R,\ t \geq r_j\},\\
A_3 = \{((j,t),(0,T)) : (j,t) \in R\},\\
A_4 = \{((j,t),(j,t+1)) : (j,t) \in V,\ (j,t+1) \in V\}.
\end{aligned}
$$

Let $\delta^-(S)$ and $\delta^+(S)$ denote the sets of arcs entering and leaving a subset $S \subseteq V$, respectively. For each job $j \in J$, define $R_j = \{(i,t) \in R : i = j\}$ and let $A_j = \delta^-(R_j) \setminus A_4$ be the set of arcs associated with the processing of job j.

For an arc $a = ((i,t),(j,t+s_{ij}+p_j)) \in A$ representing the completion of job j, we define its cost as $c_a = t + s_{ij} + p_j$. For arcs in $A_3 \cup A_4$, we set $c_a = 0$. Let x_a be a nonnegative integer variable indicating the number of times arc $a \in A$ is used.

The arc-time-indexed formulation is given by:

$$\text{(F1)} \quad \min \alpha \tag{3.46}$$

$$\text{s.t.} \quad \sum_{a \in A_j} x_a = 1, \qquad \forall j \in J, \tag{3.47}$$

$$\sum_{a \in A_2} x_a = 1, \tag{3.48}$$

$$\sum_{a \in \delta^-(\{v\})} x_a - \sum_{a \in \delta^+(\{v\})} x_a = 0, \qquad \forall v \in V \setminus \{(0,0),(0,T)\}, \tag{3.49}$$

$$\alpha \geq \sum_{a \in A_j} c_a x_a, \qquad \forall j \in J, \tag{3.50}$$

$$x_a \in \{0,1\}, \qquad \forall a \in A \tag{3.51}$$

Constraints (3.47) ensure that each job is processed exactly once, while (3.48) enforces that exactly one job initiates the schedule. Flow conservation is imposed by (3.49). Constraints (3.50) define α as an upper bound on the completion time of every job, thus modeling the makespan objective.

Let $\mathcal{P}$ be the set of all paths in G from $(0,0)$ to $(0,T)$. For each path $P \in \mathcal{P}$, let $b_a^P \in \{0,1\}$ indicate whether arc a belongs to P, and let λ_P be a binary variable equal to 1 if path P is selected. The arc and path variables satisfy

$$x_a = \sum_{P \in \mathcal{P}} b_a^P \lambda_P. \tag{3.52}$$

Applying Dantzig–Wolfe decomposition to formulation (F1) yields the following equivalent path-based formulation:

$$\text{(F2)} \quad \min \alpha \tag{3.53}$$

$$\text{s.t.} \quad \sum_{P \in \mathcal{P}} \left(\sum_{a \in A_j} b_a^P \right) \lambda_P = 1, \qquad \forall j \in J, \tag{3.54}$$

$$\alpha \geq \sum_{a \in A_j} c_a \left(\sum_{P \in \mathcal{P}} b_a^P \lambda_P \right), \qquad \forall j \in J, \tag{3.55}$$

$$\sum_{P \in \mathcal{P}} \lambda_P = 1, \tag{3.56}$$

$$\lambda_P \in \{0, 1\}, \qquad \forall P \in \mathcal{P}, \tag{3.57}$$

At each node of the branch-and-bound tree, the linear relaxation of the master problem is solved by column generation. The pricing subproblem consists of finding a minimum reduced-cost path in the underlying acyclic network. To strengthen the linear relaxation, the algorithm employs the ng-path relaxation (Baldacci et al., 2011), which limits job repetition along paths through dynamically managed memory sets associated with arcs. These memories are iteratively augmented to forbid short and relevant cycles detected in the primal solutions.

The pricing problem is solved using a bidirectional labeling algorithm with dominance rules, enabling efficient enumeration of feasible ng-paths. To accelerate convergence, the column generation procedure is implemented in two stages: an initial heuristic phase that relaxes dominance conditions to quickly identify promising columns, followed by an exact phase that guarantees optimality.

After column generation converges at a node, a reduced-cost fixing procedure is executed to eliminate arcs from the pricing network that cannot belong to improving paths, thereby reducing the size of subsequent pricing problems. If the obtained solution is fractional, branching is performed on precedence variables that indicate whether one job immediately precedes another. A two-stage strong branching strategy is used, combining pseudocost information and partial column generation to select effective branching decisions. A detailed description of the underlying B&P framework can be found in Morais et al. (2024).

The resulting BCP algorithm integrates cut separation, as defined in Section 3.3, directly into the B&P framework,strengthening the linear relaxation through the inclusion of additional valid inequalities, thereby improving both solution quality and convergence behavior. Inventory constraints are enforced by means of *lazy constraints*, which are separated whenever an integer solution is found.

Given an integer solution, the corresponding path from the source node 0 to the sink node is reconstructed, defining a complete processing sequence of jobs

$$\pi = (\pi_1, \pi_2, \ldots, \pi_{|\pi|}).$$

Starting from the initial inventory level I_0, the inventory feasibility of the sequence is evaluated along its prefixes. After processing the first k jobs, the inventory level is given by

$$I_k = I_0 + \sum_{h=1}^{k} q_{\pi_h}, \tag{3.58}$$

where q_j denotes the inventory variation associated with job j. If, for any prefix $\Pi = \{\pi_1, \ldots, \pi_k\}$, the inventory level violates the capacity bounds,

$$I_k < 0 \quad \text{or} \quad I_k > I_c, \tag{3.59}$$

the solution is declared infeasible.

Whenever such a violating prefix Π is detected, a *prefix elimination cut* is generated to forbid the corresponding inventory-infeasible path. Let X_{ij} denote the binary decision variable indicating that job j is processed immediately after job i. The cut is defined as

$$\sum_{j \notin \Pi} X_{0j} \; + \; \sum_{i \in \Pi} \sum_{j \notin \Pi} X_{ij} \; + \; \sum_{i \in \Pi} X_{i,sink} \; \geq \; 2. \tag{3.60}$$

Constraint (3.60) ensures that the solution cannot form a path that starts at the source, processes exactly the jobs in Π consecutively, and then continues along the same infeasible sequence. Instead, it forces the path to enter or leave the prefix in a different manner, thereby eliminating the inventory-infeasible structure while preserving all potentially feasible job sequences.

Since these inequalities are violated only by integer solutions, they are added as lazy constraints and do not affect the linear relaxation. For computational efficiency, at most one violating prefix cut is generated per separation call.

By combining column generation, branching, and the dynamic separation of inventory capacity cuts, the proposed BCP algorithm effectively handles the inventory constraints while preserving the strengths of the original branch-and-price framework, leading to improved convergence and solution quality.

3.5 Heuristic methods

The heuristic approach proposed in this work is based on the ILS framework combined with a constructive procedure based on *Beam Search* (BS), as originally proposed in Morais et al. (2024) for the $1|r_j, s_{ij}|C_{max}$ problem. The original ILS-BS algorithm is adapted to address the additional characteristics of the $1|r_j, s_{ij}, inv|C_{max}$ problem by incorporating alternative perturbation strategies aimed at increasing solution diversification while ensuring feasibility with respect to inventory constraints.

3.5.1 ILS-BS

The pseudocode of the ILS-BS is presented in Algorithm 4. The procedure takes six parameters as input. Parameter I_R controls the number of restarts, while I_{ILS} defines the maximum number of consecutive iterations without improvement. Parameters w and N regulate the size of the search tree explored by the beam search procedure, control the size of the search tree explored by the BS procedure, defining the maximum number of child nodes generated from a single node and the maximum total number of nodes retained at each level, respectively. Parameter γ regulates the degree of randomness in the node selection process during the beam search, and parameter l defines the maximum number of neighborhood structures explored during the local search phase. At each restart, an initial solution is generated by the beam search procedure (line 4) and then iteratively improved through a combination of local search and perturbation steps (lines 7–13). The algorithm terminates by returning the best solution found over all restarts (lines 14–17).

Constructive procedure

The constructive procedure is an adaptation of the BS originally proposed in Velez-Gallego et al. (2016). The method performs a restricted enumeration of partial solutions organized in a tree structure, whose root corresponds to an empty sequence. New nodes are generated by appending jobs to the end of the current sequence. At each tree level, exactly one job is appended to each node, so that complete solutions are obtained at level n.

The enumeration is restricted in two ways: each node may generate at most w child nodes, and each tree level may contain no more than N nodes. The constructive version of the BS differs from the original procedure by introducing a randomness factor in the selection of the surviving nodes at each level, controlled by parameter γ.

Algorithm 4 ILS-BS

```
1: procedure ILS-BS(I_R, I_ILS, w, N, γ, l)
2:     f* ← ∞
3:     for i ← 1, ..., I_R do
4:         s ← BeamSearch(w, N, γ)
5:         s' ← s
6:         IterILS ← 0
7:         while IterILS < I_ILS do
8:             s ← LocalSearch(s)
9:             if f(s) < f(s') then
10:                s' ← s
11:                IterILS ← 0
12:            s ← Perturbation(s')
13:            IterILS ← IterILS + 1
14:        if f(s') < f(s*) then
15:            s* ← s'
16:            f* ← f(s')
17:    return s*
```

This modification allows the method to produce diversified solutions across different ILS iterations.

The pseudocode of the constructive BS is presented in Algorithm 5. In line 2, the root node is initialized with an empty sequence and added to the list of nodes at level Π_0 in line 3. From level 0 to level $n-1$, for each node in list Π_k, up to w child nodes are generated by fixing a job j from the set of unscheduled jobs $U(\pi)$. When the number of available jobs exceeds w, those that induce the largest idle time are discarded until only w candidates remain. Each selected job is then appended to the end of the sequence, represented by function $\beta(j, \pi)$, and the resulting nodes are added to list Π_{k+1} (lines 7–12).

If list Π_{k+1} contains more than N nodes, all nodes are evaluated and sorted in non-decreasing order according to a *makespan* lower bound (LB), computed by Equation 3.61, where $H(\pi)$ denotes the union of $U(\pi)$ and the last job in sequence π (line 14). A candidate list composed of the $\lfloor (1+\gamma)N \rfloor$ best-ranked nodes is then formed (line 15), and in lines 16–20 the list is reduced to N nodes selected at random from this candidate set.

$$LB(\pi) = \max\{C(\pi)\ ,\ \min_{j \in U(\pi)} \{r_j\}\} + \sum_{k \in U(\pi)} \min_{t \in H(\pi)} \{s_{tk}\} + \sum_{k \in U(\pi)} p_k \tag{3.61}$$

Finally, in lines 22–25, all nodes at level n are generated, corresponding to complete

Algorithm 5 Beam search

```
1: procedure BeamSearch(γ)
2:     π_0 ← {*, *, ..., *}
3:     Π_0 ← {π_0}
4:     for k ← 0, ..., n − 1 do
5:         Π_{k+1} ← {∅}
6:         for all i ∈ Π_k do
7:             Θ ← U(i)
8:             while |Θ| > w do
9:                 j* ← max{I(j, i)|j ∈ Θ}
10:                Θ ← Θ \ j*
11:            for all j ∈ Θ do
12:                Π_{k+1} ← Π_{k+1} ∪ β(j, i}
13:        if |Π_{k+1}| > N then
14:            Sort Π_{k+1} according to LB(π);
15:            Create a candidate list CL with the ⌊(1 + γ)N⌋ first nodes of Π_{k+1};
16:            Π_{k+1} ← {∅}
17:            while |Π_{k+1}| < N do
18:                Randomly choose a node i* ∈ CL;
19:                Π_{k+1} ← Π_{k+1} ∪ {i*};
20:                CL ← CL \ {i*};
21:    Π_k ← {∅}
22:    for all i ∈ Π_k do
23:        for all j ∈ U(i) do
24:            Π_k ← Π_k ∪ β(j, i}
25:    s* ← min{C(i)|i ∈ Π_k}
26:    return s*
```

solutions. The method returns the solution with the smallest *makespan*.

Local Search

The local search procedure follows the *Random Variable Neighborhood Descent* (RVND) paradigm (Mladenovic and Hansen, 1997; Subramanian et al., 2010). At the beginning, a set of neighborhood structures, referred to as *NL*, is constructed. One neighborhood is then randomly chosen from this set, and its associated moves are exhaustively examined and evaluated. When no improvement over the current incumbent solution is identified, the selected neighborhood is discarded from *NL*. On the other hand, whenever an improving solution is obtained, the neighborhood list is fully reset, allowing all neighborhood structures to be reconsidered in subsequent iterations.

Two types of neighborhood structures are employed within the RVND framework:

- *Swap*: this move exchanges the positions of two jobs in the sequence. Figure 3.2 illustrates an example of a swap move and the resulting sequence;
- *l-block insertion*: a block of l consecutive jobs is removed from its current position and reinserted elsewhere in the sequence. Figure 3.3 illustrates an example of a 1-block insertion move and the resulting sequence.

The local search adapts the efficient move evaluation scheme proposed in Morais et al. (2024) to address the inventory component of the problem. With this approach, each move can be assessed in amortized $\mathcal{O}(1)$ time, leading to a reduction in the neighborhood exploration complexity to $\mathcal{O}(n^2)$.

Under this scheme, the evaluation of a move is performed by concatenating subsequences of jobs. For each subsequence $\pi = (\pi_1, \pi_2, \ldots, \pi_{|\pi|})$ in a given solution, the following attributes are stored in an auxiliary data structure:

- $F(\pi)$ = first job in the subsequence;
- $L(\pi)$ = last job in the subsequence;
- $E(\pi)$ = earliest time at which processing can start without idle time;
- $D(\pi)$ = sum of processing times and setup times.
- $Q_s(\pi)$ = accumulated inventory level when finishing processing π;

- $Q_{min}(\pi)$ and $Q_{max}(\pi)$ = minimum and maximum inventory levels reached while processing the jobs in π, respectively.

For a unitary subsequence $\pi = (j)$, these attributes are initialized as follows:

- $F(\pi) = j$;
- $L(\pi) = j$;
- $E(\pi) = r_j$;
- $D(\pi) = p_j$;
- $Q_s(\pi) = q_j$;
- $Q_{min}(\pi) = \min(0, q_j)$;
- $Q_{max}(\pi) = \max(0, q_j)$.

Using the information stored in the auxiliary data structure, any subsequence π with $|\pi| > 1$ can be constructed by concatenating smaller subsequences. As a result, any local search move can be efficiently evaluated by applying the subsequence concatenation procedure described in Algorithm 6. Figures 3.2 and 3.3 illustrate how swap and l-block insertion moves, respectively, are evaluated through subsequence concatenation. Inventory infeasibilities are handled by monitoring the values of $Q_{\min}(\pi)$ and $Q_{\max}(\pi)$; whenever violations occur, the objective function value is penalized by adding the total magnitude of the violations multiplied by a penalty factor ρ.

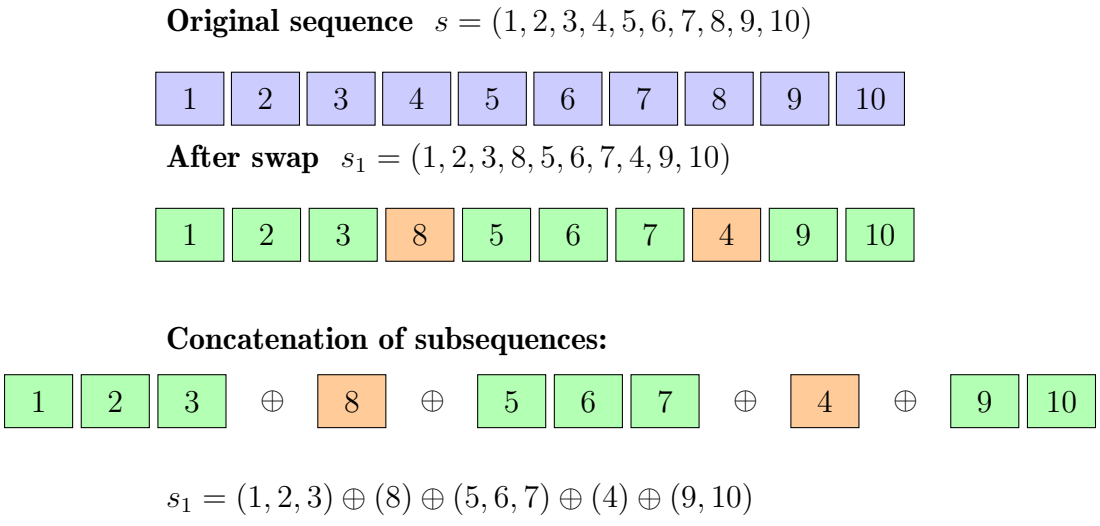

Figure 3.2: Evaluation of a swap move by concatenation of subsequences.

Algorithm 6 Subsequences concatenation $\pi^1 \oplus \pi^2$

procedure ConcatenateSubsequences(π^1, π^2)
  $Q_s(\pi^1 \oplus \pi^2) \leftarrow Q_s(\pi^1) + Q_s(\pi^2)$;
  $Q_{min}(\pi^1 \oplus \pi^2) \leftarrow \min(Q_{min}(\pi^1), Q_s(\pi^1) + Q_{min}(\pi^2))$;
  $Q_{max}(\pi^1 \oplus \pi^2) \leftarrow \max(Q_{max}(\pi^1), Q_s(\pi^1) + Q_{max}(\pi^2))$;
  $I \leftarrow 0$;
  $i, j \leftarrow L(\pi^1), F(\pi^2)$;
  if $C(\pi^1) < E(\pi^2)$ **then**
    if $C(\pi^1) < r_j$ **then**
      $I \leftarrow r_j - C(\pi^1)$;
    if $C(\pi^1) + I + s_{ij} < E(\pi^2)$ **then**
      $I \leftarrow E(\pi^2) - (C(\pi^1) + s_{ij})$;
  $E(\pi^1 \oplus \pi^2) \leftarrow E(\pi^1) + I$;
  $D(\pi^1 \oplus \pi^2) \leftarrow D(\pi^1) + D(\pi^2)$;
  return $\pi^1 \oplus \pi^2$

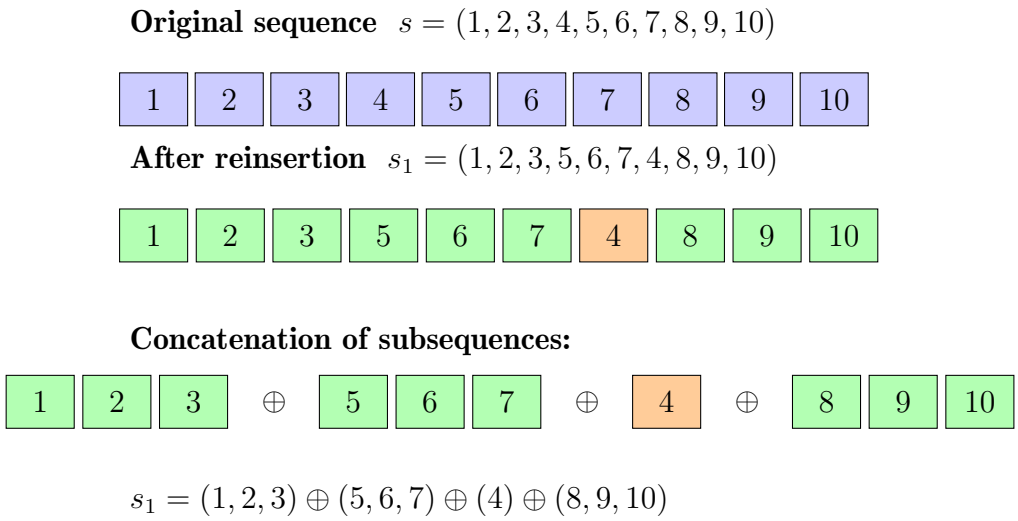

Figure 3.3: Evaluation of a reinsertion move by left-aligned concatenation of subsequences.

Perturbation

When the local search is unable to further improve the reference solution, a perturbation is performed on the best solution obtained in the current iteration. In the original ILS-BS algorithm proposed in Morais et al. (2024), a double-bridge move is used as the perturbation strategy. However, in this work, we explore alternative approaches based on a ruin-and-recreate method, which has shown promising results in various combinatorial optimization problems.

The first approach employs the standard ruin-and-recreate algorithm outlined in Algorithm 7. This method randomly removes a fixed number of jobs d from the current solution π and subsequently reinserts them in positions that minimize the makespan. The reinsertion is performed one job at a time, with each job being placed in the position that yields the best improvement in the objective function.

Algorithm 7 Standard ruin-and-recreate algorithm

```
1: procedure RUIN-AND-RECREATE(π, d)
2:     Randomly remove d jobs from π
3:     Create a set CL with the d jobs removed from π
4:     while CL ≠ ∅ do
5:         Randomly select a job j ∈ CL
6:         π ← reinsert(j, π)
7:         CL ← CL \ {j}
8:     return π
```

An alternative perturbation strategy explored in this work is based on the *Slack Induction by String Removals* (SISRs) method, originally proposed in Christiaens and Berghe (2020) for vehicle routing problems. This approach combines a ruin procedure that removes a set of jobs from the current solution with a recreate procedure that reinserts the removed jobs in a strategic manner.

The Ruin procedure (Algorithm 8) removes a set of jobs from the current solution π to create a partial solution π'. The removed jobs are stored in set $\mathcal{A}$. The procedure begins by selecting a seed job j uniformly at random from the solution (line 3). A string length l_t is then drawn uniformly from the interval $[1, \min(L_{\max}, |\pi|)]$ (line 4), where $L_{\max}$ is the maximum string length parameter. Depending on a random value and the parameter *splitRate*, either a standard string removal (lines 5–8) or a split string removal (lines 10–14) is performed. In the standard string removal, a contiguous subsequence of length l_t containing job j is selected and removed from π. In the split string removal, a window of size $l_t + m$ containing job j is selected, where m is determined based on the parameter *splitDepth*. A contiguous substring of length m is preserved inside the window, and the remaining l_t jobs are removed from π.

After the solution has been ruined, the Recreate procedure (Algorithm 9) reinserts the removed jobs from set $\mathcal{A}$ back into the partial solution π'. The jobs in $\mathcal{A}$ are first sorted using one of four strategies: random, demand, release, setup (line 2) to ensure a consistent reinsertion order. Under the random strategy, customers

are inserted without any predefined order. The demand and release strategies ranks customers in nondecreasing order of demand and release dates, respectively. Finally, the setup strategy uses the sum of setups to sort the jobs in nondecreasing order. The choice of ordering strategy for set $\mathcal{A}$ is governed by a weighted mechanism, assigning weights of four to random, four to demand, two to release, and one to setups. For each job j in $\mathcal{A}$, the procedure evaluates potential insertion positions in π'. A best position is determined based on the cost of the solution after insertion, considering a random factor controlled by the parameter *blinkRate*. Finally, job j is inserted at the best position found (line 11). Once all jobs have been reinserted, the reconstructed solution π' is returned.

Algorithm 8 SISRs Ruin procedure

```
1:  procedure RUIN(π)
2:      A ← ∅                                          ▷ Set of removed jobs
3:      Select a seed job j uniformly at random from π
4:      Draw l_t ∈ {1, ..., min(L_max, |π|)}
5:      if rand() > splitRate then                     ▷ Standard string removal
6:          Select uniformly a contiguous subsequence of length l_t containing j
7:          Remove all jobs in the subsequence from π and add them to A
8:      else                                           ▷ Split string removal
9:          m ← 1
10:         while m is feasible and rand() < splitDepth do
11:             m ← m + 1
12:         Select a window containing j of size l_t + m
13:         Preserve a contiguous substring of length m inside the window
14:         Remove the remaining l_t jobs and add them to A
15:     π' ← π
16:     return π', A
```

Algorithm 9 SISRs Recreate procedure

```
1: procedure RECREATE(π′, 𝒜)
2:     sort 𝒜
3:     for all j ∈ 𝒜 do
4:         bestCost ← +∞, bestPos ← −1
5:         evaluated ← ∅
6:         for i = 0 to |π′| do
7:             if rand() < 1 − blinkRate then
8:                 c ← Cost of inserting j at position i in π′
9:                 if c < bestCost then
10:                    bestCost ← c, bestPos ← i
11:        Insert j at position bestPos in π′
12:    return π′
```

Chapter 4

Computational experiments and results

This chapter presents the computational experiments and the results obtained from the different methods proposed for the $1|r_j, s_{ij}, inv|C_{\max}$ problem presented in this work. All methods were implemented in C++ and executed on a machine with processor Intel Core i7-12700, 16 GB RAM, and Ubuntu 24.04 operating system. The MILPs and the B&C algorithm used the CPLEX 22.11 solver with a time limit of 1 hour, limited to 4 threads and with tree memory usage limited to 4 GB, while the BCP algorithm was implemented over the BapCod platform (Vanderbeck et al., 2017) using the same parameters as in Morais et al. (2024) and time limit of 1 hour.

4.1 Instances generation

The computational experiments were conducted on a set of 900 instances specifically generated for the $1|r_j, s_{ij}, inv|C_{\max}$ problem. The instances vary in size and characteristics to comprehensively evaluate the performance of the proposed formulations and solution methods. The generation of the instances was based on the methods described in Ovacikt and Uzsoy (1994) and Davari et al. (2020). The set consists of ten groups of instances, varying in size with $n \in \{10, 20, 30, 40, 50, 60, 70, 80, 90, 100\}$. The processing times p_j were drawn from a uniform integer distribution in the range $[1, \alpha]$, with $\alpha \in \{10, 100\}$. The release dates were generated from a uniform integer distribution in the range $[0, \tau \sum_{j \in J} p_j]$, where $\tau \in \{0.2, 0.6, 1.0, 1.4, 1.8\}$. The setup times were drawn from a uniform distribution in the range $[1, 200]$.

Similarly, the absolute values $|q_j|$ of the inventory variations were obtained from a uniform integer distribution between 1 and 10. The inventory capacity I_c was randomly defined in the range $[10\eta, 20\eta]$, with $\eta \in \{1, 3, 5\}$. For each combination of the parameters α, τ, and η, 3 instances were generated, totaling 900 instances in the experimental set.

4.2 MILP results

Initially, computational experiments were conducted for all formulations on instances with $n \leq 50$, in order to assess which model was more promising for solving larger instances, where the required computational effort is significantly higher.

Table 4.1 presents a summary of the aggregated results obtained by each formulation, considering different instance sizes. The total number of optimally solved instances, the overall average gap, and the average execution time in cases where the optimal solution was reached were reported. It is observed that the increase in the number of jobs significantly impacts the problem's difficulty, with this effect being particularly pronounced in the position-based formulation. Although this approach solved all instances with $n = 10$, its performance deteriorates in other scenarios, evidenced by high gaps and a low rate of optimally solved instances.

Among the analyzed formulations, the arc-based one stood out for its overall best performance, being able to solve instances with up to 50 jobs. Furthermore, for all values of n, this formulation was the one that found the most optimal solutions, in addition to recording the lowest average resolution times compared to the other approaches.

4.2.1 Sensitivity analysis

In order to assess how instance characteristics influence the resolution complexity of the problem, an analysis of the resolution rate for each formulation was conducted, segmented by different instance sizes and combinations of the parameters α, τ, and η. The results of this analysis are summarized in Table 4.2.

It is observed that all formulations face greater difficulty in instances with $\alpha = 100$, a scenario in which jobs tend to have higher processing times and release dates.

In the specific case of the arc-time formulation, a progressive reduction in the resolu-

tion rate is observed as the value of τ increases. This behavior suggests a limitation of this approach in dealing with more spaced release dates, which results in higher makespan values and, consequently, a significant increase in the number of model variables.

Table 4.1: Performance of different MILP formulations for the $1|r_j, s_{ij}, inv|C_{\max}$ problem

Model	$n = 10$			$n = 20$			$n = 30$			$n = 40$			$n = 50$		
	#Opt	Gap (%)	Time (s)	#Opt	Gap (%)	Time (s)	#Opt	Gap (%)	Time (s)	#Opt	Gap (%)	Time (s)	#Opt	Gap (%)	Time (s)
Position	90	0.00	6.38	1	1730.54	474.83	0	5822.76	–	0	79640.91	–	0	18128.44	–
Arc-indexed	90	0.00	1.19	33	3.10	259.76	22	2.56	212.90	8	2.27	582.29	1	2.22	196.46
Arc-inventory	73	0.93	266.32	14	3.76	624.56	5	2.46	1355.72	1	2.25	1025.05	0	2.27	–
Arc-time	89	0.17	170.83	29	1.31	995.28	2	1.27	1426.42	0	1.49	–	0	7.84	–

Table 4.2: Percentage of instances solved for different models and configurations of parameters α, τ, η

(a) Position-indexed model

Parameters	**n**				
	10	20	30	40	50
$\alpha = 10$	100%	2%	0%	0%	0%
$\alpha = 100$	100%	0%	0%	0%	0%
$\tau = 0.2$	100%	0%	0%	0%	0%
$\tau = 0.6$	100%	6%	0%	0%	0%
$\tau = 1.0$	100%	0%	0%	0%	0%
$\tau = 1.4$	100%	0%	0%	0%	0%
$\tau = 1.8$	100%	0%	0%	0%	0%
$\eta = 1$	100%	0%	0%	0%	0%
$\eta = 3$	100%	0%	0%	0%	0%
$\eta = 5$	100%	3%	0%	0%	0%

(b) Arc-indexed model

Parameters	**n**				
	10	20	30	40	50
$\alpha = 10$	100%	42%	38%	13%	2%
$\alpha = 100$	100%	31%	11%	4%	0%
$\tau = 0.2$	100%	6%	0%	0%	0%
$\tau = 0.6$	100%	17%	6%	0%	0%
$\tau = 1.0$	100%	44%	17%	6%	0%
$\tau = 1.4$	100%	50%	50%	11%	6%
$\tau = 1.8$	100%	67%	50%	28%	0%
$\eta = 1$	100%	30%	23%	10%	0%
$\eta = 3$	100%	30%	27%	7%	3%
$\eta = 5$	100%	50%	23%	10%	0%

(c) Arc-inventory model

Parameters	**n**				
	10	20	30	40	50
$\alpha = 10$	89%	18%	7%	2%	0%
$\alpha = 100$	73%	13%	4%	0%	0%
$\tau = 0.2$	100%	6%	0%	0%	0%
$\tau = 0.6$	61%	0%	0%	0%	0%
$\tau = 1.0$	83%	17%	0%	0%	0%
$\tau = 1.4$	78%	22%	6%	0%	0%
$\tau = 1.8$	83%	33%	22%	6%	0%
$\eta = 1$	100%	30%	10%	3%	0%
$\eta = 3$	80%	7%	0%	0%	0%
$\eta = 5$	63%	10%	7%	0%	0%

(d) Arc-time model

Parameters	**n**				
	10	20	30	40	50
$\alpha = 10$	100%	40%	4%	0%	0%
$\alpha = 100$	98%	24%	0%	0%	0%
$\tau = 0.2$	100%	72%	11%	0%	0%
$\tau = 0.6$	100%	50%	0%	0%	0%
$\tau = 1.0$	100%	22%	0%	0%	0%
$\tau = 1.4$	100%	17%	0%	0%	0%
$\tau = 1.8$	94%	0%	0%	0%	0%
$\eta = 1$	97%	3%	0%	0%	0%
$\eta = 3$	100%	50%	0%	0%	0%
$\eta = 5$	100%	43%	7%	0%	0%

4.3 Branch-and-cut results

Since the arc-based formulation exhibited the best performance among the exact methods evaluated, the B&C algorithm was developed based on this formulation. The results obtained by the B&C algorithm are summarized in Table 4.3, which reports the average gap, average total solution time and the number of solved instances for different instance sizes. For comparison purposes, the corresponding results of the arc-indexed MILP formulation are also included.

The results show that, similarly to the arc-indexed formulation, the performance of the B&C algorithm deteriorates as the instance size increases. While the algorithm is able to optimally solve most instances with $n = 10$, its effectiveness decreases for larger instances, and no optimal solutions are obtained for instances with more than 30 jobs within the imposed time limit. Nevertheless, the B&C algorithm consistently achieves slightly lower average gaps than the arc-indexed MILP for almost all instance sizes, highlighting the benefits of incorporating valid inequalities within a B&C framework.

Table 4.3: Branch-and-Cut vs. MILP results aggregated by instance size

n	**B&C**			**MILP**		
	Avg Gap	Avg Time	#Opt	Avg Gap	Avg Time	#Opt
10	0.00	26.06	90	0.00	1.19	90
20	3.36	2919.75	18	3.10	539.88	36
30	2.50	3322.75	8	2.56	827.34	22
40	2.06	3600.15	0	2.27	1772.95	8
50	2.03	3600.20	0	2.22	2088.61	1
60	1.41	3600.25	0	1.51	2340.88	1
70	1.45	3600.42	0	1.48	2641.30	0
80	1.33	3600.60	0	1.32	3031.82	0
90	1.17	3601.21	0	1.16	3172.32	0
100	1.17	3561.58	0	1.17	3493.65	0

The comparison between the results obtained by the B&C algorithm and those achieved by the arc-indexed MILP formulation is further detailed in Tables 4.4 – 4.8. The comparison is made for different instance values α, τ, and η. Table 4.4 shows the results for different values of α. It is observed that both methods face

greater difficulty in instances with $\alpha = 100$, a scenario in which jobs tend to have higher processing times and release dates.

Tables 4.5 and 4.6 present the results for different values of τ. It is noted that both methods exhibit better performance in instances with higher values of τ, which correspond to sparser release dates. This behavior is particularly pronounced in the case of the B&C algorithm, which achieves a higher number of optimally solved instances and lower average gaps in these scenarios.

Finally, Table 4.8 summarizes the results for different values of η. The results indicate that both methods tend to obtain slightly lower gaps for higher inventory capacities.

Table 4.4: Branch-and-Cut vs. MILP results for different values of α

n	α	B&C							MILP		
		Root Gap	Root Time	Nodes	Avg. Gap	Avg. Time	#Cuts	#Opt	Avg. Gap	Avg. Time	#Opt
10	10	9.87	0.00	8	0.00	8.82	46.51	45	0.00	0.46	45
10	100	8.47	0.00	8	0.00	43.30	64.73	45	0.00	1.92	45
20	10	3.72	0.01	365983	3.31	2705.54	63.31	12	3.04	458.95	21
20	100	4.25	0.00	389146	3.41	3133.96	94.75	6	3.16	620.80	15
30	10	2.03	0.02	182670	1.98	3199.11	32.93	6	2.11	647.42	17
30	100	3.20	0.02	175470	3.02	3446.39	71.00	2	3.01	1007.25	5
40	10	1.39	0.05	117937	1.36	3600.12	9.87	0	1.49	2032.14	6
40	100	2.85	0.05	102188	2.76	3600.18	46.67	0	3.05	1513.76	2
50	10	1.40	0.10	73543	1.40	3600.21	10.04	0	1.68	2376.91	1
50	100	2.71	0.11	61637	2.66	3600.19	31.04	0	2.75	1800.31	0
60	10	0.70	0.22	46679	0.70	3600.25	6.91	0	0.73	2062.62	1
60	100	2.15	0.21	42898	2.12	3600.25	33.78	0	2.29	2619.13	0
70	10	0.78	0.36	32764	0.78	3600.47	8.40	0	0.82	2532.12	0
70	100	2.13	0.40	28978	2.11	3600.36	32.40	0	2.14	2750.48	0
80	10	0.84	0.62	22559	0.84	3600.49	9.84	0	0.84	2978.18	0
80	100	1.83	0.74	21018	1.81	3600.71	25.44	0	1.79	3085.45	0
90	10	0.58	0.99	16102	0.58	3601.27	8.22	0	0.58	3325.19	0
90	100	1.76	1.17	14746	1.76	3601.15	18.09	0	1.74	3019.44	0
100	10	0.76	1.46	12057	0.76	3601.28	9.33	0	0.76	3445.97	0
100	100	1.59	1.64	10920	1.58	3521.88	20.96	0	1.57	3541.32	0

4.4 Branch-cut-and-price results

The BCP results are summarized and compared with those obtained by the MILP and B&C approaches in Table 4.9. Overall, the BCP method solved 201 instances to optimality, which is higher than the number achieved by the other approaches, particularly for instances with $n = 20$ and $n = 30$. However, BCP did not reach the optimal solution for two instances with $n = 10$, whereas both the MILP and

Table 4.5: Branch-and-Cut vs. MILP results for different values of τ ($n \leq 50$)

n	τ	B&C							MILP		
		Root Gap	Root Time	Nodes	Avg. Gap	Avg. Time	#Cuts	#Opt	Avg. Gap	Avg. Time	#Opt
10	0.2	24.60	0.00	8	0.00	10.42	69.22	18	0.00	0.54	18
10	0.6	13.69	0.00	8	0.00	27.24	66.67	18	0.00	1.51	18
10	1.0	2.28	0.00	8	0.00	26.49	43.50	18	0.00	1.13	18
10	1.4	2.34	0.00	8	0.00	42.01	45.67	18	0.00	1.93	18
10	1.8	2.95	0.00	8	0.00	24.14	53.06	18	0.00	0.84	18
20	0.2	12.99	0.01	382671	11.12	3600.29	119.94	0	9.43	1025.94	2
20	0.6	3.90	0.00	488670	3.71	3412.28	74.06	1	4.23	331.77	4
20	1.0	1.60	0.00	435171	1.20	3270.84	58.67	2	0.94	734.79	8
20	1.4	0.67	0.01	411735	0.28	2895.90	85.61	4	0.22	323.78	9
20	1.8	0.56	0.01	156660	0.32	1318.57	54.65	11	0.27	233.32	13
30	0.2	9.08	0.02	148431	8.62	3600.23	112.89	0	8.40	582.91	0
30	0.6	2.50	0.01	187517	2.50	3600.20	33.50	0	2.47	907.73	1
30	1.0	0.90	0.01	208347	0.86	3600.17	42.56	0	0.81	1240.95	3
30	1.4	0.25	0.01	190136	0.20	3085.98	40.67	3	0.19	681.36	9
30	1.8	0.36	0.02	160921	0.33	2727.15	30.22	5	0.35	712.88	9
40	0.2	7.20	0.06	77769	7.02	3600.14	89.17	0	6.88	602.82	0
40	0.6	2.60	0.05	94975	2.55	3600.20	6.89	0	2.53	1703.59	0
40	1.0	0.42	0.05	123557	0.41	3600.13	12.83	0	0.46	2347.31	1
40	1.4	0.17	0.05	128903	0.13	3600.09	18.56	0	0.13	2569.44	2
40	1.8	0.22	0.04	125108	0.20	3600.16	13.89	0	0.23	1913.68	5
50	0.2	7.89	0.13	45984	7.76	3600.26	64.11	0	7.67	903.08	0
50	0.6	1.66	0.10	63331	1.66	3600.18	7.06	0	1.73	2063.36	0
50	1.0	0.39	0.11	72688	0.39	3600.24	8.50	0	0.44	2166.90	0
50	1.4	0.27	0.09	75812	0.27	3600.12	10.56	0	0.33	2592.05	1
50	1.8	0.08	0.09	80132	0.08	3600.21	12.50	0	0.09	2842.80	0

Table 4.6: Branch-and-Cut vs. MILP results for different values of τ ($n > 50$)

n	τ	B&C							MILP		
		Root Gap	Root Time	Nodes	Avg. Gap	Avg. Time	#Cuts	#Opt	Avg. Gap	Avg. Time	#Opt
60	0.2	5.43	0.27	30714	5.37	3600.38	71.89	0	5.32	1373.12	0
60	0.6	1.23	0.20	44558	1.23	3600.22	7.28	0	1.46	2340.93	0
60	1.0	0.16	0.20	48477	0.16	3600.16	8.22	0	0.11	2703.60	0
60	1.4	0.19	0.18	48389	0.19	3600.29	7.83	0	0.20	2611.90	0
60	1.8	0.10	0.21	51805	0.10	3600.20	6.50	0	0.11	2733.06	1
70	0.2	5.28	0.50	21413	5.24	3600.80	67.33	0	5.20	1968.75	0
70	0.6	1.33	0.35	28958	1.33	3600.48	7.94	0	1.33	2557.47	0
70	1.0	0.45	0.36	33920	0.45	3600.18	10.06	0	0.51	2530.15	0
70	1.4	0.16	0.33	33797	0.16	3600.30	8.28	0	0.16	2748.23	0
70	1.8	0.04	0.37	36266	0.04	3600.32	8.39	0	0.04	3439.97	0
80	0.2	5.58	0.83	15164	5.55	3600.60	53.17	0	5.50	2624.98	0
80	0.6	0.80	0.65	20155	0.80	3601.07	7.94	0	0.79	2919.05	0
80	1.0	0.21	0.72	22817	0.21	3600.46	7.44	0	0.21	3061.03	0
80	1.4	0.04	0.58	25429	0.04	3600.45	12.22	0	0.04	3044.11	0
80	1.8	0.04	0.62	25377	0.04	3600.42	7.44	0	0.04	3509.92	0
90	0.2	4.47	1.41	11315	4.45	3600.99	32.28	0	4.42	3141.67	0
90	0.6	0.94	1.06	14502	0.94	3601.10	9.78	0	0.94	2646.04	0
90	1.0	0.19	0.97	17089	0.19	3600.85	8.94	0	0.19	3315.90	0
90	1.4	0.17	0.96	17322	0.17	3601.44	7.00	0	0.17	3284.23	0
90	1.8	0.10	0.99	16892	0.10	3601.66	7.78	0	0.10	3473.73	0
100	0.2	5.35	2.10	8162	5.34	3401.05	42.56	0	5.32	3369.53	0
100	0.6	0.45	1.37	11266	0.45	3601.65	8.28	0	0.45	3378.36	0
100	1.0	0.19	1.48	12258	0.19	3601.19	10.06	0	0.19	3560.23	0
100	1.4	0.03	1.40	13174	0.03	3601.99	5.61	0	0.03	3559.99	0
100	1.8	0.06	1.39	12583	0.06	3602.01	9.22	0	0.06	3600.14	0

Table 4.7: Branch-and-Cut vs. MILP results for different values of η ($n \leq 50$)

n	η	B&C							MILP		
		Root Gap	Root Time	Nodes	Avg. Gap	Avg. Time	#Cuts	#Opt	Avg. Gap	Avg. Time	#Opt
10	1	14.78	0.00	8	0.00	12.13	79.43	30	0.00	0.80	30
10	3	5.73	0.00	8	0.00	52.50	46.80	30	0.00	2.07	30
10	5	7.00	0.00	8	0.00	13.54	40.63	30	0.00	0.70	30
20	1	5.18	0.01	369002	4.19	2948.87	163.13	6	3.53	812.44	12
20	3	4.09	0.00	392084	3.64	3140.35	39.86	4	3.62	342.02	9
20	5	2.69	0.00	371706	2.26	2670.24	32.27	8	2.12	446.77	15
30	1	3.52	0.02	176117	3.33	3441.21	113.00	2	3.35	902.67	7
30	3	2.39	0.01	178833	2.30	3265.22	28.90	3	2.51	662.31	8
30	5	1.94	0.01	182260	1.88	3261.81	14.00	3	1.88	920.65	7
40	1	2.81	0.06	110657	2.70	3600.13	52.23	0	2.96	1848.50	3
40	3	1.73	0.05	109475	1.70	3600.16	21.17	0	1.87	1741.71	2
40	5	1.83	0.04	110056	1.79	3600.14	11.40	0	1.98	1726.94	3
50	1	2.76	0.14	66597	2.71	3600.21	24.10	0	2.85	2118.53	0
50	3	1.85	0.09	67782	1.83	3600.22	26.17	0	2.09	1825.24	1
50	5	1.57	0.08	68391	1.56	3600.17	11.37	0	1.77	2253.24	0

Table 4.8: Branch-and-Cut vs. MILP results for different values of η ($n > 50$)

n	η	B&C							MILP		
		Root Gap	Root Time	Nodes	Avg. Gap	Avg. Time	#Cuts	#Opt	Avg. Gap	Avg. Time	#Opt
60	1	1.98	0.31	44178	1.96	3600.28	22.10	0	1.99	2393.43	0
60	3	1.20	0.18	46308	1.19	3600.20	25.67	0	1.30	2909.29	0
60	5	1.08	0.16	43879	1.07	3600.27	13.27	0	1.18	1705.70	1
70	1	1.95	0.55	30970	1.94	3600.48	20.20	0	1.98	2527.76	0
70	3	1.18	0.33	30928	1.18	3600.38	29.87	0	1.21	2707.98	0
70	5	1.23	0.27	30714	1.22	3600.38	11.13	0	1.26	2691.92	0
80	1	1.71	1.00	21840	1.70	3600.69	16.50	0	1.68	2933.64	0
80	3	1.14	0.60	21794	1.14	3600.50	23.97	0	1.13	2883.43	0
80	5	1.14	0.45	21732	1.14	3600.61	12.47	0	1.14	3278.39	0
90	1	1.52	1.51	15581	1.51	3601.29	14.73	0	1.50	3092.96	0
90	3	0.97	0.96	15536	0.96	3601.40	16.37	0	0.96	3324.53	0
90	5	1.04	0.76	15154	1.04	3600.94	8.37	0	1.04	3099.46	0
100	1	1.84	2.18	11207	1.83	3481.26	14.97	0	1.83	3532.27	0
100	3	0.83	1.42	11558	0.83	3601.55	20.60	0	0.83	3364.06	0
100	5	0.86	1.04	11700	0.86	3601.92	9.87	0	0.86	3584.61	0

B&C methods solved all instances of this size. Regarding solution quality, BCP achieved the lowest average gaps for instances with $n \in \{20, 30, 40\}$, which coincide with the instance sizes for which it solved a substantially larger number of instances compared to the MILP and B&C approaches.

Table 4.9: BCP vs. Branch-and-Cut vs. MILP results aggregated by instance size

n	BCP			B&C			MILP		
	Avg Gap	Avg Time	#Opt	Avg Gap	Avg Time	#Opt	Avg Gap	Avg Time	#Opt
10	0.18	9.19	88	0.00	26.06	90	0.00	1.19	90
20	0.21	891.20	69	3.36	2919.75	18	3.10	539.88	36
30	0.74	2031.33	34	2.50	3322.75	8	2.56	827.34	22
40	1.42	2747.22	12	2.06	3600.15	0	2.27	1772.95	8
50	2.82	3600.00	0	2.03	3600.20	0	2.22	2088.61	1
60	2.75	3512.50	1	1.41	3600.25	0	1.51	2340.88	1
70	3.63	3600.00	0	1.45	3600.42	0	1.48	2641.30	0
80	5.35	3600.00	0	1.33	3600.60	0	1.32	3031.82	0
90	7.23	3600.00	0	1.17	3601.21	0	1.16	3172.32	0
100	11.89	3600.00	0	1.17	3561.58	0	1.17	3493.65	0

Tables 4.10 – 4.14 summarize the average results obtained by the BCP algorithm for different instance sizes and parameters. Table 4.10 presents the average performance of the BCP algorithm for varying instance sizes (n) and values of α. The results indicate that the algorithm performs better for instances with $\alpha = 100$, achieving lower average gaps and solving more instances optimally compared to instances with $\alpha = 10$.

The average performance of the BCP algorithm for different values of τ is reported

Table 4.10: Average BCP performance by n and α

n	α	Root Gap	Final Gap	Root Time	Nodes	Final Time	#Opt	Improved UBs
10	10	0.81	0.11	2.57	10.38	9.48	44	0
10	100	0.81	0.25	3.08	5.93	8.90	44	0
20	10	0.66	0.11	134.37	15.42	904.45	34	1
20	100	0.77	0.30	137.70	10.02	877.94	35	0
30	10	2.03	1.02	234.33	20.91	1944.71	15	1
30	100	0.83	0.46	803.62	9.29	2117.96	19	0
40	10	3.98	1.95	505.93	21.53	2598.69	6	3
40	100	1.09	0.88	1805.51	8.71	2895.75	6	1
50	10	4.96	3.60	1864.38	6.71	3604.12	0	0
50	100	2.15	2.04	3392.09	3.48	3615.29	0	0
60	10	3.33	2.72	2362.14	5.78	3603.25	0	0
60	100	2.78	2.78	3424.99	2.18	3425.00	1	1
70	10	6.87	4.74	3427.02	2.60	3601.27	0	0
70	100	2.52	2.52	3603.89	2.05	3603.90	0	0
80	10	8.47	8.47	3412.98	2.05	3601.58	0	0
80	100	2.22	2.22	3611.32	1.46	3611.33	0	0
90	10	12.29	12.29	3601.41	1.46	3601.42	0	0
90	100	2.17	2.17	3605.89	1.31	3605.89	0	0
100	10	14.07	14.07	3601.56	1.00	3601.57	0	0
100	100	9.71	9.71	3603.25	1.00	3603.25	0	0

in Tables 4.11 and 4.12. The results indicate that, in most cases, higher values of τ are associated with lower average gaps, suggesting improved solution quality as the dispersion of release dates increases. However, the number of instances solved to optimality is generally higher for smaller values of τ. This indicates that, although larger values of τ tend to facilitate the construction of high-quality solutions, they do not necessarily lead to faster convergence to optimality. Overall, these results suggest a trade-off between solution quality and convergence behavior across different levels of release date dispersion.

Table 4.13 and Table 4.14 present the average performance of the BCP algorithm for varying values of η. The results indicate that higher values of η generally lead to improved performance, suggesting that the algorithm benefits from higher inventory capacity, a scenario in which feasible solutions are easier to find.

4.5 Heuristic results

The heuristic algorithms were implemented in `C++` and executed under the same computational environment as the exact methods. The proposed heuristics are denoted as ILS-SISRs and ILS-RR, each corresponding to a variant of the ILS framework

Table 4.11: Average BCP performance by $n \leq 50$ and τ

n	τ	Root Gap (%)	Final Gap (%)	Root Time	Nodes	Final Time	#Opt	Improved UBs
10	0.2	0.66	0.00	0.22	1.56	0.23	18	0
10	0.6	1.27	0.27	1.00	11.00	4.47	17	0
10	1.0	0.46	0.00	1.52	8.78	9.29	18	0
10	1.4	0.65	0.00	4.21	6.33	7.01	18	0
10	1.8	1.01	0.63	7.16	13.11	24.93	17	0
20	0.2	1.53	0.00	18.75	12.67	41.47	18	1
20	0.6	1.27	0.43	93.33	16.88	398.12	15	0
20	1.0	0.31	0.20	56.99	14.25	787.50	13	0
20	1.4	0.13	0.12	130.33	9.35	1428.22	12	0
20	1.8	0.30	0.28	369.28	10.67	1791.60	11	0
30	0.2	3.23	1.78	666.97	17.56	1324.74	11	1
30	0.6	0.42	0.21	165.83	12.63	1708.39	11	0
30	1.0	0.63	0.16	466.27	13.00	2442.61	9	0
30	1.4	0.06	0.03	534.54	15.00	3382.27	2	0
30	1.8	0.26	0.23	1642.38	8.33	3379.39	1	0
40	0.2	4.07	2.28	1084.36	19.00	2102.47	10	4
40	0.6	0.62	0.37	627.31	12.45	3339.63	2	0
40	1.0	0.64	0.60	3474.35	3.00	3600.00	0	0
40	1.4	0.31	0.31	1796.11	5.00	3600.00	0	0
40	1.8	0.17	0.17	3374.44	3.00	3600.00	0	0
50	0.2	6.41	5.03	2459.52	6.00	3600.00	0	0
50	0.6	2.71	2.44	2181.78	5.50	3600.00	0	0
50	1.0	0.30	0.30	3428.68	3.25	3600.00	0	0
50	1.4	0.21	0.21	3600.00	3.00	3600.00	0	0
50	1.8	0.09	0.09	3600.00	1.00	3600.00	0	0

Table 4.12: Average BCP performance by $n > 50$ and τ

n	τ	Root Gap	Final Gap	Root Time	Nodes	Final Time	#Opt	Improved UBs
60	0.2	4.19	3.57	2193.51	6.44	3434.62	1	1
60	0.6	7.60	7.60	3600.00	2.00	3600.00	0	0
60	1.0	0.15	0.15	3600.00	2.33	3600.00	0	0
60	1.4	0.11	0.11	3600.00	1.00	3600.00	0	0
60	1.8	0.12	0.12	3600.00	1.00	3600.00	0	0
70	0.2	9.41	6.75	3386.02	3.50	3600.00	0	0
70	0.6	1.55	1.55	3600.00	2.27	3600.00	0	0
70	1.0	4.40	4.40	3600.00	1.00	3600.00	0	0
70	1.4	0.07	0.07	3600.00	1.00	3600.00	0	0
70	1.8	0.10	0.10	3600.00	1.00	3600.00	0	0
80	0.2	7.90	7.90	3361.83	2.75	3600.00	0	0
80	0.6	9.09	9.09	3600.00	1.00	3600.00	0	0
80	1.0	0.10	0.10	3600.00	1.00	3600.00	0	0
80	1.4	0.05	0.05	3600.00	1.00	3600.00	0	0
80	1.8	8.34	8.34	3600.00	1.00	3600.00	0	0
90	0.2	8.20	8.20	3600.00	1.67	3600.00	0	0
90	0.6	9.24	9.24	3600.00	1.00	3600.00	0	0
90	1.0	0.07	0.07	3600.00	1.00	3600.00	0	0
90	1.4	0.02	0.02	3600.00	1.00	3600.00	0	0
100	0.2	19.88	12.57	3600.00	1.00	3600.00	0	0
100	0.6	1.27	1.27	3600.00	1.00	3600.00	0	0
100	1.0	5.57	5.57	3600.00	1.00	3600.00	0	0

Table 4.13: Average BCP performance by $n \leq 50$ and η

n	η	Root Gap (%)	Final Gap (%)	Root Time	Nodes	Final Time	#Opt	Improved UBs
10	1	1.83	0.38	5.70	13.67	20.93	29	0
10	3	0.54	0.16	1.48	8.67	5.10	29	0
10	5	0.06	0.00	1.29	2.13	1.53	30	0
20	1	2.04	0.56	209.94	24.11	1843.54	15	0
20	3	0.18	0.06	148.15	8.72	602.76	25	0
20	5	0.05	0.03	57.80	6.33	312.91	29	1
30	1	2.98	1.77	1247.69	9.63	3364.52	2	0
30	3	0.90	0.28	272.81	22.16	1603.12	15	0
30	5	0.07	0.01	201.21	10.16	1176.52	17	1
40	1	5.78	3.23	1960.21	6.83	3600.00	0	0
40	3	0.59	0.23	274.04	27.89	1777.29	5	2
40	5	0.53	0.45	1301.06	11.53	2696.60	7	2
50	1	4.93	4.26	2985.43	4.12	3600.00	0	0
50	3	2.26	1.95	3039.97	3.60	3600.00	0	0
50	5	1.47	0.61	2129.84	7.22	3600.00	0	0

Table 4.14: Average BCP performance by $n > 50$ and η

n	η	Root Gap (%)	Final Gap (%)	Root Time	Nodes	Final Time	#Opt	Improved UBs
60	1	6.42	5.92	3115.36	2.86	3600.00	0	0
60	3	0.40	0.38	2067.49	7.00	3193.45	1	1
60	5	1.09	0.79	3122.19	3.46	3600.00	0	0
70	1	7.61	6.21	3580.09	1.94	3600.00	0	0
70	3	2.74	1.31	3418.68	2.82	3600.00	0	0
70	5	2.38	2.13	3519.63	2.38	3600.00	0	0
80	1	9.89	9.89	3474.41	1.40	3600.00	0	0
80	3	3.92	3.92	3600.00	1.67	3600.00	0	0
80	5	1.62	1.62	3313.01	3.00	3600.00	0	0
90	1	16.35	16.35	3600.00	1.44	3600.00	0	0
90	3	2.10	2.10	3600.00	1.00	3600.00	0	0
90	5	2.74	2.74	3600.00	1.75	3600.00	0	0
100	1	24.21	13.65	3600.00	1.00	3600.00	0	0
100	3	4.58	4.58	3600.00	1.00	3600.00	0	0
100	5	4.40	4.40	3600.00	1.00	3600.00	0	0

with its respective perturbation strategy. Since the heuristics are non-deterministic, each instance was executed 10 times, and the reported gaps were computed based on the average solution values obtained across these runs.

4.5.1 Parameter tuning

As defined in Chapter 3, the ILS-based algorithms depend on two parameters: the number of restarts (I_R) and the number of iterations without improvement (I_{ILS}). The beam search algorithm used to generate the initial solution for the ILS-based methods depends on three parameters: the maximum number of child nodes (w), the maximum number of nodes per level (N) and the weight associated with the survival node selection criteria (α). The values assigned to these parameters are the same as those used in Morais et al. (2024), i.e., $I_R = 10$, $I_{ILS} = 100$, $w = 5$, $N = 100$, $\alpha = 0.01$.

Initially, experiments were conducted to tune the parameters l and L_{max} of the ILS-SISRs algorithm. The parameter l defines the maximum block size considered in the *l-block-insertion* neighborhood, while the parameter L_{max} determines the maximum string size to be removed, which affects the intensity of the perturbation procedure. The values tested for l were in the range (0, 17), with 0 meaning that no block insertion move is performed. While for L_{max} the values $\{2, 5, 10, 15, 30\}$ were evaluated. The remaining SISRs parameters were kept constant during these experiments, with *splitRate* = 0.5 and *splitDepth* = 0.01 as defined in Christiaens and Berghe (2020).

The experiments were conducted on a subset of 30 instances from the experimental set, with sizes ranging from 30 to 100 jobs. The results are shown in Table 4.15, which presents the average gaps and the number of infeasible solutions (in parentheses) obtained for each combination of l and L_{max}. Based on these experimentes the values of $l = 15$ and $L_{max} = 5$ were selected for the final configuration of the ILS-SISRs algorithm. The same values were used for the ILS-RR algorithm, given the similarity between the two methods and the results obtained in Guerra et al. (2024), in which a ruin-and-recreate procedure was tuned to the same value.

Table 4.15: Average gaps and number of infeasible solutions (in parentheses) for different values of l and L_{max} in the ILS-SISRs algorithm.

l	$L_{\max}$				
	2	5	10	15	30
0	8.63 (56)	9.08 (48)	11.60 (50)	14.24 (49)	17.14 (53)
1	2.03 (15)	1.96 (1)	2.31 (0)	2.60 (1)	3.33 (1)
2	1.23 (3)	1.05 (0)	1.23 (0)	1.35 (0)	1.77 (0)
3	0.82 (0)	0.68 (0)	0.68 (0)	0.79 (0)	1.05 (0)
4	0.62 (0)	0.52 (0)	0.56 (0)	0.66 (0)	0.86 (0)
5	0.58 (0)	0.47 (0)	0.49 (0)	0.59 (0)	0.75 (0)
6	0.52 (0)	0.45 (0)	0.52 (0)	0.57 (0)	0.74 (0)
7	0.56 (0)	0.45 (0)	0.48 (0)	0.57 (0)	0.72 (0)
8	0.50 (0)	0.42 (0)	0.47 (0)	0.54 (0)	0.68 (0)
9	0.50 (0)	0.44 (0)	0.49 (0)	0.53 (0)	0.72 (0)
10	0.48 (0)	0.45 (0)	0.49 (0)	0.53 (0)	0.68 (0)
11	0.51 (0)	0.44 (0)	0.47 (0)	0.52 (0)	0.73 (0)
12	0.50 (0)	0.43 (0)	0.45 (0)	0.49 (0)	0.69 (0)
13	0.50 (0)	0.42 (0)	0.46 (0)	0.55 (0)	0.68 (0)
14	0.49 (0)	0.42 (0)	0.45 (0)	0.54 (0)	0.70 (0)
15	0.52 (0)	**0.41** (0)	0.46 (0)	0.57 (0)	0.71 (0)
16	0.50 (0)	0.45 (0)	0.45 (0)	0.56 (0)	0.69 (0)
17	0.49 (0)	0.44 (0)	0.45 (0)	0.55 (0)	0.74 (0)

4.5.2 Comparison between heuristics

The ILS-SISRs and ILS-RR algorithms were compared with the original ILS-BS algorithm proposed in Morais et al. (2024). All three methods were evaluated on the complete set of 900 instances. The results are summarized in Table 4.16, which reports the average optimality gaps, average computational times, and the number of infeasible solutions (Inf.) for each instance size.

Overall, the ILS-BS algorithm achieved the smallest average gap for most instance sizes, although its performance was closely matched by both ILS-SISRs and ILS-RR. The three algorithms exhibited robust behavior for instances with up to 60 jobs, with no infeasible solutions observed. For larger instances, however, ILS-BS produced a small number of infeasible solutions, whereas ILS-SISRs and ILS-RR consistently maintained feasibility across all tested instances. This difference can be attributed to the *double-bridge* perturbation employed in ILS-BS, which does not account for inventory feasibility during the perturbation phase, in contrast to the perturbation strategies adopted by ILS-SISRs and ILS-RR.

Table 4.16: Performance comparison of ILS variants by instance size.

n	**ILS-BS**			**ILS-RR**			**ILS-SISR**		
	Gap (%)	Time (s)	Inf.	Gap (%)	Time (s)	Inf.	Gap (%)	Time (s)	Inf.
10	0.00	0.04	0	0.00	0.05	0	0.00	0.00	0
20	0.02	0.29	0	0.02	0.28	0	0.01	0.25	0
30	0.07	0.88	0	0.08	0.79	0	0.10	0.80	0
40	0.16	1.87	0	0.13	1.56	0	0.15	1.72	0
50	0.17	3.35	0	0.16	2.80	0	0.15	3.22	0
60	0.18	5.29	0	0.18	4.35	0	0.19	5.16	0
70	0.22	8.23	3	0.22	6.63	0	0.22	8.02	0
80	0.20	12.03	3	0.21	9.62	0	0.21	11.88	0
90	0.16	15.83	2	0.16	12.47	0	0.16	15.72	0
100	0.20	22.41	6	0.25	17.71	0	0.23	22.35	0

To further investigate the impact of instance characteristics on algorithmic performance, a detailed analysis of the three ILS variants was conducted with respect to the parameters α, τ and η.

The results reported in Table 4.17 show that, similarly to the BCP algorithm, all ILS variants exhibit lower average gaps for higher values of α, a behavior that contrasts with the performance observed for the MILP and B&C approaches.

Tables 4.18 and 4.19 report the average results grouped by τ for instances with $n \leq 50$ and $n > 50$, respectively. The results show a consistent trend across all three algorithms: performance improves as τ increases. This behavior can be attributed to the greater scheduling flexibility associated with more dispersed release dates, which reduces temporal congestion and facilitates the construction of higher-quality schedules.

In addition, Tables 4.20 and 4.21 present the average performance grouped by η for instances with $n \leq 50$ and $n > 50$, respectively. The observed trends indicate that the heuristics generally achieve better results as the inventory capacity increases. This outcome is expected, since larger capacities offer greater flexibility in controlling

inventory levels throughout the scheduling process, thereby mitigating feasibility constraints and improving solution quality.

Table 4.17: Average ILS results per α

n	α	ILS-BS			ILS-RR			ILS-SISR		
		Gap (%)	Time (s)	Inf.	Gap (%)	Time (s)	Inf.	Gap (%)	Time (s)	Inf.
10	10	0.00	0.04	0	0.00	0.05	0	0.00	0.00	0
10	100	0.00	0.05	0	0.00	0.05	0	0.00	0.00	0
20	10	0.03	0.29	0	0.03	0.27	0	0.02	0.25	0
20	100	0.01	0.29	0	0.01	0.28	0	0.01	0.26	0
30	10	0.08	0.81	0	0.09	0.73	0	0.13	0.76	0
30	100	0.06	0.95	0	0.06	0.85	0	0.07	0.85	0
40	10	0.22	1.75	0	0.17	1.44	0	0.20	1.62	0
40	100	0.09	1.99	0	0.10	1.67	0	0.09	1.83	0
50	10	0.20	3.10	0	0.19	2.60	0	0.18	2.98	0
50	100	0.13	3.60	0	0.13	3.01	0	0.12	3.46	0
60	10	0.21	4.86	0	0.18	3.97	0	0.22	4.69	0
60	100	0.15	5.72	0	0.18	4.73	0	0.16	5.62	0
70	10	0.25	7.08	3	0.23	5.76	0	0.22	7.06	0
70	100	0.20	9.38	0	0.22	7.49	0	0.22	8.99	0
80	10	0.24	10.31	3	0.22	8.33	0	0.25	10.31	0
80	100	0.16	13.75	0	0.19	10.92	0	0.17	13.46	0
90	10	0.19	13.30	1	0.15	10.78	0	0.18	13.59	0
90	100	0.14	18.36	1	0.17	14.15	0	0.15	17.86	0
100	10	0.21	18.38	6	0.25	15.01	0	0.23	18.67	0
100	100	0.19	26.43	0	0.25	20.41	0	0.23	26.03	0

Finally, Table 4.22 presents a comparative analysis of the average gap and number of infeasible solutions for the ILS algorithms and the populational algorithms presented in the work of Guerra et al. (2024), namely the HyPRR and a variant that uses SISRs as ruin-and-recreate operator (HyPRR-SISRs), the Hybrid Genetic Search with different crossover operators (HGS-OX, HGS-PMX, HGS-CX), a HyPRR variant without diversity control (ND) and with a single individual (SS). All populational algorithms were tuned using the same procedure described in Guerra et al. (2024) and used the ILS-SISRs time as their stopping criterion to ensure a fair comparison.

The results indicate that both ILS-SISRs and ILS-RR consistently produced feasible solutions across all instance sizes, demonstrating their robustness in handling inventory constraints. In contrast, the populational algorithms faced challenges in maintaining feasibility, particularly as the instance size increased, leading to a notable number of infeasible solutions. This outcome highlights the effectiveness of the perturbation and local search strategies employed in the ILS variants for preserving solution feasibility in complex scheduling scenarios.

Table 4.18: Average ILS results per τ for instance sizes ≤ 50.

n	τ	ILS-BS			ILS-RR			ILS-SISR		
		Gap (%)	Time (s)	Inf.	Gap (%)	Time (s)	Inf.	Gap (%)	Time (s)	Inf.
10	0.2	0.00	0.05	0	0.00	0.05	0	0.00	0.00	0
10	0.6	0.00	0.05	0	0.00	0.05	0	0.00	0.00	0
10	1.0	0.00	0.04	0	0.00	0.05	0	0.00	0.00	0
10	1.4	0.00	0.04	0	0.00	0.05	0	0.00	0.00	0
10	1.8	0.00	0.04	0	0.00	0.05	0	0.00	0.00	0
20	0.2	0.09	0.35	0	0.10	0.32	0	0.04	0.29	0
20	0.6	0.01	0.31	0	0.00	0.29	0	0.00	0.26	0
20	1.0	0.00	0.28	0	0.00	0.26	0	0.00	0.25	0
20	1.4	0.00	0.28	0	0.00	0.26	0	0.00	0.25	0
20	1.8	0.00	0.24	0	0.01	0.24	0	0.02	0.22	0
30	0.2	0.29	1.25	0	0.35	1.08	0	0.44	1.11	0
30	0.6	0.03	0.95	0	0.03	0.84	0	0.03	0.84	0
30	1.0	0.01	0.88	0	0.01	0.81	0	0.01	0.81	0
30	1.4	0.00	0.66	0	0.00	0.61	0	0.00	0.63	0
30	1.8	0.01	0.66	0	0.00	0.60	0	0.00	0.63	0
40	0.2	0.68	2.97	0	0.60	2.29	0	0.64	2.58	0
40	0.6	0.08	2.09	0	0.06	1.74	0	0.08	1.88	0
40	1.0	0.01	1.58	0	0.00	1.36	0	0.01	1.53	0
40	1.4	0.01	1.37	0	0.01	1.23	0	0.01	1.33	0
40	1.8	0.00	1.33	0	0.00	1.18	0	0.00	1.30	0
50	0.2	0.70	5.79	0	0.72	4.57	0	0.67	5.19	0
50	0.6	0.09	3.55	0	0.04	3.05	0	0.04	3.41	0
50	1.0	0.02	2.60	0	0.01	2.27	0	0.01	2.60	0
50	1.4	0.02	2.59	0	0.02	2.20	0	0.01	2.60	0
50	1.8	0.01	2.23	0	0.00	1.93	0	0.00	2.31	0

Table 4.19: Average ILS results per τ for instance sizes > 50.

n	τ	ILS-BS			ILS-RR			ILS-SISR		
		Gap (%)	Time (s)	Inf.	Gap (%)	Time (s)	Inf.	Gap (%)	Time (s)	Inf.
60	0.2	0.68	9.14	0	0.73	6.77	0	0.81	8.26	0
60	0.6	0.16	5.72	0	0.16	4.86	0	0.15	5.66	0
60	1.0	0.02	4.00	0	0.00	3.45	0	0.00	4.04	0
60	1.4	0.02	3.81	0	0.00	3.24	0	0.00	3.86	0
60	1.8	0.01	3.78	0	0.00	3.41	0	0.00	3.96	0
70	0.2	0.90	15.50	0	0.91	11.63	0	0.89	14.31	0
70	0.6	0.16	8.08	0	0.14	6.89	0	0.16	8.08	0
70	1.0	0.06	6.88	3	0.06	5.64	0	0.04	6.92	0
70	1.4	0.01	5.57	0	0.00	4.65	0	0.00	5.57	0
70	1.8	0.00	5.12	0	0.00	4.32	0	0.00	5.24	0
80	0.2	0.84	23.59	0	0.94	17.09	0	0.93	21.87	0
80	0.6	0.12	11.95	0	0.07	9.75	0	0.09	11.70	0
80	1.0	0.03	9.36	0	0.01	8.00	0	0.02	9.70	0
80	1.4	0.01	8.24	0	0.00	7.19	0	0.00	8.77	0
80	1.8	0.01	7.01	3	0.00	6.08	0	0.00	7.37	0
90	0.2	0.60	30.03	0	0.65	20.76	0	0.67	27.90	0
90	0.6	0.16	17.22	0	0.10	14.18	0	0.12	16.85	0
90	1.0	0.04	11.61	1	0.03	9.93	0	0.01	12.37	0
90	1.4	0.01	10.89	1	0.00	9.42	0	0.00	11.54	0
90	1.8	0.01	9.41	0	0.00	8.05	0	0.00	9.94	0
100	0.2	0.77	46.14	0	1.04	32.18	0	0.93	42.56	0
100	0.6	0.17	23.80	0	0.19	19.85	0	0.19	24.10	0
100	1.0	0.04	16.80	6	0.02	14.62	0	0.02	18.03	0
100	1.4	0.01	12.48	0	0.00	10.65	0	0.00	13.17	0
100	1.8	0.00	12.81	0	0.00	11.26	0	0.00	13.88	0

Table 4.20: Average ILS results per η for instance sizes ≤ 50.

n	η	ILS-BS			ILS-RR			ILS-SISR		
		Gap (%)	Time (s)	Inf.	Gap (%)	Time (s)	Inf.	Gap (%)	Time (s)	Inf.
10	1	0.00	0.05	0	0.00	0.05	0	0.00	0.00	0
10	3	0.00	0.04	0	0.00	0.04	0	0.00	0.00	0
10	5	0.00	0.04	0	0.00	0.04	0	0.00	0.00	0
20	1	0.04	0.36	0	0.02	0.36	0	0.01	0.32	0
20	3	0.02	0.27	0	0.01	0.24	0	0.00	0.23	0
20	5	0.00	0.25	0	0.03	0.22	0	0.02	0.22	0
30	1	0.09	1.15	0	0.12	1.10	0	0.17	1.07	0
30	3	0.08	0.77	0	0.08	0.67	0	0.09	0.70	0
30	5	0.04	0.72	0	0.03	0.59	0	0.04	0.65	0
40	1	0.27	2.35	0	0.28	2.12	0	0.29	2.23	0
40	3	0.13	1.63	0	0.08	1.31	0	0.11	1.48	0
40	5	0.07	1.62	0	0.04	1.24	0	0.04	1.47	0
50	1	0.34	4.47	0	0.35	4.09	0	0.32	4.47	0
50	3	0.08	2.94	0	0.05	2.34	0	0.07	2.72	0
50	5	0.08	2.65	0	0.07	1.98	0	0.05	2.47	0

Table 4.21: Average ILS results per η for instance sizes > 50.

n	η	ILS-BS			ILS-RR			ILS-SISR		
		Gap (%)	Time (s)	Inf.	Gap (%)	Time (s)	Inf.	Gap (%)	Time (s)	Inf.
60	1	0.32	7.16	0	0.39	6.46	0	0.41	7.17	0
60	3	0.11	4.35	0	0.07	3.40	0	0.10	4.16	0
60	5	0.10	4.36	0	0.08	3.18	0	0.07	4.14	0
70	1	0.39	11.30	3	0.46	10.01	0	0.43	11.44	0
70	3	0.13	6.80	0	0.08	5.21	0	0.11	6.47	0
70	5	0.15	6.59	0	0.13	4.66	0	0.12	6.17	0
80	1	0.34	16.35	3	0.40	14.61	0	0.39	17.21	0
80	3	0.15	10.39	0	0.10	7.84	0	0.14	9.75	0
80	5	0.11	9.36	0	0.11	6.43	0	0.10	8.68	0
90	1	0.30	21.54	2	0.33	19.00	0	0.33	22.72	0
90	3	0.11	13.76	0	0.09	10.07	0	0.09	12.97	0
90	5	0.08	12.19	0	0.06	8.34	0	0.07	11.47	0
100	1	0.42	33.82	6	0.59	29.48	0	0.53	35.33	0
100	3	0.10	17.63	0	0.09	13.07	0	0.10	16.93	0
100	5	0.07	15.76	0	0.08	10.58	0	0.07	14.78	0

Table 4.22: Average gaps with number of infeasible solutions in parentheses

n	ILS-SISRs	ILS-RR	HyPRR	HyPRR-SISRs	HGS-OX	HGS-PMX	HGS-CX	ND	SS
10	0.00 (0)	0.00 (0)	0.60 (4)	0.01 (0)	0.53 (3)	0.66 (7)	1.10 (4)	0.55 (5)	0.00 (0)
20	0.01 (0)	0.02 (0)	1.29 (18)	0.12 (0)	1.52 (17)	1.84 (18)	1.69 (16)	1.62 (16)	0.08 (0)
30	0.11 (0)	0.10 (0)	0.48 (6)	0.13 (0)	0.53 (52)	0.50 (90)	0.59 (70)	0.44 (11)	0.15 (0)
40	0.18 (0)	0.17 (0)	0.14 (0)	0.17 (0)	0.26 (49)	0.14 (108)	0.49 (109)	0.16 (0)	0.21 (0)
50	0.21 (0)	0.22 (0)	0.18 (5)	0.21 (3)	0.37 (110)	0.11 (127)	0.52 (91)	0.22 (4)	0.26 (9)
60	0.25 (0)	0.24 (0)	0.22 (12)	0.22 (10)	0.40 (82)	0.21 (141)	0.71 (97)	0.24 (11)	0.29 (20)
70	0.30 (0)	0.31 (0)	0.26 (28)	0.27 (31)	0.33 (98)	0.18 (82)	0.41 (62)	0.29 (20)	0.33 (38)
80	0.30 (0)	0.27 (0)	0.29 (31)	0.23 (49)	0.33 (97)	0.24 (39)	0.74 (48)	0.26 (41)	0.30 (43)
90	0.20 (0)	0.21 (0)	0.20 (17)	0.22 (30)	0.37 (133)	0.16 (40)	0.36 (41)	0.22 (26)	0.26 (52)
100	0.23 (0)	0.26 (0)	0.21 (77)	0.17 (80)	0.35 (66)	0.18 (53)	0.38 (40)	0.19 (87)	0.24 (98)

Chapter 5

Concluding remarks

This work introduced and studied the single-machine scheduling problem with release dates, sequence-dependent setup times, and inventory constraints, denoted as $1|r_j, s_{ij}, inv|C_{\max}$. This problem arises in practical applications where production or handling operations must simultaneously account for temporal constraints, setup-dependent processing, and limited inventory capacity, thus increasing both modeling and computational complexity.

To formally define the problem, four mathematical formulations were proposed: a position-indexed formulation, an arc-indexed formulation, an arc–inventory–indexed formulation, and an arc–time–indexed formulation. A benchmark set of instances was also introduced to support a comprehensive computational evaluation. Extensive experiments were conducted for all formulations, showing that the arc-indexed formulation consistently outperformed the alternatives, emerging as the most effective exact modeling approach for the problem.

Building upon this formulation, a Branch-and-Cut (B&C) algorithm was developed by incorporating capacity cuts commonly employed in vehicle routing problems with capacity constraints. In addition, a Branch-Cut-and-Price (BCP) algorithm was proposed, extending a branch-and-price framework previously applied to related scheduling problems. Computational results indicate that the BCP solved a larger number of instances to optimality, while the B&C generally achieved lower average gaps among the exact methods.

To complement the exact solution methods, two heuristic approaches based on the Iterated Local Search (ILS) framework were proposed. The first combines ILS with

a ruin-and-recreate perturbation strategy, while the second incorporates a perturbation inspired by the SISRs algorithm. These heuristics were compared with state-of-the-art scheduling heuristics from the literature. The results showed that integrating ruin-and-recreate mechanisms within the ILS framework leads to highly competitive solutions. Moreover, both heuristic variants were able to consistently generate feasible solutions across all runs, highlighting their robustness in handling inventory constraints.

Overall, this work contributes to the scheduling literature by introducing a novel problem variant, proposing multiple exact and heuristic solution methods, and providing a thorough computational analysis that clarifies the impact of inventory constraints on problem difficulty and algorithmic performance.

Several directions for future research emerge from this study. First, the development of stronger and more problem-specific valid inequalities may further improve the performance of exact methods, particularly for larger instances. Second, extending the proposed models and algorithms to settings with uncertainty, such as stochastic processing times, release dates, or inventory variations. Finally, generalizing the proposed approaches to multi-machine or parallel-machine environments could substantially broaden the scope of the problem and its relevance to real-world applications.

References

Allahverdi, A., Ng, C. T., Cheng, T. C. E., and Kovalyov, M. Y. (2008). A survey of scheduling problems with setup times or costs. *European Journal of Operational Research*, 187:985–1032.

Augerat, P., Belenguer, J., Benavent, E., Corberán, A., and Naddef, D. (1998). Separating capacity constraints in the cvrp using tabu search. *European Journal of Operational Research*, 106(2):546–557.

Balakrishnan, N., Kanet, J. J., and Sridharan, V. (1999). Early/tardy scheduling with sequence dependent setups on uniform parallel machines. *Computers & Operations Research*, 26(2):127–141.

Baldacci, R., Mingozzi, A., and Roberti, R. (2011). New route relaxation and pricing strategies for the vehicle routing problem. *Operations Research*, 59(5):1269–1283.

Bartels, J.-H. and Zimmermann, J. (2009). Scheduling tests in automotive r&d projects. *European Journal of Operational Research*, 193(3):805–819.

Bianco, L., Ricciardelli, S., Rinaldi, G., and Sassano, A. (1988). Scheduling tasks with sequence- dependent processing times. *Naval Research Logistics*, 35(2):177–184.

Briskorn, D., Choi, B.-C., Lee, K., Leung, J., and Pinedo, M. (2010). Complexity of single machine scheduling subject to nonnegative inventory constraints. *European Journal of Operational Research*, 207(2):605–619.

Briskorn, D., Jaehn, F., and Pesch, E. (2013). Exact algorithms for inventory constrained scheduling on a single machine. *Journal of scheduling*, 16:105–115.

Briskorn, D. and Leung, J. Y. (2013). Minimizing maximum lateness of jobs in inventory constrained scheduling. *Journal of the Operational Research Society*, 64(12):1851–1864.

Briskorn, D. and Pesch, E. (2013). Variable very large neighbourhood algorithms for truck sequencing at transshipment terminals. *International Journal of Production Research*, 51(23-24):7140–7155.

Bulhões, T., Subramanian, A., Erdoğan, G., and Laporte, G. (2018). The static bike relocation problem with multiple vehicles and visits. *European Journal of Operational Research*, 264(2):508–523.

Carlier, J. and Kan, A. R. (1982). Scheduling subject to nonrenewable-resource constraints. *Operations Research Letters*, 1(2):52–55.

Chang, T.-Y., Chou, F.-D., and Lee, C.-E. (2004). A heuristic algorithm to minimize total weighted tardiness on a single machine with release dates and sequence-dependent setup times. *Journal of the Chinese Institute of Industrial Engineers*, 21(3):289–300.

Chemla, D., Meunier, F., and Wolfler Calvo, R. (2013). Bike sharing systems: Solving the static rebalancing problem. *Discrete Optimization*, 10(2):120–146.

Chou, F.-D., Wang, H.-M., and Chang, T.-Y. (2008). Algorithms for the single machine total weighted completion time scheduling problem with release times and sequence-dependent setups. *Int J Adv Manuf Technol*, 43(810).

Christiaens, J. and Berghe, G. V. (2020). Slack induction by string removals for vehicle routing problems. *Transportation Science*, 54(2):417–433.

Davari, M., Ranjbar, M., De Causmaecker, P., and Leus, R. (2020). Minimizing makespan on a single machine with release dates and inventory constraints. *European Journal of Operational Research*, 286(1):115–128.

Diana, R. O., de Souza, S. R., and Filho, M. F. (2018). A variable neighborhood descent as ils local search to the minimization of the total weighted tardiness on unrelated parallel machines and sequence dependent setup times. *Electronic Notes in Discrete Mathematics*, 66:191–198. 5th International Conference on Variable Neighborhood Search.

Driessel, R. and Mönch, L. (2011). Variable neighborhood search approaches for scheduling jobs on parallel machines with sequence-dependent setup times, precedence constraints, and ready times. *Computers & Industrial Engineering*, 61(2):336–345. Combinatorial Optimizatiion in Industrial Engineering.

Fan, J., Öztop, H., Tasgetiren, M., and Gao, L. (2019). A variable block insertion heuristic for single machine with release dates and sequence dependent setup times for makespan minimization. In *2019 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 1676–1683, Xiamen, China. IEEE.

Gafarov, E. R., Lazarev, A. A., and Werner, F. (2011). Single machine scheduling problems with financial resource constraints: Some complexity results and properties. *Mathematical Social Sciences*, 62(1):7–13.

Ghorbanzadeh, M., Ranjbar, M., and Jamili, N. (2019). Transshipment scheduling at a single station with release date and inventory constraints. *Journal of Industrial and Production Engineering*, 36(5):301–312.

Graham, R., Lawler, E., Lenstra, J., and Kan, A. (1979). Optimization and approximation in deterministic sequencing and scheduling: a survey. In P.L. Hammer, E. J. and Korte, B., editors, *Discrete Optimization II Proceedings of the Advanced Research Institute on Discrete Optimization and Systems Applications of the Systems Science Panel of NATO and of the Discrete Optimization Symposium co-sponsored by IBM Canada and SIAM Banff, Aha. and Vancouver*, volume 5 of *Annals of Discrete Mathematics*, pages 287 – 326. Elsevier.

Guerra, F., Morais, R., Moura, A. C., Bruck, B., and Subramanian, A. (2024). Uma heurística populacional híbrida para problemas de sequenciamento de tarefas com datas de liberação e restrições de inventário. *Anais do LVI Simpósio Brasileiro de Pesquisa Operacional*.

Györgyi, P. and Kis, T. (2014). Approximation schemes for single machine scheduling with non-renewable resource constraints. *Journal of Scheduling*, 17:135–144.

Györgyi, P. and Kis, T. (2017). Approximation schemes for parallel machine scheduling with non-renewable resources. *European Journal of Operational Research*, 258(1):113–123.

Herr, O. and Goel, A. (2016). Minimising total tardiness for a single machine scheduling problem with family setups and resource constraints. *European Journal of Operational Research*, 248(1):123–135.

Kim, S.-I., Choi, H.-S., and Lee, D.-H. (2007). Scheduling algorithms for parallel machines with sequence-dependent set-up and distinct ready times: minimizing total tardiness. *PROCEEDINGS OF THE INSTITUTION OF MECHANICAL ENGINEERS PART B-JOURNAL OF ENGINEERING MANUFACTURE*, 221(6):1087–1096.

Lawler, E. L., Lenstra, J. K., Rinnooy Kan, A. H., and Shmoys, D. B. (1993). Chapter 9 sequencing and scheduling: Algorithms and complexity. In *Logistics of Production and Inventory*, volume 4 of *Handbooks in Operations Research and Management Science*, pages 445–522. Elsevier.

Lin, S.-W., Lee, Z.-J., Ying, K.-C., and Lu, C.-C. (2011). Minimization of maximum lateness on parallel machines with sequence-dependent setup times and job release dates. *Computers & Operations Research*, 38(5):809–815.

Lin, Y.-K. and Hsieh, F.-Y. (2014). Unrelated parallel machine scheduling with setup times and ready times. *International Journal of Production Research*, 52(4):1200 – 1214.

Logendran, R., McDonell, B., and Smucker, B. (2007). Scheduling unrelated parallel machines with sequence-dependent setups. *Computers & Operations Research*, 34(11):3420–3438.

Mladenovic, N. and Hansen, P. (1997). Variable neighborhood search. *Computers & Operations Research*, 24(11):1097–1100.

Montoya-Torres, J., González-Solano, F., and Soto-Ferrari, M. (2012). Deterministic machine scheduling with release times and sequence-dependent setups using random-insertion heuristics. *International Journal of Advanced Operations Management*, 4(1/2):4–26.

Morais, R., Bulhões, T., and Subramanian, A. (2024). Exact and heuristic algorithms for minimizing the makespan on a single machine scheduling problem with sequence-dependent setup times and release dates. *European Journal of Operational Research*, 315(2):442–453.

Neumann, K. and Schwindt, C. (2003). Project scheduling with inventory constraints. *Mathematical Methods of Operations Research*, 56(3):513–533.

Nogueira, T. H., Ramalhinho, H. L., de Carvalho, C. R. V., and Ravetti, M. G. (2022). A hybrid VNS-Lagrangean heuristic framework applied on single machine scheduling problem with sequence-dependent setup times, release dates and due dates. *Optimization Letters*, 16:59–78.

Ovacikt, I. and Uzsoy, R. (1994). Rolling horizon algorithms for a single-machine dynamic scheduling problem with sequence-dependent setup times. *International Journal of Production Research*, 32(6):1243–1263.

Pinedo, M. L. (2022). *Scheduling: Theory, Algorithms, and Systems*. Springer-Verlag, Nova Iorque, 6 edition.

Pinheiro, J. C. S. N. and Arroyo, J. E. C. (2020). Effective ig heuristics for a single-machine scheduling problem with family setups and resource constraints. *Annals of Mathematics and Artificial Intelligence*, 88:169–185.

Ruiz, R. and Stützle, T. (2007). A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. *European Journal of Operational Research*, 177(3):2033–2049.

Schwindt, C. and Trautmann, N. (2000). Batch scheduling in process industries: an application of resource–constrained project scheduling. *OR-Spektrum*, 22:501–524.

Sergey Kovalev, I. C. and Bécuwe, A. (2024). Single machine scheduling with resource constraints: Equivalence to two-machine flow-shop scheduling for regular objectives. *Journal of the Operational Research Society*, 75(7):1343–1346.

Shin, H., Kim, C.-O., and Kim, S. (2002). A tabu search algorithm for single machine scheduling with release times, due dates, and sequence-dependent setup times. *The International Journal of Advanced Manufacturing Technology*, 19(11):859–866.

Slowiński, R. (1984). Preemptive scheduling of independent jobs on parallel machines subject to financial constraints. *European Journal of Operational Research*, 15(3):366–373.

Soares, L. C. and Carvalho, M. A. (2022). Application of a hybrid evolutionary algorithm to resource-constrained parallel machine scheduling with setup times. *Computers & Operations Research*, 139:105637.

Su, Y.-C. and Lin, B. M. (2022). Minimizing the total weighted completion time in relocation scheduling. *Computers & Industrial Engineering*, 173:108662.

Subramanian, A., Drummond, L., Bentes, C., Ochi, L., and Farias, R. (2010). A parallel heuristic for the vehicle routing problem with simultaneous pickup and delivery. *Computers & Operations Research*, 37(11):1899–1911. Metaheuristics for Logistics and Vehicle Routing.

Toker, A., Kondakci, S., and Erkip, N. (1991). Scheduling under a non-renewable resource constraint. *Journal of the Operational Research Society*, 42(9):811–814.

Torabi, S., Sahebjamnia, N., Mansouri, S., and Bajestani, M. A. (2013). A particle swarm optimization for a fuzzy multi-objective unrelated parallel machines scheduling problem. *Applied Soft Computing*, 13(12):4750–4762.

Vanderbeck, F., Sadykov, R., and Tahiri, I. (2017). Bapcod — a generic branch-and-price code. Technical report.

Velez-Gallego, M. C., Maya, J., and Montoya-Torres, J. (2016). A beam search heuristic for scheduling a single machine with release dates and sequence dependent setup times to minimize the makespan. *Computers & Operations Research*, 73:132–140.

Yepes-Borrero, J. C., Villa, F., Perea, F., and Caballero-Villalobos, J. P. (2020). Grasp algorithm for the unrelated parallel machine scheduling problem with setup times and additional resources. *Expert Systems with Applications*, 141:112959.

Ying, K.-C. and Cheng, H.-M. (2010). Dynamic parallel machine scheduling with sequence-dependent setup times using an iterated greedy heuristic. *Expert Systems with Applications*, 37(4):2848–2852.

Ümit Bilge, Kıraç, F., Kurtulan, M., and Pekgün, P. (2004). A tabu search algorithm for parallel machine total tardiness problem. *Computers & Operations Research*, 31(3):397–414.