UNIVERSITY OF MODENA AND REGGIO EMILIA
Department of Sciences and Methods for Engineering
Doctorate School in Industrial Innovation Engineering
XXXVII Cycle

Optimization techniques for workforce allocation, dynamic scheduling, and storage location in supply chains

DOCTORAL THESIS

Author:
Beatrice BOLSI

Supervisor:
Prof. Manuel IORI
Co-Supervisor:
Vinícius Loti DE LIMA
Coordinator:
Prof. Franco ZAMBONELLI

Academic year 2023/2024

UNIVERSITY OF MODENA AND REGGIO EMILIA

Abstract

Doctorate School in Industrial Innovation Engineering
Department of Sciences and Methods for Engineering

Optimization techniques for workforce allocation, dynamic scheduling, and storage location in supply chains

by Beatrice BOLSI

This thesis presents a collection of optimization studies that arise in perishable product industries and healthcare supply chains, particularly focusing on workforce allocation, scheduling, and storage location optimization in real-world operational environments.

The first study examines a two-stage flexible flow shop problem for processing perishable products. Key decisions include shifts for workers, machine allocation and order scheduling, with the goal of lexicographically minimizing three objectives: the number of unscheduled orders, the weighted tardiness, and production costs. The resulting problem is addressed by developing a heuristic and applying three metaheuristic algorithms: Random Multi-Start (MR), Biased Random Key Genetic Algorithm (BRKGA), and Variable Neighborhood Search (VNS). A constraint programming model is also formulated to evaluate the quality of the scheduling solution. Computational experiments demonstrate that BRKGA is the most effective approach and obtains better results than MR and VNS.

The second study deals with the dynamic assignment of multi-skill configurations to multiple servers in the healthcare context. Two different models are considered: the first prohibits patients from leaving the system before receiving service, with the objective of minimizing total tardiness based on target service times. The second model introduces priority-weighted tardiness and allows patients to leave the system if waiting times become excessive, incurring additional penalties. To address the challenge of solution selection in the common Scenario-Based Planning Approach (SBPA), we propose a new approach that balances solutions across multiple scenarios, which we call Scenario-Based Planning and Recombination Approach (SBPRA). Extensive computational experiments prove that SBPRA outperforms the traditional SBPA, achieving a 38% improvement in the average objective function value.

The third study focuses on a storage location assignment problem arising in a pharmaceutical warehouse, incorporating incompatibility and isolation constraints. The objective is to minimize the total distance traveled by order pickers to retrieve all required items. An Iterated Local Search (ILS) algorithm is proposed to solve the problem, with numerical experiments carried out on simulated data. A detailed procedure is also provided for structuring the input data of the warehouse layout. The results show a significant improvement over the greedy heuristic procedure, commonly used in real-life operations.

Overall, this thesis demonstrates the effectiveness of the developed optimization techniques in addressing a series of challenging real-world problems. In particular, it highlights the utility of genetic algorithms, such as the BRKGA, in obtaining high-quality solutions for complex flexible flow shop scheduling problems, as well as the SBPRA in solving dynamic multi-skill assignment problems. While these methods were specifically tailored to the unique requirements of the studied production and operational settings, their versatility makes them applicable to broader classes of optimization problems. Indeed, this opens several promising avenues for future research. For instance, we aim to investigate the performance of the BRKGA algorithm in dynamic versions of the flexible flow shop problem, where real-time changes require continuous adaptation. Similarly, the SBPRA approach could be further generalized by extending its recombination framework to a wider range of assignment and scheduling problems, thereby unlocking new possibilities for both theoretical advancements and practical applications.

Contents

List of Tables

List of Figures

Acknowledgements

At the end of this doctoral journey, it is my willing duty to thank all the colleagues who in some way contributed more or less directly to this result, which is hopefully meant to be an opening to new perspectives and directions.

My first thanks go to my mentor and supervisor Manuel, who patiently introduced and thoughtfully accompanied me in the discovery of Operations Research, its methods and tools, and in the challenging paths of academic life. Secondly, to Vinícius, who, in addition to having transmitted an inestimable amount of knowledge, managed so often to contaminate me with his essential curiosity and perennial, genuine search for deepening and improvement. To Thiago, Arthur and Nielson, for having directed me from the beginning in research, as well as in writing and dissemination. To Paolo, never a colleague but a generous researcher and friend and now an expert origamist. To all my zealous doctoral companions and researchers, a special thank you for all the shared efforts, laughter, planes to catch, and dormitories. Finally, thanks to the Teatro Sociale di Gualtieri, a true testing ground for every attempt at optimization.

Chapter 1

Introduction

Optimization techniques for resource allocation, scheduling, and storage assignment represent foundational pillars of Operations Research. Their study is motivated not only by their vast industrial relevance—where operational efficiency is a key driver of competitiveness in increasingly dynamic and heterogeneous markets—but also by the deep methodological and theoretical challenges they pose. These challenges frequently emerge from real-world problems, rooted in complex and variable operational environments, and require innovative solutions that balance rigor with practical applicability.

Dynamic and stochastic versions of these problems further amplify their relevance, offering fertile ground for advancing both theory and practice. Unlike their static counterparts, where deterministic information is fully known in advance, dynamic and stochastic problems involve decision-making under uncertainty, with information revealed progressively over time. This inherent complexity mirrors the realities of modern industries, where disruptions, variability, and rapid changes demand adaptive and robust optimization frameworks. The ability to tackle such challenges effectively can lead to significant advancements in domains such as manufacturing, logistics, and healthcare.

This thesis addresses three distinct, yet interconnected optimization problems, each embodying key aspects of resource allocation, scheduling, and storage management.

In Chapter 2, we focus on a two-stage flexible flow shop with heterogeneous machines problem for processing perishable products, which presents a challenging scheduling scenario due to both time-sensitive demands and workforce allocation related decisions impact. Multistage flexible flow shop problems have been extensively studied, as highlighted in the foundational work of [88] and the recent survey by [106]. The problem arises in a meat industry, where a set of orders (jobs) is revealed daily and needs to be processed within a single day, meeting release and due date constraints, together with a maximum given waiting time between the two stages. Key decisions include worker shifts, machine allocation, and order scheduling, all aimed at minimizing three objectives: the number of unscheduled orders, weighted tardiness, and production costs. Random Key Genetic Algorithms, as well as Variable Neighborhood Search ones as introduced in [46], provide a foundation for solving complex optimization problems. This problem is addressed by developing a heuristic approach and applying three metaheuristic algorithms—Random Multi-Start (MR), Biased Random Key Genetic Algorithm (BRKGA), and Variable Neighborhood Search (VNS). A constraint programming model is also formulated to benchmark the solution quality. Computational experiments show that BRKGA outperforms MR and VNS, offering a more effective solution to scheduling problems in industries dealing with perishable goods. We further address this problem in our recent works [17, 18].

Chapter 3 examines the dynamic assignment of multi-skill configurations to servers in healthcare systems, addressing the need to allocate resources in a way that minimizes tardiness while considering patient priorities. We model two different approaches: one that prevents patients from leaving the system before service is provided, and another that allows patient departure under high waiting times, incurring penalties. While moving from

the seminal work by [10] on Scenario-Based Planning Approach (SBPA), the novelty of this chapter lies in the introduction of the Scenario-Based Planning and Recombination Approach (SBPRA), which balances solutions across multiple scenarios, improving the selection process compared to the traditional SBPA. The computational experiments demonstrate a significant improvement in the average objective function value with SBPRA, which offers valuable insights into resource management in healthcare settings where demand and patient priorities can change dynamically. The practical and theoretical significance of the recombination step is elaborated in our recent study [19], while earlier versions of this problem were presented in [5, 16].

In Chapter 4, we focus on a variant of the Storage Location Assignment Problem (SLAP), which introduces additional complexities by incorporating specific constraints that govern the placement of items, such as safety regulations and operational restrictions. Specifically, we address the Storage Location Assignment Problem with Product-Cell Incompatibility and Isolation Constraints (SLAP-PCIIC), a more complex variant that arises in the context of pharmaceutical warehouse management. The challenge in this setting is to assign storage locations to items while respecting product-cell incompatibility (e.g., ventilation, refrigeration) and isolation (e.g., radioactivity, toxicity) constraints. The goal is to minimize the total distance that order pickers travel to retrieve all required items. To solve this problem, we propose an Iterated Local Search (ILS) algorithm, which is evaluated through extensive computational experiments using simulated data. ILS algorithms are the subject of the seminal work of [70]. The results demonstrate that the ILS algorithm effectively handles the complexities of the SLAP-PCIIC and provides significant improvements in the operational efficiency of warehouse systems. This chapter contributes to the growing body of research on storage location assignment problems, testing a solution approach for complex logistics environments, particularly in industries with stringent safety and operational requirements, such as pharmaceuticals. This research is presented also in [73, 74].

This thesis investigates three distinct optimization problems, each representing an important aspect of resource allocation, scheduling, and storage management, while also highlighting the underlying commonality of optimizing systems under constraints and uncertainty. Despite the differences in application domains and specific challenges, these problems all require advanced decision-making frameworks that can adapt to variability in real-time.

Chapter 2

Heuristic algorithms for integrated workforce allocation and scheduling of perishable products

In this chapter, we address the integrated problem of workforce allocation and scheduling for perishable products in a two-stage flexible flow shop environment. The aim is to optimize production planning by minimizing unscheduled orders, weighted tardiness, and production costs. To solve this complex problem, we propose a constructive heuristic and embed it within three metaheuristic algorithms: Random Multi-Start, Biased Random Key Genetic Algorithm, and Variable Neighborhood Search. Details of this work can be found in [17, 18].

2.1 Introduction

Production scheduling along with workforce sizing and allocation constitute one of the most common classes of problems faced by companies seeking to optimise their manufacturing system. In production scheduling problems, a set of jobs has to be scheduled in a set of machines, while in workforce sizing and allocation problems, workers have to be (eventually) allocated in a set of machines. In both cases, a set of practical constraints has to be satisfied and a given objective optimised. The scheduling problem becomes even more general and demanding when integrated with that of workforce sizing and allocation, because the processing times of the jobs depend on these decisions, so that the number of workers operating each machine drastically influences decisions about the jobs to be scheduled.

In this work, we study a scheduling problem faced daily by a meat producing company, characterised by deterioration-prevention constraints linking the two stages. These constraints are very common in planning problems of perishable products supply chains. The company receives daily a set of orders to be produced in a single day. Each order is associated with a due date and is produced by following up to two stages: in the first stage, the perishable product (meat) is processed (cut) on a given bench, and in the second stage it is sent to a conveyor to be packed into disposable trays. To avoid the product deterioration, the company imposes a maximum waiting time between the processing of an order on the first stage and its processing on the second. Benches and conveyors can be seen as heterogeneous parallel machines, and their productivity depends on the number of workers operating each of them. The company derives a new production plan daily, comprising the workforce allocation and the scheduling of operations composing the final orders. The scheduling problem faced by the company (which has a number of operational constraints that are formally discussed in Section 2.2) is a generalisation of the well-known two-stage flexible (or hybrid) flow shop problem (see, e.g., [40]). The overall problem considers, in lexicographic

way, the minimisation of the number of unscheduled orders, the weighted tardiness, and the production costs.

Flexible flow shop problems have been studied for decades. For extensive surveys, see, e.g., [69], [99], and [106]. There are several well-studied practical variants of these problems, and in the following we focus on two-stage variants. In [118], a tabu-search algorithm is proposed to solve a variant of the problem with identical machines that considers sequence-dependent setup times with the aim of minimising the makespan. In [112], the authors study how repetitive scheduling influences an environment with a single machine in the first stage and multiple lines in the second stage. An application related to a sterilisation plant can be found in [98]. The objective is to simultaneously minimise the makespan and the number of tardy jobs, with jobs being allowed to be processed in parallel batches. In [67], the authors study a system with unrelated parallel machines and task tail group constraints, with the objective of minimising the total tardiness. Nikzad, Rezaeian, Mahdavi, and Rastgar [82] worked on the assembling of components. Stage one has parallel machines with different speeds, while stage two has two dedicated lines. After passing the first stage, components are allocated to a line in accordance with their specifications. The authors combined the simulated annealing and imperialist competitive algorithms in order to obtain an effective solution method.

For two-stage flexible flow shops, Yu, Huang, and Lee [131] studied the impact of determining the number, composition, and allocation of batches, besides the scheduling of batches to minimise the total tardiness. The authors combined an integer linear programming model with iterative algorithms to obtain high quality solutions. The work in [55] proposes different solution methods for environments composed of a single machine either in the first or in the second stage. In [121], the authors propose an integer linear programming model and metaheuristics to solve a problem from the glass-ceramic industry, with objective of minimising the makespan and energy consumption. [107] studied a two-stage flexible flowshop problem with batch processing machines and jobs with unequal release and due dates. The authors present a mixed integer programming model and propose a hybrid stage-based decomposition with neighborhood search to solve the problem. Evolutionary algorithm (EA) approachs for solving manufacturing scheduling problems has been addressed since some decades ago, leading to valuable results, as pointed out in [44]. In the last years, EAs are still employed and explored in addressing scheduling problems, also with stochastic models. In [43], a dual-objective stochastic hybrid flow shop deteriorating scheduling model is established, with the objectives of minimising makespan and total tardiness. Then, a hybrid multiobjective EA is proposed and compared with another multi-objective EA from the hybrid flow shop scheduling literature.

A preliminary version of the current paper has been published in the proceedings of APMS 2021 [15]. The preliminary work proposes a constructive heuristic, which is embedded within a Random Multi-Start Algorithm. The current work represents an extended version of the preliminary paper, with a number of additional contents, including: a detailed description of the problem; two additional metaheuristics (i.e., a Biased Random Key Genetic Algorithm and a Variable Neighborhood Search based one) that embed a lighter although structure-preserving constructive heuristic than the one proposed in [15]; a Constraint Programming model that provides a lower bound (and possibly an optimal solution) on the primary objective of the scheduling problem; and an extension of the instance set that allows for a much broader computational study.

The remaining of the paper is organised as follows. Section 2.2 provides a formal description of the problem. Section 2.3 presents the constructive heuristic. Section 2.4 describes the three proposed metaheuristic algorithms. Section 2.5 outlines the Constraint Programming model for the scheduling problem. Section 2.6 provides the results of the computational experiments. Finally, Section 2.7 presents concluding remarks and some directions for future

researches.

2.2 Problem description

In this section, we present a detailed description of the problem. The company receives a set O of orders to be scheduled in a workday, by following several operational constraints in a two-stage production environment. Each order is expected to be produced before its due date, which corresponds to the time in which it has to be shipped to its final destination. Then, the problem is to determine the workforce and production schedule of a workday.

A workday.

A workday is defined over a time horizon of 24 hours (discretised into minutes) and has a set $\mathcal{P}$ of disjoint working shifts. In our case study, the number of shifts is limited to two, i.e., $|\mathcal{P}| \leq 2$. The problem input related to a workday consists of:

- $s^{\min}, s^{\max} \in \mathbb{Z}_+$: minimum and maximum start time of any shift;
- $e^{\min}, e^{\max} \in \mathbb{Z}_+$: minimum and maximum end time of any shift;
- $l^{\min}, l^{\max} \in \mathbb{Z}_+$: minimum and maximum duration of any shift;
- $w^{\min}_{\rho,k}, w^{\max}_{\rho,k} \in \mathbb{Z}_+$: minimum and maximum number of workers at stage $k \in \{1,2\}$ in shift $\rho \in \mathcal{P}$.

A first set of decisions involves the number $|\mathcal{P}|$ of working shifts, as well as, the start time $S^k_\rho \in [s^{\min}, s^{\max}]$, end time $E^k_\rho \in [e^{\min}, e^{\max}]$ and number of hired workers $W^k_\rho \in [w^{\min}_{\rho,k}, w^{\max}_{\rho,k}] \cap \mathbb{Z}_+$, for each shift $\rho \in \mathcal{P}$ and stage $k \in \{1,2\}$. Operational constraints related to a workday are:

$$E^k_\rho - S^k_\rho \in [l^{\min}, l^{\max}], \qquad \forall \rho \in \mathcal{P}, k \in \{1,2\}, \tag{2.1}$$

$$[S^k_{\rho_1}, E^k_{\rho_1}] \cap [S^k_{\rho_2}, E^k_{\rho_2}] = \emptyset, \qquad \forall \rho_1, \rho_2 \in \mathcal{P}, \rho_1 \neq \rho_2, k \in \{1,2\}, \tag{2.2}$$

where constraints (2.1) impose the minimum and maximum duration of each shift, and constraints (2.2) impose that shifts are disjoint.

A shift in a workday.

After deciding the duration and number of workers for each shift, production-related decisions must be made. The production in each shift is composed by two stages. Each stage $k \in \{1,2\}$ has a set $\mathcal{M}_k$ of heterogeneous parallel machines whose speed is variable, where $\mathcal{M}_1 \cap \mathcal{M}_2 = \emptyset$. For each $m \in \mathcal{M} = \mathcal{M}_1 \cup \mathcal{M}_2$, we receive in input:

- $w^{\min}_m, w^{\max}_m \in \mathbb{Z}_+$: minimum and maximum number of workers that can operate machine m.

We assume workers have the same efficiency and the speed of each machine $m \in \mathcal{M}$ is proportional to the number of workers operating it. Machines speeds are thus not given in input, but rather used a priori to compute the processing times of all orders (described next). Workers cannot be reallocated to different machines during a shift. In our case study, first-stage machines $\mathcal{M}_1$ are benches on which the meat is prepared, whereas second-stage machines $\mathcal{M}_2$ are conveyors where the products are distributed to be packed into disposable trays that finally receive a stamp with the product information. Besides, some orders must be processed in a single stage, whereas others must be processed (sequentially) on both stages.

Decisions internal to a shift $\rho \in \mathcal{P}$ consist of determining the number $W_{m\rho} \in [w_m^{\min}, w_m^{\max}] \cap \mathbb{Z}_+$ of workers operating each machine $m \in \mathcal{M}$ and, for each order $o \in O$, deciding whether o is processed in ρ, which is indicated by a binary variable $B_{o\rho} \in \{0,1\}$. The following constraints must be satisfied:

$$\sum_{m \in \mathcal{M}_k} W_{m\rho} \leq W_\rho^k, \qquad \forall \rho \in \mathcal{P}, k \in \{1,2\}, \tag{2.3}$$

$$\sum_{\rho \in \mathcal{P}} B_{o\rho} \leq 1, \qquad \forall o \in O. \tag{2.4}$$

Constraints (2.3) limit the number of workers operating machines in each shift and stage, whereas constraints (2.4) impose that each order is processed in at most one shift (some orders may be left unscheduled).

Details on the production in a shift.

Now we provide details on the order scheduling in each shift. For each order $o \in O$, we receive in input:

- $d_o \in \mathbb{Z}_+$: due date;
- $r_o \in \mathbb{Z}_+$: release date;
- $q_o \in \mathbb{Z}_+$: net weight;
- y_o^{tray}: disposable tray type;
- $y_o^{\text{stp.}}$: stamp type;
- y_o^{raw}: raw product type;
- $p_{omw} \in \mathbb{Z}_+ \cup \{+\infty\}$: processing time in machine m, given w operating workers;
- $\mathcal{K}_o \subseteq \{1,2\}$: the set of stages in which o must exactly be processed in order to be scheduled, depending on the preparation type;
- $\mathcal{M}_o \subseteq \mathcal{M}$: set of machines in which o can be processed.

Processing times are computed a priori based on machine speeds, worker efficiency, and orders' net weight and raw product type. There is a buffer (assumed to be of unlimited size) between the two stages. Hence, the problem input proceeds with:

- $t_{\text{buffer}}^{\max} \in \mathbb{Z}_+$: maximum waiting time that any order can remain in the buffer (to preserve the product quality);
- $t_{m_1,m_2}^{\text{trans}} \in \mathbb{Z}_+ \cup \{+\infty\}$: transportation time between each machine $m_1 \in \mathcal{M}_1$ of the first stage to every machine $m_2 \in \mathcal{M}_2$ of the second stage;

Each machine $m \in \mathcal{M}$ has setup times, given in input:

- $u_m^{\text{raw}} \in \mathbb{Z}_+$: first-stage setup time whenever the raw product is changed;
- $u_m^{\text{tray}} \in \mathbb{Z}_+$: second-stage setup time whenever a different tray is required;
- $u_m^{\text{stp.}} \in \mathbb{Z}_+$: second-stage setup time whenever a different stamp is required.

Notice that, to facilitate problem comprehension, we allowed processing time for a given order, machine and number of workers to be infinite in some cases, e.g. when the productivity of o on the machine is null (this is always the case if the machine belongs to a stage which is not in $\mathcal{K}_o$, but it may also happen when the product type is not compatible with the machine), or there are no workers operating the machine. Also $t^{\text{trans}}_{m_1,m_2}$ is allowed to be infinite, meaning that it is not possible to transport any product to m_2 departing from m_1. From the implementation side we choose, for each order, to remove from the set of its available machines those with associated infinite processing time and second stage machines with infinite transportation time with respect to the first stage machine chosen. The last decisions are to determine the start time $S^k_o \in \mathbb{Z}_+$, end time $E^k_o \in \mathbb{Z}_+$, and processing machine $M^k_o \in \mathcal{M}_k \cap \mathcal{M}_o$ of each order $o \in O$ in stage $k \in \mathcal{K}_o$. Then, constraints related to the order-scheduling in a shift $\rho \in \mathcal{P}$ and for each $o \in O$ such that $B_{o\rho} = 1$, are the following:

$$S^k_\rho \le r_o \le S^k_o \le E^k_o \le E^k_\rho, \qquad \forall k \in \mathcal{K}_o, \tag{2.5}$$

$$(M^k_o = m \text{ and } W_{m\rho} = w) \implies E^k_o - S^k_o \ge p_{omw}, \qquad \forall k \in \mathcal{K}_o, \tag{2.6}$$

$$\begin{aligned}&(\mathcal{K}_o = \{1,2\} \text{ and } M^1_o = m_1 \text{ and } M^2_o = m_2) \implies \\ &\quad t^{\text{trans}}_{m_1,m_2} \le S^2_o - E^1_o \le t^{\max}_{\text{buffer}}, \qquad \forall m_1 \in \mathcal{M}_1, m_2 \in \mathcal{M}_2.\end{aligned} \tag{2.7}$$

In addition, for each $\rho \in \mathcal{P}$ and for each $o_1, o_2 \in O$ such that $B_{o_1\rho} = B_{o_2\rho} = 1$, we have the following:

$$\begin{aligned}&(1 \in \mathcal{K}_{o_1} \cap \mathcal{K}_{o_2}, M^1_{o_1} = M^1_{o_2}, \text{ and } y^{\text{raw}}_{o_1} \ne y^{\text{raw}}_{o_2}) \implies \\ &\quad S^1_{o_1} \ge E^1_{o_2} + u^{\text{raw}}_{M^1_{o_1}} \text{ or } S^1_{o_2} \ge E^1_{o_1} + u^{\text{raw}}_{M^1_{o_1}},\end{aligned} \tag{2.8}$$

$$\begin{aligned}&(2 \in \mathcal{K}_{o_1} \cap \mathcal{K}_{o_2}, M^2_{o_1} = M^2_{o_2}, \text{ and } y^{\text{tray}}_{o_1} \ne y^{\text{tray}}_{o_2}) \implies \\ &\quad S^2_{o_1} \ge E^2_{o_2} + u^{\text{tray}}_{M^2_{o_1}} \text{ or } S^2_{o_2} \ge E^2_{o_1} + u^{\text{tray}}_{M^2_{o_1}},\end{aligned} \tag{2.9}$$

$$\begin{aligned}&(2 \in \mathcal{K}_{o_1} \cap \mathcal{K}_{o_2}, M^2_{o_1} = M^2_{o_2}, \text{ and } y^{\text{stp.}}_{o_1} \ne y^{\text{stp.}}_{o_2}) \implies \\ &\quad S^2_{o_1} \ge E^2_{o_2} + u^{\text{stp.}}_{M^2_{o_1}} \text{ or } S^2_{o_2} \ge E^2_{o_1} + u^{\text{stp.}}_{M^2_{o_1}}.\end{aligned} \tag{2.10}$$

Constraints (2.5) impose that any order o that is scheduled in ρ must be processed within the shift operating time, and that the release date of o is satisfied. Constraints (2.6) model the processing times of the orders. Constraints (2.7) impose the transportation time between the first and the second stage, and the maximum waiting time that each order can remain in the buffer. Constraints (2.8), (2.9), and (2.10) guarantee that setup times are satisfied. We remark that constraints (2.5) and (2.7) limit the choice of the second stage machines to those having finite transportation time from the machine chosen on the first stage.

Overview.

Summarising, the decisions of the overall problem involve the distribution of the working shifts, the number of workers on each shift, the number of workers operating each machine in each shift, and the production scheduling of the jobs. The quality of a solution is measured by three objectives to be minimised according to a lexicographic order. The first objective minimises the total number of unscheduled orders $\text{TUS}(\cdot)$:

$$\text{TUS}(\cdot) = \min(|O| - \sum_{o \in O} \sum_{\rho \in \mathcal{P}} B_{o\rho}).$$

The second objective minimises the total weighted tardiness of scheduled orders $\mathrm{TWT}(\cdot)$:

$$\mathrm{TWT}(\cdot) = \min \sum_{o \in O} B_o \tau_o q_o,$$

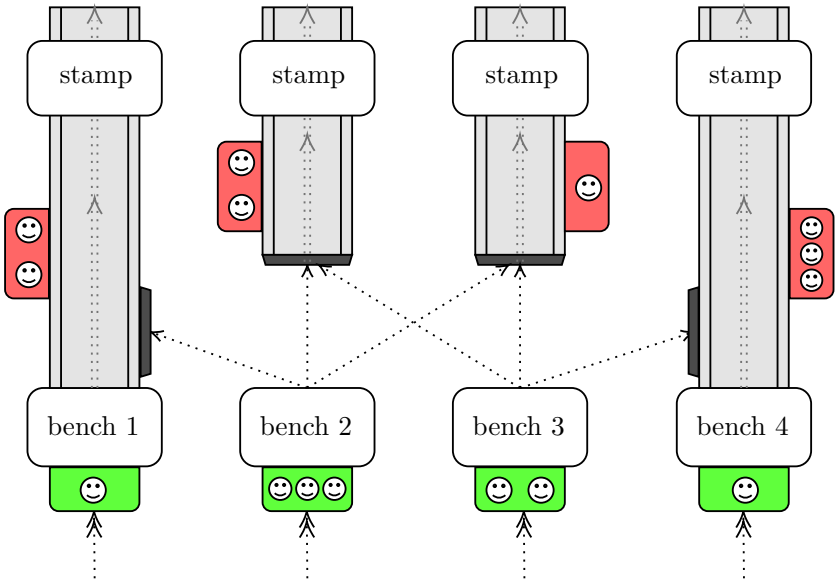
where $\tau_o = \max\{0, \max_{k \in \mathcal{K}_o}\{E_o^k\} - d_o\}$ is the tardiness of order o and $B_o = \sum_{\rho \in \mathcal{P}} B_{o\rho}$. The tardiness weight of an order is the corresponding order net weight. The third and last objective minimises the total production cost $\mathrm{TPC}(\cdot)$:

$$\mathrm{TPC}(\cdot) = \min \sum_{\rho \in \mathcal{P}} \sum_{k \in \{1,2\}} (E_\rho^k - S_\rho^k) W_\rho^k c^k,$$

where c^k is the wage per unit of time of a stage-k worker, given in input. The company has additional production-related costs that are fixed, and so, we choose not to formally include them in the objective. In addition, although machine setups are not formally included in the objective, they are highly disliked by the company and it is always preferable to keep their number as small as possible. This aspect is directly treated within the heuristic algorithms proposed in the next section.

Figure 2.1 depicts an example of the production system. Each first-stage and second-stage machine is associated with a number of workers. As in the real system, benches 1 and 4 are directly connected to the first and fourth conveyors, so the meat processed in these benches cannot be reallocated to other conveyors, nor stay in the buffer. Since benches 2 and 3 are not directly connected to conveyors, their processed meat go to the buffer, to later be distributed to any of the conveyors.

FIGURE 2.1: An illustrative example of the production system: four benches and four conveyors, each with a varying number of operating workers. First bench is connected to the first conveyor and the fourth bench is connected to the fourth conveyor. The remaining benches and conveyors are not connected between each other.



2.3 Constructive heuristic

This section describes the constructive heuristic we propose to solve the problem with a fixed number W_ρ^k of workers for each stage $k \in \{1,2\}$ of each shift $\rho \in \mathcal{P}$. This heuristic is

embedded within the metaheuristics described in Section 2.4, which are also responsible for iteratively attempting different values for W_{ρ}^{k}.

2.3.1 Overall Algorithm

The overall structure of the constructive heuristic is presented in Algorithm 1. The algorithm uses a function, called `schedule`, which schedules a set of orders to a shift with workers already allocated to machines, and considers a list $\mathcal{L}$ of priority coefficients that is used to minimise tardiness.

Algorithm 1: Overall constructive heuristic

Input: (Sequential) set of orders O and number of workers W_{ρ}^{k} for each stage $k \in \{1,2\}$ and shift $\rho \in \mathcal{P}$

$(\rho_1^*, \rho_2^*) \leftarrow$ empty solution
$\mathcal{L}^1 \leftarrow (0,0,...,0)$
Assign workers to machines of the first shift by considering a workload balance
for $i \leftarrow 1,\ldots,\Pi_{list}$ **do**
  $\rho_1 \leftarrow$ `schedule`$(O, \mathcal{L}^1)$
  $O' \leftarrow O$ minus the orders scheduled in shift ρ_1
  $\mathcal{L}^2 \leftarrow \mathcal{L}^1$
  Assign workers to machines of the second shift by considering a workload balance
  for $j \leftarrow 1,\ldots,\Pi_{list}$ **do**
    $\rho_2 \leftarrow$ `schedule`$(O', \mathcal{L}^2)$
    if (ρ_1, ρ_2) *has better objective value than* (ρ_1^*, ρ_2^*) **then**
      $(\rho_1^*, \rho_2^*) \leftarrow (\rho_1, \rho_2)$
    if *there is tardiness in shift* ρ_2 **then**
      Increase priority coefficients in $\mathcal{L}^2$ of all tardy orders
    else
      Break the loop
  if *there is tardiness in shift* ρ_1 **then**
    Increase priority coefficients in $\mathcal{L}^1$ of all tardy orders
  else
    Break the loop
return (ρ_1^*, ρ_2^*)

In the beginning of the algorithm, the first-shift workers are allocated to machines by means of a function that considers a balance of the workload. After creating the schedule for the first shift, the workload based function is used again to allocate the second-stage workers to the machines. Then, the remaining orders are scheduled in the second shift. During the process of creating a schedule for a shift, we produce up to Π_{list} different schedules, which are iteratively obtained by updating a list $\mathcal{L}$ and calling the function `schedule`. Recall that $\mathcal{L}$ is used within `schedule` as an order-priority quantifier. In this way, in order to avoid tardiness, $\mathcal{L}$ is updated by increasing the priority coefficients of tardy orders in the current schedule.

The start time of a given shift is set as the minimum possible start time among the start times of first-stage machines that can be used in the shift. Workers are assigned to machines

proportionally with the expected workload. The expected workload for a given machine m is calculated by adding $\frac{p_{om1}}{NAM_o}$ for each order that can be processed on machine m, where NAM_o is the number of admissible (i.e., compatible) machines for the product associated with the order o.

2.3.2 Details on the `schedule` Function

Algorithm 2: `schedule`$(O, \mathcal{L})$

Input: (Sequential) set of orders O and $\mathcal{L} \in \mathbb{Z}^{|O|}$

$\rho \leftarrow$ empty solution
$O_U \leftarrow O$.`getUnscheduledOrders`()
$\mathcal{M}_A \leftarrow \mathcal{M}$.`getAvailableMachines`()
while $\mathcal{M}_A \neq \emptyset$ *and* $O_U \neq \emptyset$ **do**
  $\mathcal{L}_U^2 \leftarrow \mathcal{L}$.`getPriorityCoefficientsOfSecondStageOrders`$(O_U)$
  $l_* \leftarrow \max\{\mathcal{L}_U^2\}$
  $O'_U \leftarrow O_U$.`getFirstStageOnlyOrders`()
  $(\rho, O_U) \leftarrow$ `tryScheduleOrdersOfPriorityOnBestMachine`(O'_U, l_*)
  $d_* \leftarrow \min\{O_U$.`getDueDatesOfSecondStageorders`$()\}$
  $(\rho, O_U) \leftarrow$ `tryScheduleFirstStageOnlyOrdersEarlierDueDate`(O'_U, d_*)
  $m^2_{\text{best}} \leftarrow$ `getEarliestAvailableSecondStageMachine`()
  $(o_{\text{best}}, m^1_{\text{best}}) \leftarrow$ `getBestOrderAndFirstStageMachine`(O_U, m^2_{best})
  $(\rho, O_U, \mathcal{M}_A) \leftarrow$ `tryScheduleOrderOnMachines`$(o_{\text{best}}, m^1_{\text{best}}, m^2_{\text{best}})$
  if $o_{best} \notin O_U$ **then**
    $O'_U \leftarrow O_U$.`getSimilarOrdersToOrder`(o_{best})
    $O'_U \leftarrow O'_U$.`sortByDominanceCriteria`()
    for $o \in O'_U$ **do**
      $m^1_{\text{best}} \leftarrow$ `getBestFirstStageMachineToScheduleOrder`(o)
      $(\rho, O_U, \mathcal{M}_A) \leftarrow$ `tryScheduleOrderOnMachines`$(o, m^1_{\text{best}}, m^2_{\text{best}})$
return ρ

Function `schedule`$(O, \mathcal{L})$ schedules orders in O to a shift that is already set (i.e., its time window is already fixed and workers are already allocated to machines). The list $\mathcal{L}$ has a priority coefficient l_o for each order $o \in O$. The algorithm iteratively schedules orders based on dominance criteria, and it halts either when all orders are successfully scheduled or no more orders can be scheduled to the current shift (i.e., when machines are all closed). The algorithm prioritises first-stage-only orders (i.e., those with $\mathcal{K} = \{1\}$), due to structure required by the company, which imposes that first stage production starts considerably earlier than the second stage one and maximum waiting-time constraint is easily found to be violated at the beginning of the shift. For this reason, we consider a special type of prioritisation of first-stage-only orders, focusing only on those (if any) whose priority coefficient in $\mathcal{L}$ is strictly higher than any other unscheduled order that must be processed on the second stage and those (if any) whose due dates are strictly earlier than all other unscheduled orders. To determine the scheduling sequence of first-stage-only orders, the first dominance criterion is their priority coefficients in $\mathcal{L}$ and the second one is the order sequence in O (tie-breaker). The dominant order is scheduled to the first-stage machine minimising the order completion time.

At any iteration in which no more first-stage-only orders satisfy their scheduling conditions stated above (concerning maximal priority and minimal due-date), the algorithm proceeds by scheduling both-stage and second-stage-only orders. First, the algorithm chooses

a second-stage machine with earliest available time, then it chooses a candidate order, and, if this is a both-stage order, the first-stage machine selected is the one with earliest available time and that satisfies time-horizon feasibility constraints. If no such machine exists, the order is not listed as a candidate one. Then, we choose among candidate orders the dominant one by following an extensive list of dominance criteria. The first criterion is whether the next order will create a stamp-setup in the second-stage machine. These setups are the less desired by the company, and this strategy helps to avoid them. The second criterion chooses all non-dominated orders with the highest priority coefficients in $\mathcal{L}$. The next dominance criterion keeps only orders with earliest-possible start time in the second-stage machine chosen. The fourth criterion prioritises orders with earliest due-dates. The next criterion prioritises all remaining second-stage-only orders, if any is still non-dominated, otherwise, it prioritises both-stage orders that are of the same raw product as of the bench with earliest-possible start-time. This choice helps to minimise raw-product related setups. Finally, if there are still more than one non-dominated order, the last dominance criterion breaks tie by selecting the earliest order in the sequence of O given in input. This tie-breaking criterion based on the order sequence is further exploited with randomisation by the metaheuristics described in Section 2.4.

2.4 Metaheuristics

In this section, we describe the three developed metaheuristics: the Random Multi-Start Algorithm, the Biased Random Key Genetic Algorithm, and the Variable Neighborhood Search based one. Each metaheuristic uses as underlying constructive algorithm the one described in Section 2.3, and as a stopping criterion a time limit τ.

2.4.1 Random Multi-Start Algorithm (MR)

The first algorithm we implemented is the Random Multi-Start Algorithm (MR). MR is a metaheuristic widely used in real case studies to find good quality solutions in a reasonable time. Among the advantages that this algorithm offers, we certainly appreciate the simplicity and immediacy of implementation. At each iteration: (1) a vector of four elements is randomly generated representing the number of workers on the first and second stages in the first and second shifts, and (2) a shuffling of the vector of orders is made. The resulting vectors are then given as input to the constructive algorithm. The solution thus obtained is compared with the incumbent solution: if the current solution is better than the incumbent one, then the incumbent one is updated.

2.4.2 Biased Random Key Genetic Algorithm (BRKGA)

Genetics-based algorithms had been successfully used since many decades ago in a wide variety of combinatorial optimisation problems, including scheduling problems [46]. Genetic Algorithms (GAs) are metaheuristics based on Darwinian evolutionary principles of species. Classically, GAs are used to search for optimal or near-optimal solutions through the evolution of a population, based on the biological criterion of 'survival of the fittest'. Initially, a *population* of N_{pop} individuals, i.e., solutions encoded in a vector s of real components (*alleles*) representing their chromosome, is generated. Each individual s is associated with the value of a fitness function $\Phi(s)$ which places it in the social hierarchy of the current population. It follows that if the fitness function and sorting criteria are consistent with the definition of the objective function, the elite population represents the set of best solutions for the problem. To construct the next generation, a subset of *parents* of the current population is selected, from which pairs of individuals are extracted. The child of a couple inherits

part of the alleles from one parent and the remaining part needed to complete a chromosome from the other. Each genetic algorithm has mechanisms of both diversification, such as the introduction, at each generation, of a certain number of *mutants*, and intensification, aimed at preserving the part of the population hierarchically highest according to the fitness function. In the Random-Key Genetic Algorithm (RKGA), each solution is encoded as a random vector $s \in [0,1)^n$. In the Biased Random Key Genetic Algorithm (BRKGA), a bias is introduced in the choice of one parent always from the elite set, while the other is randomly chosen among all the non-elite population. From couples of chosen parents a new set of chromosomes (*offspring*) is generated. The offspring chromosome is composed by choosing, allele by allele, with a certain probability p_a the allele from the elite parent. In each subsequent generation, the elite set is totally preserved from the previous generation as a p_e percentage of 'best' individuals. A percentage p_m of mutants is introduced, and the remaining part of the population (offspring) is generated from mating parents as just discussed. Thus, raising the p_a, N_{pop}, and p_e thresholds leads to an *intensification* of the local search, while raising p_m imposes a *diversification* threshold in the local search.

In our framework, each chromosome needs to encode both the information about the sequence of orders to be processed and a 4-component vector establishing the number of workers for the first and the second shift on the first and the second stage ($n = |O| + 4$). In both cases, the random vector $s \in [0,1)^n$ (chromosome) with which a solution is encoded is used as ranking for re-sorting the sequence of orders given in the input file for the constructive algorithm. Evaluation of the given chromosome is done by using a fitness function $\Phi(\cdot)$ defined as:

$$\Phi(s) = \gamma_1 \text{TUS}(s) + \gamma_2 \text{TWT}(s) + \gamma_3 \text{TPC}(s), \tag{2.11}$$

where weights γ_i are used in consistence with the original lexicographic objective function, satisfying $\gamma_1 >> \gamma_2 >> \gamma_3$. With respect to the magnitude order of our data, we set $\gamma_1 = 10^6, \gamma_2 = 1, \gamma_3 = 10^{-4}$.

2.4.3 Variable Neighborhood Search (VNS)

The VNS metaheuristic is based on a systematical change of problem-dependent neighborhoods. It sequentially visits each given neighborhood, while following three basic sequential steps. In the first step (*shaking*), randomised operations based on the current neighborhood are used to derive a new solution starting from the current reference solution (i.e., the incumbent solution). In the next step, a *local search* is used to improve the solution from the previous step. Finally, in the last step (*move or not move*), if the solution obtained with the local search step is chosen to become the new incumbent solution, then the search returns to the first neighborhood; otherwise the search moves on to the next neighborhood [78].

The implementation of our VNS based metaheuristic uses four types of neighborhoods: (1) generate a vector with four number of workers (first and second shifts, first and second stages) chosen randomly within the range given by the constraints; (2) randomly select a subsequence of length at most k within the order vector and randomly change the positions of the elements of the subsequence; (3) randomly select a subsequence of length at most k within the order vector and reverse the positions of the elements of the subsequence; (4) randomly select a subsequence of length at most k within the order vector, and another random outer element, and insert the subsequence before that element. In the *shaking* step, starting from an initial solution, a solution belonging to the neighborhood of type N is generated, where N starts as the neighborhood of type (1). In the *move or not to move* step, the solution found and the incumbent one are compared according to the lexicographic objective function, and

if the solution found is better than the incumbent one, then the incumbent solution is updated and the process restarts from the neighborhood of type $N = (1)$, otherwise it changes the neighborhood, switching to the next one and continues up to neighborhood $N = (4)$. The *local search* step is applied only after passing through all four neighborhoods. This step consists in generating k more solutions in a neighborhood of type (4). We save the best solution obtained from the *local search* step if it is better than the best one, and the VNS based metaheuristic restart from the first neighborhood.

2.5 Constraint Programming model (CP)

In this section we define a Constraint Programming (CP) model resulting from the pure scheduling problem (i.e., with workers fixed on machines) described in Section 2.2. The main motivation of deriving such CP model is to evaluate the quality of the scheduling computation inside the constructive algorithm. Computational results regarding this evaluation are later reported in Section 2.6.3.

The CP model consists of interval variables $x^{\rho}_{o,k,m}$, representing the processing operation of order $o \in O$ on stage $k \in \mathcal{K}$ and machine $m \in \mathcal{M}$ within the shift $\rho \in \mathcal{P}$. These are optional interval variables, meaning that their domain can be either the empty set (i.e., they are not present), or an interval with integer extremes, as explained in constraint (2.12).

$$\texttt{DomainOf}\left(x^{\rho}_{o,k,m}\right) \subseteq \{\emptyset\} \cup \{[s,e) : \ s < e;\ s,e \in \mathbb{Z}\}, \tag{2.12}$$

where $s = \texttt{Start}\left(x^{\rho}_{o,k,m}\right)$ and $e = \texttt{End}\left(x^{\rho}_{o,k,m}\right)$. In addition, the following constraints must hold for each shift $\rho \in \mathcal{P}$.

For each $o \in O$, for each $k \in \mathcal{K}$ and for each $m \in \mathcal{M}$, the following must hold if $x^{\rho}_{o,k,m}$ is present:

$$\begin{aligned}
&\texttt{Start}\left(x^{\rho}_{o,k,m}\right) \geq \max\left\{r_o, S^k_{\rho}\right\},\\
&\texttt{End}\left(x^{\rho}_{o,k,m}\right) \leq E^k_{\rho},\\
&\texttt{End}\left(x^{\rho}_{o,k,m}\right) - \texttt{Start}\left(x^{\rho}_{o,k,m}\right) = p^{\rho}_{o,k,m}.
\end{aligned} \tag{2.13}$$

For each $o \in O$ such that $\mathcal{K}_o = \{1,2\}$:

$$\sum_{m\in\mathcal{M}} \texttt{PresenceOf}\left(x^{\rho}_{o,1,m}\right) = \sum_{m\in\mathcal{M}} \texttt{PresenceOf}\left(x^{\rho}_{o,2,m}\right). \tag{2.14}$$

For each $o \in O$ such that $\mathcal{K}_o = \{1\}$:

$$\sum_{m\in\mathcal{M}} \texttt{PresenceOf}\left(x^{\rho}_{o,2,m}\right) = 0. \tag{2.15}$$

For each $o \in O$ such that $\mathcal{K}_o = \{2\}$:

$$\sum_{m\in\mathcal{M}} \texttt{PresenceOf}\left(x^{\rho}_{o,1,m}\right) = 0. \tag{2.16}$$

For each $o \in O$ and for each $k \in \mathcal{K}$:

$$\sum_{\rho' \in \mathcal{P}, m \in \mathcal{M}} \texttt{PresenceOf}\left(x^{\rho'}_{o,k,m}\right) \leq 1. \tag{2.17}$$

For each $o \in O$ such that $\mathcal{K}_o = \{1,2\}$, for each $m_1 \in \mathcal{M}_1$, and for each $m_2 \in \mathcal{M}_2$, the following must hold if both $x^{\rho}_{o,1,m_1}$ and $x^{\rho}_{o,2,m_2}$ are present:

$$t^{\text{trans}}_{m_1,m_2} \leq \texttt{Start}\left(x^{\rho}_{o,2,m_2}\right) - \texttt{End}\left(x^{\rho}_{o,1,m_1}\right) \leq t^{\text{max}}_{\text{buffer}}. \tag{2.18}$$

For each $o_1, o_2 \in O$, for each $k \in \mathcal{K}$, and for each $m \in \mathcal{M}$:

$$\texttt{NoOverlapWithIntervalGap}\left(x^{\rho}_{o_1,k,m}, x^{\rho}_{o_2,k,m}, \text{SetupTime}_k\left(o_1,o_2\right)\right). \tag{2.19}$$

Constraints (2.13) limit the interval variables extremes s and e, due to the release date r_o and processing time $p^{\rho}_{o,k,m}$ of the order and the opening and closing time for each stage on each shift (S^k_{ρ} and E^k_{ρ}, respectively). Constraints (2.14), (2.15) and (2.16) define the meaning of preparation type for scheduled orders: both-stage preparation, first-stage-only preparation and second-stage-only preparation, respectively. Constraints (2.17) state that each order must be scheduled at most once on each stage, on at most one machine and in at most one shift. Constraints (2.18) set the interval variables extremes limitations for both-stage preparation orders, due to the transportation time t^{trans} and the maximum waiting time between stages $t^{\text{max}}_{\text{buffer}}$ for the quality preservation of the product. Finally, constraints (2.19) ensure both the non overlapping of orders on the same machine and the setup time compliance, where the setup time needed is defined as follows:

$$\text{SetupTime}_k\left(o_1,o_2\right) = \begin{cases} u^{\text{raw}}, & \text{if } k = 1 \text{ and } y^{\text{raw}}_{o_1} \neq y^{\text{raw}}_{o_2}, \\ u^{\text{stp.}}, & \text{if } k = 2 \text{ and } (y^{\text{stp.}}_{o_1} \neq y^{\text{stp.}}_{o_2} \text{ and } y^{\text{tray}}_{o_1} = y^{\text{tray}}_{o_2}), \\ u^{\text{tray}}, & \text{if } k = 2 \text{ and } (y^{\text{stp.}}_{o_1} = y^{\text{stp.}}_{o_2} \text{ and } y^{\text{tray}}_{o_1} \neq y^{\text{tray}}_{o_2}), \\ \max\left\{u^{\text{stp.}}, u^{\text{tray}}\right\}, & \text{if } k = 2 \text{ and } (y^{\text{stp.}}_{o_1} \neq y^{\text{stp.}}_{o_2} \text{ and } y^{\text{tray}}_{o_1} \neq y^{\text{tray}}_{o_2}), \\ 0, & \text{otherwise.} \end{cases}$$

For the implementation of the `NoOverlapWithIntervalGap` function, we use in practice the global constraint *IloNoOverlap* implemented in the IBM ILOG CP Optimizer, giving as input the list of all orders of a given stage, machine, and shift, rather than having a constraint for each pair of interval variables as described in the model above. For the setup times, we combine the use of *IloIntervalSequenceVar* and *IloTransitionDistance* by a straighforward implementation, as the one that can be found in the CP Optimizer's reference manual.

2.6 Computational experiments

We propose computational experiments, whose goals are: (1) studying the impact of Π_{list} parameter in the performance of the heuristic, and (2) making a comparison among the three developed metaheuristics. For that, we implemented the constructive algorithm and all metaheuristics in C++. All tests were executed in a computer with an Intel Xeon CPU E3-1245 v5 processor at 3.50 GHz and 32 GB of RAM. To solve the CP model, we use the CP Optimizer package present in IBM ILOG CPLEX Optimization Studio 12.10. The first test set was based on an initial set I of realistic instances, with $|I| = 35$, which consistently extends the set of 12 instances used in [15]. Each instance represents one day of production at the

company. Some randomisation was applied to all instances in order to meet the company's privacy requirements.

Since the BRKGA has many parameters, in order to calibrate their values, instead of a trial and error approach, we used the *irace* (iterated racing for automatic algorithm configuration) package [72], choosing among small, medium, and large-sized realistic instances from set I. The following parameter ranges were provided in input to irace: $N_{\text{pop}} \in \{10, 11, \ldots, 150\}$; $p_e \in [0.1, 0.3]$; $p_m \in [0.2, 0.7]$; and $p_a \in [0.5, 0.8]$. Therefore, according to irace, the following values produce overall good solutions: $N_{\text{pop}} = 66$; $p_e = 0.13$; $p_m = 0.55$; and $p_a = 0.79$.

The first experiments, presented in Section 2.6.1, study how the Π_{list} parameter affects the constructive algorithm in terms of solution quality and time. These tests allow both a tuning of the Π_{list} parameter for all three metaheuristics and to select a subset $I' \subseteq I$ with $|I'| = 23$ of harder instances. Set I' is defined by excluding from I instances in which a single run of the constructive algorithm manages to produce solutions with no unscheduled nor tardy jobs. The second test set is to analyze the performance of the three metaheuristics, under time limits of $\tau \in \{60, 300, 600, 1200\}$ seconds.

2.6.1 Tuning of Π_{list} parameter

We run the first set of experiments to determine a good Π_{list} value. The results are summarised in Tables 2.1 and 2.2. The evaluation metrics considered are the computational time `time` (in seconds) and the three components of the objective function, i.e., the total number of unscheduled orders, TUS, the total weighted tardiness, TWT (in minutes per kilograms) and the total production cost, TPC (in euros). After preliminary experiments, especially regarding computational time, we have reduced the values under consideration for the Π_{list} parameter to $\{1, 5, 20, 50\}$. Table 2.1 presents the absolute values of evaluation metrics obtained with $\Pi_{\text{list}} = 1$. The table shows that a single run of the constructive algorithm requires on average 0.05 seconds, which is a reasonably small time, given the complexity of the problem.

TABLE 2.1: Results of a single run of the constructive algorithm with $\Pi_{\text{list}} = 1$.

Instance	$\lvert O \rvert$	time	TUS	TWT	TPC
I01	168	0.05	0	395	8385
I02	158	0.05	3	52533	11012
I03	194	0.06	8	304544	11123
I04	253	0.06	0	48485	8151
I05	169	0.05	1	41284	9905
I06	176	0.05	4	86509	11406
I07	153	0.05	0	15308	11058
I08	164	0.05	4	105855	10552
I09	207	0.06	10	247040	11687
I10	194	0.06	26	367463	11006
I11	189	0.06	19	569142	11615
I12	160	0.05	0	434	8050
I13	178	0.05	13	0	6726
I14	180	0.05	8	0	7285
I15	183	0.05	8	0	8041
I16	201	0.06	10	149380	10953
I17	267	0.07	8	76821	10362
I18	102	0.04	2	416	5906
I19	154	0.05	0	159194	10205
I20	145	0.04	0	786	7400
I21	141	0.04	0	13732	6598
I22	127	0.04	0	8437	6920
I23	151	0.05	1	132505	10105
Average		0.05	5.43	103489.70	9323.97

Table 2.2 presents the relative percentage results obtained with $\Pi_{\text{list}} = \{5, 20, 50\}$ compared to $\Pi_{\text{list}} = 1$. For each metric $\nu = \{\text{TUS}, \text{TWT}, \text{TPC}\}$, we define $\nu_{\%,\Pi_{\text{list}}}$ as:

$$\nu_{\%,\Pi_{\text{list}}} = \frac{\nu_{\Pi_{\text{list}}} - \nu_1}{\nu_1},$$

while we denote by $\texttt{time}_r$ the computational time ratio obtained with any Π_{list} value with respect to 1, i.e.:

$$\texttt{time}_{r,\Pi_{\text{list}}} = \frac{\texttt{time}_{\Pi_{\text{list}}}}{\texttt{time}_1}.$$

Table 2.2 shows that increasing Π_{list} from 1 to 5 causes both a significant increase in computational time and a marked improvement in the solution quality, especially in terms of unscheduled jobs. Also, the improvement for objective function parameters is less impressive with higher values of Π_{list}, while computational time increases considerably. These observations motivate the use of $\Pi_{\text{list}} = 5$ for the subsequent tests.

TABLE 2.2: Impact of Π_{list} parameter on a single run of the constructive algorithm.

Inst.	$\lvert O\rvert$	$\texttt{time}_r$			$\text{TUS}_{\%}$			$\text{TWT}_{\%}$			$\text{TPC}_{\%}$		
		5	20	50	5	20	50	5	20	50	5	20	50
I01	168	1.5	2.0	3.0	0.0	0.0	0.0	-100.0	-100.0	-100.0	3.9	3.9	3.9
I02	158	1.2	2.0	3.8	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
I03	194	4.0	10.0	22.8	-25.0	-37.5	-37.5	-22.9	-38.9	-38.9	-0.2	0.2	0.2
I04	253	3.7	6.0	9.3	0.0	0.0	0.0	-99.4	-100.0	-100.0	2.1	3.2	3.2
I05	169	1.8	2.6	3.4	0.0	-100.0	-100.0	-33.4	-100.0	-100.0	3.9	13.4	13.4
I06	176	2.2	4.6	9.4	-75.0	-75.0	-75.0	-23.4	-23.4	-23.4	-1.5	-1.5	-1.5
I07	153	2.8	4.8	8.8	0.0	0.0	0.0	-100.0	-100.0	-100.0	2.0	2.0	2.0
I08	164	3.0	4.6	7.8	-25.0	-50.0	-50.0	-100.0	-99.1	-99.1	0.8	4.7	4.7
I09	207	3.3	9.2	21.0	0.0	0.0	-10.0	0.0	0.0	-48.5	0.0	0.0	-1.4
I10	194	3.4	9.4	21.0	-7.7	-7.7	-7.7	-20.2	-20.2	-20.2	1.4	1.4	1.4
I11	189	2.8	6.4	14.0	0.0	0.0	0.0	-47.3	-47.3	-47.3	1.9	1.9	1.9
I12	160	1.5	1.8	2.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
I13	178	1.0	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
I14	180	1.0	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
I15	183	1.0	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
I16	201	2.3	5.0	10.2	-20.0	-20.0	-20.0	-90.0	-99.2	-99.2	-1.3	-0.6	-0.6
I17	267	5.7	61.1	350.4	0.0	0.0	0.0	-72.9	-74.7	-74.7	2.4	2.4	2.4
I18	102	1.5	1.8	2.5	-100.0	-100.0	-100.0	-100.0	-100.0	-100.0	5.0	5.0	5.0
I19	154	2.8	6.0	12.5	0.0	0.0	0.0	-61.2	-62.9	-62.9	-4.5	-5.5	-5.5
I20	145	1.8	1.8	1.8	0.0	0.0	0.0	-100.0	-100.0	-100.0	-1.7	-1.7	-1.7
I21	141	1.5	1.5	1.5	0.0	0.0	0.0	-100.0	-100.0	-100.0	-0.8	-0.8	-0.8
I22	127	1.5	1.5	1.5	0.0	0.0	0.0	-100.0	-100.0	-100.0	-1.5	-1.5	-1.5
I23	151	4.0	35.3	195.3	-100.0	-100.0	-100.0	-35.6	-40.5	-40.5	-2.6	-3.5	-3.5
Average		2.40	7.83	30.65	-15.3	-21.3	-21.7	-52.5	-56.8	-58.9	0.4	1.0	0.9

2.6.2 Evaluation of metaheuristics

Next, we discuss the results of the experiments that evaluate the performance of all metaheuristics under different time limits. Tables 2.3, 2.4, 2.5, and 2.6 show the results from all metaheuristics under a time limit of 60, 300, 600, and 1200 seconds, respectively.

Regarding the first objective, which has the highest priority, the BRKGA obtained better (or equal) average results than the other two metaheuristics, for all time limits under consideration. When compared to the average results of a single run of the constructive heuristic (5.43 unscheduled orders), the results of all metaheuristics represent a good improvement already under 60 seconds of time limit.

The second and third objectives cannot be compared between different algorithms or time limits directly by the average results of all instances, since an improvement in the first

objective value often implies on worsening the second and third objectives values. This can be observed in Table 2.3, e.g., when comparing the BRKGA with the other algorithms, in instances I02, I09, and I10. However, other cases, as in instance I03, the first objective has been improved together with the second and third ones.

Similarly to what happens for the intra-method comparison with the same τ, we can observe that there is no assurance of an improvement for the second and third objectives as τ increases. Nevertheless, for both BRKGA and VNS based algorithm, we note that this improvement occurs both on average and point-wise in many cases. As for the BRKGA, we get improvements on the average TWT for each time limit transition, apart from when switching from $\tau = 600$ to $\tau = 1200$, while point-wise speaking there is an improvement in all cases except for I08 (60-300), I10 (600-1200), and I11 (60-300). Furthermore, the VNS improves on average the TWT on each time limit transition, apart from the switch from $\tau = 300$ to $\tau = 600$, while point-wise speaking there is an improvement in all cases except for I02 (60-300), I10 (600-1200), and I11 (300-600). For TPC we can see many improvements when raising τ as well, but with respectively 9 and 6 exceptions for BRKGA and VNS. These results lead us to claim that these two algorithms succeed in handling all the three objective parameters, even if they have different importance.

In general, the BRKGA obtained the best average results. This algorithm obtained the best results for the first objective, being dominant in the first three tables, and having drawn with MR in the last table ($\tau = 1200$). In addition, for time limits 600 and 1200, BRKGA obtained the best average results also for the second and third objective, providing a clear overall dominance when a large time limit is allowed.

Since production planning takes place on a daily time horizon and it is possible that the decision process must be carried out more than once, being subject to variations on the availability of workforce and orders requested, we suggest the value of the parameter $\tau = 600$. Indeed, the improvement in the transition from $\tau = 600$ to $\tau = 1200$ is not high enough to justify the additional time spent.

Table 2.7 summarises the quantitative and qualitative comparison between the three algorithms. Let $\sigma \in \{\text{BRKGA}, \text{MR}, \text{VNS}\}$. For each $\iota \in I'$, there is a best algorithm σ_{best} on that instance according to the objective function lexicographic minimisation procedure. We have in columns 'Count of Best' the number of times that an algorithm achieves the best solution on the instance set I'. We write $\nu_{\sigma_{\text{best}}}$ for the value of the best algorithm solution for that given instance ι for the parameter $\nu \in \{\text{TUS}, \text{TWT}, \text{TPC}\}$. For each instance we write $\nu_{\%,\sigma}$ for the relative gap from the best solution found for that instance by one of the three metaheuristics, i.e.:

$$\nu_{\%,\sigma}(\iota) = \frac{\nu_{\sigma}(\iota) - \nu_{\sigma_{\text{best}}}(\iota)}{\nu_{\sigma_{\text{best}}}(\iota)}.$$

As a consequence, values for $\nu_{\%,\sigma}$ close to zero mean proximity to the best algorithm (negative values meaning improvement on that parameter of the best algorithm, while positive values mean a worsening). Then we denote by $\overline{\nu_{\%}}_{,\sigma}$ the average gap of the parameter ν from the best solution for a given algorithm σ, taking from all instances solutions, i.e.:

$$\overline{\nu_{\%}}_{,\sigma} = \frac{1}{|I'|} \sum_{\iota \in I'} \nu_{\%,\sigma}(\iota).$$

As we can see, BRKGA succeeds in achieving the best results 13 times out of 23, on average, never ever dropping below 12, which is more than 50% of the instance set. Besides that, the average gap from best is the lowest among the three algorithms for any of the objective function parameters. Also, we mention VNS as a competitive approach, providing the best results on average for 8 instances.

Motivated by all the previous considerations, we suggest the BRKGA, with $\Pi_{\text{list}} = 5$ and $\tau = 600$.

TABLE 2.3: Metaheuristics results with $\tau = 60$.

Instance	$\|O\|$	TUS			TWT			TPC		
		BRKGA	MR	VNS	BRKGA	MR	VNS	BRKGA	MR	VNS
I01	168	0	0	0	0	0	0	7855.8	7918.9	7808.9
I02	158	0	1	1	703	0	0	11548.9	10740.0	10693.5
I03	194	0	1	1	0	5296	3967	11295.3	11371.3	11508.7
I04	253	0	0	0	0	0	0	7785.8	7779.9	7751.7
I05	169	0	0	0	0	0	0	9365.4	9446.0	9480.4
I06	176	0	0	0	0	0	293	11376.2	11278.8	11258.4
I07	153	0	0	0	0	0	0	10610.8	10801.4	10588.0
I08	164	1	1	0	0	0	130204	10151.9	10261.6	11874.5
I09	207	3	4	4	249195	209328	175298	11546.7	11450.1	11313.8
I10	194	10	11	11	327513	239432	182228	11369.3	11534.2	11791.1
I11	189	13	13	14	259382	262702	205462	11163.9	11509.0	11181.8
I12	160	0	0	0	0	0	0	7035.8	6994.0	7067.4
I13	178	13	13	13	0	0	0	6387.7	6500.3	6338.9
I14	180	8	8	8	0	0	0	6248.6	6344.4	6281.6
I15	183	8	8	8	0	0	0	7405.3	7504.7	7301.9
I16	201	8	8	8	0	0	0	9520.6	9642.2	9242.7
I17	267	8	8	8	0	0	0	9239.5	9355.5	9369.3
I18	102	0	0	0	0	0	0	4181.8	4400.0	4251.4
I19	154	0	0	0	15175	23257	11724	9522.0	9351.0	9696.4
I20	145	0	0	0	0	0	0	6747.9	6873.1	6805.7
I21	141	0	0	0	0	0	0	6265.2	6357.4	6319.7
I22	127	0	0	0	0	0	0	6174.1	6355.3	6365.9
I23	151	0	0	0	7409	13148	11154	9713.9	9487.4	9644.8
Average		3.1	3.3	3.3	37364.2	32746.2	31318.7	8804.9	8837.2	8866.8

TABLE 2.4: Metaheuristics results with $\tau = 300$.

Instance	$\|O\|$	TUS			TWT			TPC		
		BRKGA	MR	VNS	BRKGA	MR	VNS	BRKGA	MR	VNS
I01	168	0	0	0	0	0	0	7681.1	7747.3	7744.4
I02	158	0	0	0	456	9138	216	11543.9	11582.4	11453.9
I03	194	0	0	0	0	3517	1301	11046.4	11089.1	11584.7
I04	253	0	0	0	0	0	0	7665.2	7738.4	7572.2
I05	169	0	0	0	0	0	0	9217.6	9390.9	9469.2
I06	176	0	0	0	0	0	0	10422.9	10552.8	10140.8
I07	153	0	0	0	0	0	0	10497.5	10574.4	10584.0
I08	164	0	0	0	4028	0	0	11625.2	11790.2	11797.9
I09	207	3	4	4	238409	136520	175298	11548.8	11615.5	11313.8
I10	194	10	9	11	194080	250442	140656	11500.3	11296.9	11696.5
I11	189	11	12	14	290416	280744	190613	11585.8	11287.7	11065.0
I12	160	0	0	0	0	0	0	6885.7	6979.9	7015.2
I13	178	13	13	13	0	0	0	6362.4	6459.9	6301.6
I14	180	8	8	8	0	0	0	6236.4	6310.8	6227.6
I15	183	8	8	8	0	0	0	7342.1	7429.4	7290.9
I16	201	8	8	8	0	0	0	9180.3	9534.9	9178.8
I17	267	8	8	8	0	0	0	9134.8	9355.5	9241.8
I18	102	0	0	0	0	0	0	4181.8	4400.0	4251.4
I19	154	0	0	0	11118	16191	11079	9531.8	9735.2	9696.4
I20	145	0	0	0	0	0	0	6702.9	6786.6	6782.2
I21	141	0	0	0	0	0	0	6236.8	6279.6	6307.6
I22	127	0	0	0	0	0	0	6150.0	6280.5	6164.7
I23	151	0	0	0	6805	11488	1988	9667.9	9695.4	9503.6
Average		3.0	3.0	3.2	32404.9	30784.3	22658.7	8780.3	8865.8	8799.3

TABLE 2.5: Metaheuristics results with $\tau = 600$.

Instance	$\|O\|$	TUS			TWT			TPC		
		BRKGA	MR	VNS	BRKGA	MR	VNS	BRKGA	MR	VNS
I01	168	0	0	0	0	0	0	7676.5	7747.3	7742.5
I02	158	0	0	0	418	3518	27	11383.0	11650.2	11409.9
I03	194	0	0	0	0	3517	480	11006.4	11089.1	11515.3
I04	253	0	0	0	0	0	0	7575.2	7644.4	7446.4
I05	169	0	0	0	0	0	0	9206.9	9376.0	9416.8
I06	176	0	0	0	0	0	0	10193.6	10552.8	10061.3
I07	153	0	0	0	0	0	0	10482.5	10555.2	10548.1
I08	164	0	0	0	0	0	0	11292.4	11790.2	11729.8
I09	207	3	3	3	100074	283154	127785	11599.3	11777.1	11286.9
I10	194	10	9	11	129672	250442	140656	11134.4	11296.9	11694.7
I11	189	11	12	13	287399	280744	375688	11606.7	11287.7	11115.2
I12	160	0	0	0	0	0	0	6879.1	6979.9	7015.2
I13	178	13	13	13	0	0	0	6320.3	6430.4	6296.4
I14	180	8	8	8	0	0	0	6230.3	6276.2	6227.6
I15	183	8	8	8	0	0	0	7314.8	7398.8	7289.5
I16	201	8	8	8	0	0	0	9163.4	9344.2	9132.5
I17	267	8	8	8	0	0	0	9134.8	9298.8	9238.2
I18	102	0	0	0	0	0	0	4030.0	4400.0	4251.4
I19	154	0	0	0	11023	13654	2872	9540.4	9302.3	9602.0
I20	145	0	0	0	0	0	0	6679.1	6786.6	6759.4
I21	141	0	0	0	0	0	0	6236.8	6279.6	6307.6
I22	127	0	0	0	0	0	0	6147.5	6238.5	6164.7
I23	151	0	0	0	6805	6688	1508	9621.8	9589.4	9493.9
Average		3.0	3.0	3.1	23277.9	36596.4	28218.1	8715.4	8830.1	8771.5

TABLE 2.6: Metaheuristics results with $\tau = 1200$.

Instance	$\|O\|$	TUS			TWT			TPC		
		BRKGA	MR	VNS	BRKGA	MR	VNS	BRKGA	MR	VNS
I01	168	0	0	0	0	0	0	7668.9	7692.1	7718.8
I02	158	0	0	0	0	1240	20	11302.3	11434.5	11388.7
I03	194	0	0	0	0	3517	480	10998.2	11089.1	11515.3
I04	253	0	0	0	0	0	0	7569.0	7644.4	7377.1
I05	169	0	0	0	0	0	0	9166.8	9342.1	9222.4
I06	176	0	0	0	0	0	0	10069.0	10552.8	9995.9
I07	153	0	0	0	0	0	0	10376.5	10518.3	10432.3
I08	164	0	0	0	0	0	0	11125.2	11790.2	11699.6
I09	207	3	3	3	100072	237413	110931	11599.3	11333.1	11500.1
I10	194	9	9	9	137062	250442	242782	11517.3	11296.9	11686.6
I11	189	11	11	13	286679	277060	205486	11503.0	11600.2	11309.2
I12	160	0	0	0	0	0	0	6846.6	6979.9	6952.9
I13	178	13	13	13	0	0	0	6301.3	6428.4	6274.0
I14	180	8	8	8	0	0	0	6226.7	6276.2	6225.8
I15	183	8	8	8	0	0	0	7314.8	7398.8	7289.5
I16	201	8	8	8	0	0	0	9144.7	9344.2	9037.3
I17	267	8	8	8	0	0	0	9053.3	9298.8	9214.4
I18	102	0	0	0	0	0	0	4030.0	4400.0	4251.4
I19	154	0	0	0	11023	12709	2824	9540.4	9543.0	9602.0
I20	145	0	0	0	0	0	0	6668.4	6756.7	6758.6
I21	141	0	0	0	0	0	0	6236.8	6279.6	6307.6
I22	127	0	0	0	0	0	0	6147.5	6238.5	6164.7
I23	151	0	0	0	2293	4613	1174	9595.0	9685.4	9544.3
Average		3.0	3.0	3.1	23353.4	34217.1	24508.6	8695.7	8822.7	8759.5

TABLE 2.7: Comparative results.

τ	Count of Best			$\overline{\text{TUS}_{\%}}$			$\overline{\text{TWT}_{\%}}$			$\overline{\text{TPC}_{\%}}$		
	BRKGA	MR	VNS	BRKGA	MR	VNS	BRKGA	MR	VNS	BRKGA	MR	VNS
60	13	2	8	0.0	1.9	2.2	0.0	-2.8	-6.3	0.1	0.4	0.3
300	12	2	9	0.5	1.8	3.6	-1.0	-2.0	-4.6	0.0	0.6	0.6
600	13	1	9	0.5	0.4	1.8	-2.1	7.9	0.6	-0.1	1.0	0.9
1200	14	1	8	0.0	0.0	0.8	0.2	9.6	2.7	0.0	1.1	1.0
Average	13.0	1.5	8.5	0.2	1.0	2.1	-0.7	3.2	-1.9	0.0	0.8	0.7

2.6.3 Evaluation of BRKGA scheduling performance

In order to further motivate the choice of BRKGA among other metaheuristics, we investigate its scheduling performance by using the lower bounds provided by the implementation of the CP model described in Section 2.5, given the promising results obtained in Section 2.6.

For these experiments, we fixed the best workers configuration on machines given by the BRKGA with $\tau = 1200$. We set the same time limit also for the CP model implementation and compared the results for the primary objective TUS, re-running BRKGA with fixed workers. We summarise results in Table 2.8, where the lower bounds (LB) are obtained from the CP runs.

First, we can observe that BRKGA succeeds in finding a proven optimal solution within the given time limit in 20 instances out of 23, against 17 optimal solutions of the CP model. With respect to BRKGA, in the three unsolved instances, namely $I09, I10, I11$, neither the CP model succeeds in reaching the lower bound within the given time limit. Second, BRKGA achieves better upper bounds than CP in four instances, $I03, I08, I09, I22$. We also remark that in three among these cases, namely $I03, I08, I22$, the CP model fails to equal the lower bound within the given time limit, while BRKGA algorithm succeeds.

TABLE 2.8: CP and BRKGA lower bound comparison on the TUS for $\tau = 1200$.

Instance	$\lvert O\rvert$	LB	TUS	
	—	—	BRKGA	CP
I01	168	0	0	0
I02	158	0	0	0
I03	194	0	0	1
I04	253	0	0	0
I05	169	0	0	0
I06	176	0	0	0
I07	153	0	0	0
I08	164	0	0	2
I09	207	0	3	4
I10	194	2	9	8
I11	189	5	11	10
I12	160	0	0	0
I13	178	13	13	13
I14	180	8	8	8
I15	183	8	8	8
I16	201	8	8	8
I17	267	8	8	8
I18	102	0	0	0
I19	154	0	0	0
I20	145	0	0	0
I21	141	0	0	0
I22	127	0	0	1
I23	151	0	0	0

2.7 Conclusions

We studied a complex integrated workforce allocation and two-stage flexible flow shop problem to handle production planning for perishable products. We proposed a constructive algorithm that allocates workers to machines and schedules orders by following a list of priorities in order to reach solutions with all orders scheduled, no tardiness, and the smallest possible production cost. The constructive heuristic is embedded within three metaheuristics: a Random Multi-Start Algorithm, a Biased Random Key Genetic Algorithm, and a Variable Neighborhood Search based one.

To minimise tardiness, we based our overall constructive procedure on the concept of priority updating of orders, which led to the introduction of the parameter Π_{list}. The computational experiments indicate that a small value of Π_{list} is enough to consistently improve solutions. This limits computational effort, while preserving solution quality.

We compared the results of all metaheuristics under different time limits $\tau \in \{60, 300, 600, 1200\}$. A small time limit $\tau = 60$ is good enough to substantially improve the results of a single run of the constructive algorithm. This indicates that the constructive algorithm is sensitive to randomness, allowing for further investigation regarding the use of this algorithm within other metaheuristics. In addition, among the three proposed algorithms, the BRKGA provides the best overall results, and the algorithm converges very well already under a time limit of $\tau = 600$ seconds. For this reason, we compared the BRKGA performance in solving the scheduling problem with fixed workers to the lower bounds given by the implementation of the exact CP model.

In future research, we are interested in studying how dynamic changes in assignments of workers to machines affect productivity. Another direction is related to extend the algorithms to handle multiple objectives as, e.g., in [89] for the variable neighborhood search. We are also working with the company to develop a demand forecast approach. Such an approach would allow one to take important decisions at an earlier stage, and, when combined with the algorithms presented in this work, could further improve the production system of the company.

Chapter 3

Assigning multi-skill configurations to multiple servers

This chapter focuses on the dynamic assignment of multi-skill configurations to multiple servers in outpatient healthcare facilities. We develop the Scenario-Based Planning and Recombination Approach (SBPRA), an innovative algorithm that generalizes the Scenario-Based Planning Approach by leveraging a mathematical model to combine solutions across scenarios. This study expands on previous work and is detailed in [16, 19, 90].

3.1 Introduction

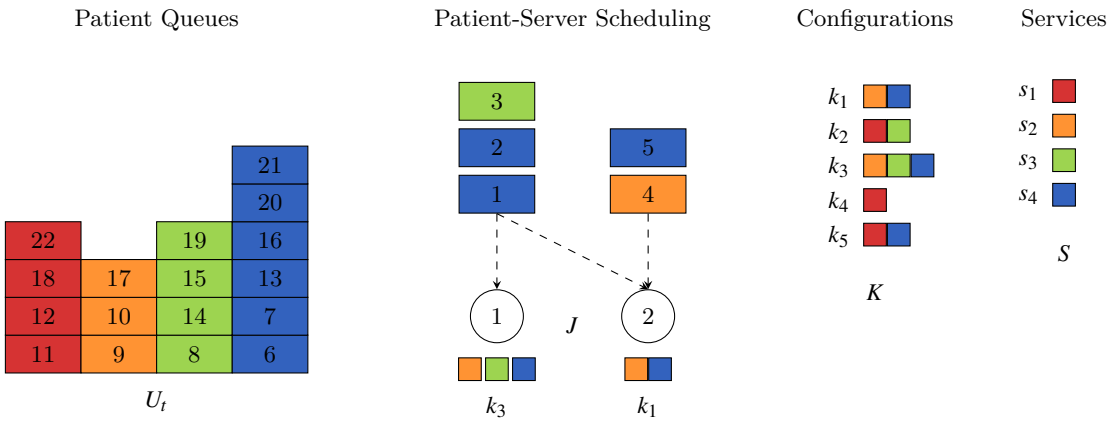
Queue management has been widely studied in many research fields because of its high practical and theoretical relevance. Problems in this area are found in several different contexts. One particular context is healthcare, where the presence of multiple queues and servers, the dynamic arrival of patients, the unknown service times, and the different priorities, among other features, provide several difficult challenges [38, 92, 127]. Queue management problems have been addressed in the literature by different approaches, particularly operations research methods (see, e.g., [122, 126]), which are indeed the focus of this work.

The problem we address focuses on queue management optimization in an outpatient facility system and arises from a real-world case study in Italy. Patients requiring specific services arrive dynamically during the working time horizon. They are then grouped into multiple queues, one per service, and are served by multiple identical servers working in parallel. At a given time instant, each server can provide a subset of all possible services the facility provides. This subset is called *configuration*. Thus, a patient can be served by a given server in a given time instant only if the server can provide at that time the service requested by the patient, that is, only if the configuration currently adopted by the server includes the requested service. At any time, it is possible that a patient decides to leave the queue. In this case, an abandonment penalty is considered.

Upon entry to the facility, each patient is associated with a requested service, a priority weight, and a target time. A tardiness occurs if the start of the service for a patient surpasses their target time, resulting in a weighted tardiness penalty. Additionally, a weighted penalty is applied for patient abandonment, which is triggered when a patient's waiting time surpasses her deadline. Notably, this deadline is a stochastic and dynamic parameter, not known a priori but only revealed when a patient opts to abandon the facility.

During the considered time horizon, the configurations associated with the servers are allowed to change. In practice, this possibility enables the facility managers to adapt the offered services according to the overall needs of the patients in the queues (and possibly of those expected to arrive soon). The possible configurations the servers can assume belong to a finite set and are known in advance, as the facility managers define them.

FIGURE 3.1: Outpatient facility example. Servers (J) are given by circles, patients by numbered rectangles, services (S) by colored squares, and configurations (K) by sequences of squares.



Server-Configurations Assignments

Thus, the decision to be made is twofold: first, which configuration to assign to each server on a rolling time horizon, and second, how to schedule patients to the servers. The aim is to minimize the sum of the total weighted tardiness of patients and the total weighted penalty for abandonment.

To better clarify the problem, let us consider the example depicted in Figure 3.1. In this facility there are two servers, and the overall provided services can be "blood tests" (s_1), "appointment booking from polyclinics" (s_2), "private practice"(s_3), and "biological material delivery"(s_4). At time t, configuration k_3 is assigned to server 1, enabling it to offer services s_2, s_3 and s_4. Similarly, configuration k_1 is assigned to server 2, with services s_2 and s_4. Consequently, at time t, patient 1 can be served by servers 1 and 2 because she requires service s_4, which is included in both configurations currently assigned to the servers, whereas patient 4 can only be served by server 2. Patients not served up to time t are inserted in a number of queues, each associated with a given service. Decisions regarding the order in which patients are served and by which server can be made by following a given policy (e.g., weighted first in, first out, or weighted earliest due date, as described in the next sections). These decisions are strongly constrained by the server-configuration assignments, and, therefore, the opportunity to change configurations during the time horizon may help in reducing the weighted tardiness and abandonment of patients.

The resulting problem integrates dynamic and stochastic elements, requiring decisions to be made online based on data not entirely known in advance. Many studies in healthcare have addressed problems of a dynamic or stochastic nature. Concerning hospitals and outpatient facilities (e.g., emergency departments), some authors have focused on the scheduling of patients [6, 36], nurses [65, 125], physicians [25, 61], and surgeries rooms [8, 21, 33]. Others study the optimization of given Key Performance Indicators (KPI), such as patients' waiting time and length of stay [2, 100, 116]. The literature considers methods based on combinatorial optimization, simulation, and a combination of both. A detailed literature review is provided in the next section.

To solve the overall problem, we propose a Scenario-Based Planning and Recombination Approach (SBPRA), a new algorithm that extends the traditional Scenario-Based Planning Approach (SBPA) from the literature [10]. Both approaches tackle the problem by generating possible scenarios based on known probability distributions and then solving them with an

optimization method. However, SBPRA not only solves each scenario independently but also recombines, with the use of a mathematical model, the solutions of the scenarios to produce an additional balanced solution. In both algorithms, the solution for each scenario is obtained by an updated version of the Reduced Variable Neighborhood Search (RVNS) that we originally presented in Alves de Queiroz, Bolsi, Lima, Iori, and Kramer [5]. We indeed generalize [5] to a great extent: we study a more general problem in which patients have a priority weight and may decide to abandon the system; we adapt the original RVNS to solve this new problem; we move from a simple re-optimization algorithm to the more sophisticated SBPA and SBPRA.

The main contributions of this work are: (i) the formal definition of a practical dynamic-stochastic problem derived from the activity of an outpatient facility; (ii) the proposal of SBPRA, an innovative algorithm that achieves results that are approximately 38% better on average than those obtained with the traditional SBPA; (iii) the development and assessment of various SBPA and SBPRA configurations, which include three consensus functions and three policies for scheduling patients; (iv) an extensive set of computational results on realistic instances from the outpatient facility, comprising more than 235000 numerical tests, that prove the effectiveness of the proposed algorithms and provide interesting managerial insights on the problem.

The remainder of the paper is organized as follows. Section 3.2 offers a comprehensive literature review, highlighting our contributions to the existing body of knowledge. Section 3.3 provides a formal definition of the dynamic-stochastic problem being investigated and introduces a mathematical model for its static variant, where all information is known at the beginning of the time horizon. Section 3.4 describes the re-optimization framework, the RVNS, the traditional SBPA, and the new SBPRA. This section also details scenario generation, scheduling policies, and the proposed consensus functions. Section 3.5 presents an extensive set of computational tests and evaluates the pros and cons of each scheduling policy, the influence of the number of scenarios, and the impact of each consensus function. Section 3.6 provides final remarks and directions for future research.

3.2 Literature review

This section reviews contributions from the healthcare planning and scheduling literature with a focus on queue management problems, with the aim of providing a brief but comprehensive review of the context of the problem addressed in this work. Important aspects that impact healthcare systems and outpatient facilities are related to, e.g., the waiting time of patients, the idle time of staff, the facility's dynamic environment, the adopted scheduling policies, and the level of stochastic information. Readers interested in deepening these aspects can refer to the recent survey by Youn, Geismar, and Pinedo [130].

Our review focuses on the most recent contributions to capacity management and scheduling in healthcare, but first of all, we would like to point out some of the seminal works in the field. The literature on healthcare problems dates back to the 1950s, with, e.g., Bailey [7] and Welch and Bailey [123] handling appointment scheduling problems in which patients are served on a first in, first out basis. In the same period, Lindley [68] investigated theoretical aspects of a single queue served by a single server, commenting that handling multiple queues and many servers would "be a problem of considerable difficulty".

At the end of the 1990s, Kaboudan [56] proposed a computer program to adjust dynamically the number of open servers depending on the number of customers in a single queue. The program was developed for systems with multiple servers sharing a common queue. The author observed better use of the servers and control of the queue length. A few years later, Cayirli and Veral [26] presented a comprehensive survey on appointment scheduling

in outpatient facilities. They discussed aspects of static and dynamic healthcare problems, pointing out that most literature contributions concern static problems. Dynamic aspects, e.g., unpunctual patients, no-shows, walk-ins, late doctors, and emergencies, may not be handled efficiently by the facility operations. Besides that, questions concerning the outpatient environments were detailed, such as the influence of the number of services, doctors, and appointments, the arrival process, the service times, and the schedule of patients. Performance indices to measure outpatient efficiency were also discussed, such as patients' waiting time, physicians' idle time, number of patients in the facility, and queues. One conclusion raised by the authors was related to closing the gap between theory and practice, e.g., by using historical data to better precise the probability distributions and develop heuristics that can handle the facility's dynamic situations and provide results for the daily issues. Within this regard, our paper contributes to the point raised by [26], as we use historical data to understand probability distributions and create scenarios that are used to develop efficient dynamic heuristics.

Different queueing models and their use for healthcare applications were described by Green [47]. The author commented on server utilization, timely access, and the facility size. Moreover, she discussed the importance of collecting and using (historical) data to meet time-varying demands, besides considering KPIs that could reflect patient perspectives. Klassen and Yoogalingam [59] handled the appointment scheduling problem with the proposal of a simulation optimization-based approach. In the simulation step, the authors created different scenarios using a neural network and setting relevant statistics. Next, optimization was performed on scenarios by means of a scatter search heuristic with tabu memory. The approach was compared with policies found in the literature aimed at minimizing the total cost of waiting and server idle time. The authors concluded that disruptive environments and heterogeneous patients could be interesting topics of research. Su, Yao, Su, Shi, Zhu, and Xue [105] studied the registration process of a hospital located in China through simulation. To reduce the patients' waiting time, the authors proposed to change from a multiple-queue and multiple-servers to a single-queue and multiple-servers system. Notably, in our work, we indeed consider heterogeneous patients and simulate a multiple-servers system that handles configurations of multiple queues.

The related nurse scheduling problem was addressed by, e.g., Wong, Xu, and Chin [125] and Legrain, Omer, and Rosat [65]. The former work proposed a binary linear formulation and a two-stage heuristic that first executes a shift assignment heuristic to obtain a solution satisfying the hard constraints and then performs a local search to meet the soft constraints and improve the solution. The authors showed that the two-stage heuristic could find good-quality solutions in short execution times. The latter work considered the presence of uncertainty. The authors proposed an online stochastic algorithm that generated feasible schedules at each time step of a given time horizon and evaluated them over the entire time horizon using sampled scenarios. Berg, Denton, Ayca Erdogan, Rohleder, and Huschka [11] considered booking, sequencing, and scheduling decisions to optimize patient reimbursements and costs associated with patient satisfaction, particularly waiting time, provider idle time, and overtime. The authors proposed three solution methods besides a two-stage stochastic mixed integer linear model to handle instances from the division of gastroenterology and hepatology at a clinic in Rochester, United States. They concluded that new research directions could focus on dynamic environments since, as in their case, static booking is less realistic. In Huang, Carmeli, and Mandelbaum [54], the problem of asymptotically evaluating optimal policies for scheduling multi-class patients to servers in emergency departments was handled. Under heavy traffic assumptions (leading to quasi-stationary hypotheses for the stochastic model), an asymptotically optimal policy was described and tested to minimize patients' queuing costs in the presence of deadline constraints.

In Dellaert and Jeunet [33], the authors proposed a mixed integer linear programming

model to solve a surgery scheduling problem. Since the size of instances the model could solve was limited, they also developed a variable neighborhood search heuristic to obtain good-quality solutions. The heuristic applied a shaking procedure only when the local search could not improve the incumbent solution. Habibi, Abadi, Tabesh, Vakili-Arki, Abu-Hanna, and Eslami [50] investigated the impact of appointment scheduling systems in ten outpatient clinics in Mashhad, Iran. The authors observed that only 57.7% of the patients were satisfied with their waiting time (an average time of 64 minutes to get served). Besides, service time and the clinic environment also greatly influenced patient satisfaction. In a similar way, Abaalkhayl, Al-Najjar, and Abukraym [1] analyzed patients' satisfaction concerning the services provided in the outpatient facilities of a care hospital in the Qassim region of Saudi Arabia. Our paper tries to mitigate the possible long waiting time by assuming patients may abandon the outpatient facility without being served, and this is highly penalized in the problem objective function.

Wang, Chen, and Xu [119] handled an online appointment scheduling problem by means of a dynamic programming model. The model suggested time slots for patients to book their services, aiming at maximizing the daily expected revenue. The authors considered the reward related to the patient and/or the available physicians. They also proposed an approximation method to reduce the number of system states in each booking period. They discussed the importance of future studies handling situations where patients book and receive care on the same day. Wen, Geng, and Xie [124] investigated how the arrival of urgent-level patients impacts the existing facility queues. The authors investigated how to insert these patients in order to minimize the total weighted waiting time of all patients and thus improve service quality. Considering a multi-server system in a dynamic environment, the authors proposed a heuristic policy integrating local searches and a simulation to schedule patients to servers. In our study, we do not deal with a booking system, and in addition to the total weighted waiting time, we also minimize the patient's abandonments.

Advances in queuing theory regarding applications in healthcare were reviewed by Worthington, Utley, and Suen [127], who focused on queuing systems with an infinite number of servers. The authors commented on the need for historical and real-time data in order to support operational decision-making. Di Mascolo, Martinez, and Espinouse [34] surveyed home healthcare problems concerning routing and/or scheduling problems. In these problems, service is provided to patients at their homes. The aim is to have a service whose quality is equivalent to that of a hospital while controlling costs and improving patients' conditions. Besides discussing details of the relevant literature, such as problems, constraints, and solution methods, the authors commented on the importance of handling uncertain data and stochastic or dynamic problems since less than 30% of the surveyed literature considered them. Abdalkareem, Amir, Al-Betar, Ekhan, and Hammouri [2] discussed planning and scheduling decisions regarding healthcare systems, paying attention to the improvement of the scheduling systems as a way to reduce the waiting time and facilitate access to services while avoiding staff overworking, patient dissatisfaction, and facility overcrowding. The authors surveyed scheduling problems concerning patients, nurses, operation rooms, surgeries, physicians, telemedicine, and home healthcare-related issues. They pointed out the lack of solution methods, especially metaheuristics, that can handle several hospital issues dynamically. Other challenges were related to taking advantage of historical data to propose new data sets and improve decisions regarding the many healthcare problems. We note that, also in this case, our paper advances the literature by proposing a dynamic approach based on a metaheuristic and testing it on real-world data.

Youn, Geismar, and Pinedo [130] reviewed healthcare problems, considering the past 30 years and focusing on planning and scheduling decisions. In problems with planning decisions, it is necessary to manage the healthcare resources to meet the demand for services. These may involve optimizing outpatient facilities, operating rooms, intensive care units, and

emergency departments. On the other hand, concerning scheduling decisions, the authors reviewed problems regarding appointment scheduling, surgery and workforce scheduling, and recurring and integrated scheduling. The authors commented on the small number of contributions integrating patient flow and scheduling decisions, especially in the presence of uncertainties. Choudhary, Shastri, Silswal, and Kulkarni [28] also surveyed scheduling problems, focusing on dynamic and unpredictable aspects (e.g., concerning patients and staff) that may affect healthcare systems. The authors commented on combining different solution methods to achieve high-quality results.

In the context of emergency departments, Bolandifar, DeHoratius, and Olsen [13] investigated the impact of relevant parameters, particularly waiting time, queue length, and service rate, on the patient abandonment time. The authors simulated the impact of patients' abandonments on the facility, concluding that abandonment assumptions should be taken into account when studying a facility's performance and modeling its dynamics. The context of emergency departments was also studied by Alves de Queiroz, Iori, Kramer, and Kuo [6], who focused on the dynamic problem of scheduling patients. The authors modeled the problem as a machine scheduling problem and proposed a re-optimization heuristic and an SBPA. Differently from these authors, we improve SBPA by adding an additional step to recombine and balance the solutions of scenarios. This step uses an integer linear programming model to minimize the additional configurations required to cover all requested services over all scenarios. A similar problem of scheduling patients in emergency departments was previously studied by Dosi, Iori, Kramer, and Vignoli [36]. They used a discrete event simulation model to test several organizational changes in a facility in northern Italy. The changes were modeled as what-if scenarios and evaluated using the proposed simulation model. The authors concluded that simple organizational changes can improve relevant KPIs, such as patients' waiting time and length of stay.

An operating room scheduling problem, including human resource availability constraints, is solved in Bargetto, Garaix, and Xie [8]. The authors proposed a branch-and-price-and-cut algorithm based on a time-indexed formulation, a label-correcting procedure for the pricing problem, and a Benders decomposition for the cutting procedure. The resulting framework could effectively handle all constraints and deliver good-quality solutions compared with the current solutions implemented at the facility.

3.3 Problem description

This section provides a formal description of the queue management problem we address. The facility receives many patients daily under a stochastic arrival distribution and serves them by means of multiple servers operating in parallel. When arriving at the facility, each patient is given a ticket for the requested service and then waits to be called by one of the servers. A priority weight is then assigned to the patient on the basis of the service she requested. A queue is associated with each service. If two or more patients request the same service, they are inserted in the queue associated with that service. Each patient has a target time, which depends on the service she requested, and the facility should serve the patient before it. Failing to meet target times produces a weighted tardiness. In addition, each patient has a deadline time. If the patient is still waiting in her queue when this deadline is met, then she leaves the facility without being served. This produces an abandonment penalty. The target time depends on the selected service and is defined by the facility managers, while the abandonment time depends on each patient and is unknown to the facility managers.

This problem is dynamic, with patients arriving over a time horizon, and has stochastic elements related to the arrival time and deadline of each patient. The other parameters are assumed to be deterministic but only revealed when the patient enters the facility and requests

a service. With the problem is associated a set S of services, a set J of identical parallel servers, a set K of configurations, a set I of patients, a time horizon T and a re-optimization time-step parameter Δ. A configuration $k \in K$ is a subset of services a server may hold. Sets S, J, and K are known in advance since they depend on the facility managers.

Each patient $i \in I$ is characterized by an arrival time r_i, a requested service $s_i \in S$, a priority weight w_i (indicating the urgency of the request), an expected service time p_i, a target time d_i (i.e., a due date), and an abandonment time a_i (i.e., a deadline). We denote by K_s the subset of configurations that include service $s \in S$. Thus, the subset K_{s_i} contains the configurations that can serve the request of patient i.

According to the facility managers, the configuration-server assignments are changed each Δ units of time. We assume the time horizon is discretized in minutes, namely $T = \{0, 1, \ldots, T_{\max}\}$, where 0 and $T_{\max}$ represent the opening and closing times of all servers. Patients can enter the facility until the doors close at $T'_{\max}$, so $r_i \leq T'_{\max}$ for all $i \in I$. When the servers close (i.e., at time $T_{\max}$), each patient will either have been served or have abandoned the facility, and so $a_i \leq T_{\max} - p_i$. We consider $T'_{\max}$ to be smaller enough than $T_{\max}$ to ensure the facility commitment to serving each patient, possibly after closing the doors.

We define the re-optimization time set $T_\Delta = \{0, \Delta, \ldots, \lfloor T_{\max}/\Delta \rfloor \Delta\} \subseteq T$, as the times in which the configurations can be changed. Patients can be partitioned accordingly, as $I = \cup_{t \in T_\Delta} I_t$, where $I_t = \{i \in I : r_i \in [t, t+\Delta)\}$. This does not imply that the problem can be solved independently for each Δ units of time, because previous decisions impact current and future ones. Decisions involve two key aspects: server-configuration assignments at each re-optimization time step in T_Δ, and patient-servers scheduling over the time horizon.

For the sake of clarity, we provide an integer linear programming (ILP) model for the static version (also known as offline or utopic in the literature) of our problem. This version assumes all information is precisely known at the beginning of the time horizon. The ILP model uses: a binary variable y_{jkt} that takes the value 1 if configuration $k \in K$ is assigned to server $j \in J$ from time $t \in T_\Delta$ to $t + \Delta - 1$; a binary variable x_{ijt} that takes the value 1 if patient $i \in I$ is served by server $j \in J$ at time $t \in T$; and an auxiliary binary variable u_i that takes the value 1 if $i \in I$ is not served by any server by the end of the time horizon. The ILP model is then:

$$\min \sum_{i \in I} \sum_{t \in T} w_i \tau_{it} \left(\sum_{j \in J} x_{ijt} \right) + \sum_{i \in I} \phi_i u_i \tag{3.1}$$

subject to:

$$\sum_{j \in J} \sum_{t \in T} x_{ijt} + u_i = 1 \qquad \forall i \in I \tag{3.2}$$

$$x_{ijt} = 0 \qquad \forall i \in I, \forall j \in J, \forall t \notin \{r_i, \ldots, a_i\} \tag{3.3}$$

$$\sum_{i \in I} \sum_{t' = \max\{t - p_i + 1,\, 0\}}^{t} x_{ijt'} \leq 1 \qquad \forall j \in J, \forall t \in T \tag{3.4}$$

$$\sum_{k \in K} y_{jkt} = 1 \qquad \forall j \in J, \forall t \in T_\Delta \tag{3.5}$$

$$\sum_{t' = \max\{t - p_i + 1,\, 0\}}^{t + \Delta - 1} x_{ijt'} \leq \sum_{k \in K_{s_i}} y_{jkt} \qquad \forall i \in I, \forall j \in J, \forall t \in T_\Delta \tag{3.6}$$

$$x_{ijt} \in \{0, 1\} \qquad \forall i \in I, \forall j \in J, \forall t \in T \tag{3.7}$$

$$y_{jkt} \in \{0, 1\} \qquad \forall j \in J, \forall k \in K, \forall t \in T_\Delta \tag{3.8}$$

$$u_i \in \{0, 1\} \qquad \forall i \in I. \tag{3.9}$$

The objective function (3.1) minimizes the total weighted tardiness and the total weighted penalties for unserved patients. Tardiness is calculated as $\tau_{it} = \max\{t - d_i, 0\}$ for patient i served at time t. The weighted penalty for patients that have abandoned the facility is denoted by ϕ_i and is set to $\phi_i = w_i T_{\max}$, thus ensuring that an abandonment is always penalized more than a late service. Constraints (3.2) ensure that either a patient has been served or has abandoned the facility. Constraints (3.3) eliminate situations in which a patient is served before her release date or after her abandonment time. Constraints (3.4) guarantee that a server can service at most one patient at a given time and that that patient occupies the server for all her service time. Constraints (3.5) impose that each server is assigned with a unique configuration and can change only in times $t \in T_\Delta$. Constraints (3.6) impose that a patient i is served by a server j at time t only if this server holds at that time a configuration that offers the service required by i. Constraints (3.7)-(3.9) define the domain of the variables.

3.4 Scenario-Based Planning and Recombination Approach

When not all problem information is known in advance, but just probability distributions are known, a decision problem can be formulated as a two-stage stochastic ILP model. In this model, decision variables are divided into *first-stage*, referring to decisions that can be made now without knowing the future, and *second-stage* variables, concerning decisions to be taken after the realization of the stochastic variables. In our context, the first-stage variables are related to assigning configurations to servers at each re-optimization step (namely, $Y_t = \{y_{jkt} : j \in J, k \in K\}$). In contrast, second-stage variables, which involve patient scheduling, are determined after stochastic information about the patients becomes available (resulting in $X_t = \{x_{ijt} : i \in I, j \in J\}$).

Typically, stochastic ILP models require large execution times and are more concerned with strategic planning, whereas operational decisions are taken by means of quick dynamic heuristic algorithms (see, e.g., [115]). Among the dynamic heuristics, the SBPA by [10], able to address complex decision-making dynamic-stochastic problems, has received significant attention in the literature. Similar to two-stage ILP models, SBPA explores potential scenarios that realize the stochastic variables. Given the dynamic nature of these problems, SBPA is typically incorporated within a re-optimization framework. Generally, the following steps are performed at each re-optimization time t:

1. **Scenario Sampling.** The events space is sampled in a set Ξ of scenarios. Each scenario $\xi \in \Xi$ is a realization of the stochastic variables obtained by sampling the probability distributions (in our case, we do this by using the facility's historical data);

2. **Scenario Solution.** Each scenario $\xi \in \Xi$ is solved taking into consideration the current (partial) problem solution Y associated with the first-stage decisions, resulting in the scenario solution $Y_t(\xi)$. The scenario is solved with any appropriate optimization algorithm (e.g., RVNS, as in our case);

3. **Solution Evaluation.** A score is assigned to each scenario solution $Y_t(\xi)$ with a so-called *consensus function* ω. This function evaluates the solution of a scenario by confronting it with all other scenario solutions;

4. **Solution Update.** The current problem solution is updated by observing the highest $\omega-$score scenarios solutions in the set $\{Y_t(\xi)\}_{\xi \in \Xi}$. This may require refining, modifying, and/or combining these scenario solutions and then choosing the one to be implemented on the problem at hand, eventually updating first-stage and second-stage variables.

In this work, we propose the SBPRA, an enhanced version of the above-mentioned classical SBPA. This new version includes an additional step between the *Scenario Solution* and *Solution Evaluation* steps, specifically designed to recombine scenario solutions and then improve first-stage decisions. In this new step, called *Solution Recombination*, a new solution Y_t^R is generated and added as part of the set of scenario solutions. The main idea is to take advantage of the information in the sampled scenarios and find an optimally balanced scenario solution that mitigates the risk of not well-hedging the realization of stochastic variables, which may happen for solutions of a single scenario.

In the *Solution Recombination* step, we consider hedging parameters of every scenario solution (in our case, they are the number of servers offering a given service if requested in such a scenario, as described next). Then, by considering the first-stage decisions from each solution, we minimize the hedging loss relative to these parameters (in our case, aiming at balancing the number of additional servers needed by each scenario) over all scenarios. In other words, this step handles a *Hedging-Balancing problem* over all scenarios (which is solved with the ILP model (3.10)-(3.18) below).

We first describe the SBPRA in Algorithm 3. Its procedures are presented in the next sections, with the RVNS detailed in Algorithm 4 and the cost function described in Algorithm 5. Table 3.1 contains the parameters not mentioned before but needed for describing the proposed algorithm.

TABLE 3.1: Parameters used by the proposed algorithms.

Parameter	Description
π	Policy applied to schedule patients to servers
U_t	Set of patients not served up to time t, for $t \in T$
$I_t(\xi)$	Set of fictive patients of scenario $\xi \in \Xi$, with arrival in $\{t,\ldots,t+\Delta-1\}$, for $t \in T_\Delta$
$Y_t(\xi)$	Solution of scenario $\xi \in \Xi$, i.e., the configuration-server assignments at time $t \in T_\Delta$
$\overline{M_s}(\xi)$	Service hedging parameter for service $s \in S$ in solution $Y_t(\xi)$ at time t, which is equal to the number of servers that offers service s in such a solution if at least a patient requires s, and 0 otherwise
$\overline{M}(\xi)$	Set $\{\overline{M_s}(\xi), s \in S\}$ with all service hedging parameters for solution $Y_t(\xi)$ at time t
Y_t^R	Solution generated by the Hedging-Balancing problem at time $t \in T_\Delta$

Algorithm 3 starts with an empty patient scheduling solution X and configuration assignment solution Y (line 1). Next, it iterates over the time horizon T. For each time step t, it identifies the set U_t of patients not served up to that time and schedules them to servers based on the current configuration solution Y and by using policy π. The servers can update their configurations if t is a multiple of Δ (line 6). In this case, we apply the RVNS to compute the scenario solution. The RVNS receives as input the set with sampled fictive patients and the (real) patients not served yet. Observing the first-stage decisions $Y_t(\xi)$ of each scenario solution, the service hedging parameter $\overline{M_s}(\xi)$ is calculated in line 11. After looping up over all scenarios, the *Solution Recombination* step is applied to solve exactly the Hedging-Balancing model (3.10)-(3.18) in line 12. The resulting solution Y_t^R is added to the set of scenario solutions, all being scored by the same consensus function (line 13). The solution with the best score is chosen, so the configuration-server assignment is updated accordingly. In line 15, the algorithm computes $U_{T_{\max}}$, with all patients not served up to the time $T_{\max}$. Notice the final solution cost is obtained from the patient scheduling solution X and $U_{T_{\max}}$.

Scenario Sampling

For SBPA, a sampled scenario $\xi \in \Xi$ consists of fictive patients arriving at the facility over the time horizon. We generated scenarios using the probability distributions obtained from the

Algorithm 3: SBPRA

Input: I; J; K; $T_{\max}$; Δ; Ξ; π; ω

$X, Y \leftarrow \emptyset$
foreach $t \in T$ **do**
  $U_t \leftarrow$ patients not served up to time t
  $X_t \leftarrow$ Apply policy π to schedule patients in U_t to servers according to the current solution Y
  $X \leftarrow X \cup X_t$
  if $t \in T_\Delta$ **then**
    foreach $\xi \in \Xi$ **do**
      $I_t(\xi) \leftarrow$ Fictive patients whose arrival time are in $\{t, \ldots, t+\Delta-1\}$
      $U \leftarrow U_t \cup I_t(\xi)$
      $Y_t(\xi) \leftarrow \text{RVNS}(U)$
      $\overline{M}(\xi) \leftarrow$ Hedging parameters from the scenario solution $Y_t(\xi)$
    $Y_t^R \leftarrow$ Solution of the Hedging-Balancing problem for $\{\overline{M}(\xi)\}_{\xi \in \Xi}$
    $Y_t \leftarrow$ Solution in $\{Y_t(\xi)\}_{\xi \in \Xi} \cup \{Y_t^R\}$ with the best score according to the consensus function ω
    $Y \leftarrow Y \cup Y_t$
$U_{T_{\max}} \leftarrow$ patients not served up to time $T_{\max}$

Output: X; Y; $U_{T_{\max}}$

facility's historical data when available. For example, no data about abandonment time were tracked, so we supposed realistic values by discussing them with operators at the facility. On the other hand, consistent data are available for arrival times and required services for quite a long period. We estimated parameters that best fitted chosen distributions (described in Section 3.5.1 below) for each attribute. This mechanism ensures the representativeness of scenarios because each scenario is a random possible realization of joint distributions of single stochastic variables, thus well-defining patient attributes and the overall situation the facility faces daily. So, besides sampling the arrival time of fictive patients i, we also sampled their other parameters (i.e., s_i, w_i, p_i and d_i). The abandonment time a_i is a uniform random integer value picked in the interval $[d_i + 1,\ T_{\max} - p_i]$.

The number $|\Xi|$ of sampled scenarios has a relevant influence on the solution quality and computing time. We test several $|\Xi|$ values and evaluate their impact in Section 3.5.2.

Scenario Solution

Each scenario could be seen as a static version of the problem, which has to be partially or completely solved by an optimization method. Many scenario solutions are generated in a re-optimization framework, requiring computationally efficient optimization methods. In this work, we solve each scenario with an enhanced version of the RVNS we introduced in Alves de Queiroz, Bolsi, Lima, Iori, and Kramer [5]. In contrast with the precedent version, the new RVNS, described in Algorithm 4, considers patient abandonment times and priority weights.

Variable neighborhood search (VNS) is a metaheuristic algorithm that works over a set of neighborhoods. It systematically changes the neighborhood whenever the incumbent solution is not improved. The aim is to escape from local optima and achieve a solution that is optimal for all neighborhoods. It was proposed by Mladenović and Hansen [78] and, since then, has been successfully applied to handle many optimization problems. Comprehensive reviews dedicated to VNS are provided in, e.g., Hansen and Mladenović [51], Hansen,

Mladenović, and Moreno Pérez [52], and Hansen, Mladenović, Todosijević, and Hanafi [53]. Special issues dedicated to the VNS can be found in Duarte and Pardo [37] and Mladenović, Souza, and Sörensen [79], while Lan, Fan, Yang, Pardalos, and Mladenovic [64] reviewed the applications of VNS concerning problems in the healthcare area. Recently, Brimberg, Salhi, Todosijević, and Urošević [23] revisited the VNS principles and variants, commented on the variable formulation space and its implication on solving problems that may have alternative formulations, and discussed the less-is-more approach in the perspective of proposing effective but simple algorithms.

A VNS-based heuristic has the following phases: *shaking*, *local search*, and *change of neighborhood*, inside an iteration loop with a given *stopping criterion* (e.g., the maximum number of iterations). The shaking phase generates perturbed solutions within a neighborhood. Next, the local search phase is performed to improve the perturbed solutions. If an improved solution is found, the search restarts from the first neighborhood; otherwise, it proceeds to the next neighborhood. The local search phase can be a computational bottleneck, especially in dynamic problems. This motivates the use of a reduced VNS (i.e., RVNS), which contains only the shaking and change of neighborhood phases. The RVNS often mitigates the computational effort while still keeping good solutions, justifying its use to solve scenarios in the proposed SBPRA.

Algorithm 4: RVNS TO OBTAIN A SOLUTION TO SCENARIO ξ

Input: U; $It_{\max}$; $\mathrm{VN}_{\max}$
1 $Y \leftarrow$ randomly generated solution of configuration-server assignments
2 **for** $it \leftarrow 1, 2, \ldots, It_{\max}$ **do**
3 $n \leftarrow 1$
4 **do**
5 $Y' \leftarrow$ neighbor solution of Y in neighborhood VN_n `// Shaking phase`
6 **if** $cost(Y') < cost(Y)$ `// Change of neighborhood`
7 **then**
8 $Y \leftarrow Y'$; $n \leftarrow 1$
9 **else**
10 $n \leftarrow n+1$
11 **while** $n \leq VN_{\max}$
Output: Y

The RVNS in Algorithm 4 generates a solution $Y = Y_{t_{\text{now}}}$ to scenario ξ giving the configuration-server assignments to be used at the current time t_{now}. Solution Y is coded as a vector of $|J|$ integers, in which each vector position is associated with a server j and reports the number of the configuration k assigned to the server. The shaking phase of the RVNS has $\mathrm{VN}_{\max} = 3$ and uses the following neighborhoods:

VN_1: Randomly select a server and change its configuration to a randomly selected configuration in K. The selected configuration must be different from the current one;

VN_2: Randomly select two servers and set the first server to have the same configuration as the second one;

VN_3: Randomly select a subset of servers and change the configuration of each of these servers to a randomly selected configuration in K. The new configuration of a server may remain the same as the current one.

We tested other neighborhoods in preliminary computational tests. Some structures were based on setting weights for configurations. Configurations with the most requested services

were associated with the highest weights (i.e., they were more likely to be chosen). Other neighborhoods we tested were based on duplicating servers having the configurations with the highest weights. None of these improved the final solution as expected, but instead, they even biased the choices, resulting in configurations never being used even if services (patients) depended on them. Based on these preliminary computational analyses, we decided to keep the current simple but effective configuration with three neighborhoods.

Each time we generate a new solution, in line 6 of Algorithm 4, for the current time t_{now}, we evaluate its cost with Algorithm 5. This algorithm calculates the cost by simulating what is going to happen in the facility from t_{now}. It first assumes that past decisions (real patients already served) and decisions in progress (real patients in service) cannot be changed anymore. Then, it schedules the set U, comprising real patients (already arrived at t_{now} at the facility and currently waiting in the queues) and fictive patients (expected to arrive soon after t_{now} and depending on the given scenario), as follows. In the loop of lines 3-7, for each time t, Algorithm 5 schedules the remaining patients in U observing the policy π and the free (i.e., not occupied with a patient) servers. The policy π is used to choose the next patient that fits one of the configurations assigned to the free servers. The process continues until $T_{\max}$ is reached. Variables WT and Φ report, respectively, the total weighted tardiness and total weighted penalty, which are the two components of the solution cost.

We implemented three different policies π to select the next patient to be served:

Weighted First In First Out (wFIFO). The queue of eligible patients in U is ordered by decreasing values of the ratio $\frac{r}{w}$, which weighs the patients' arrival time by their priority weight;

Weighted Earliest Due Date (wEDD). The queue of eligible patients in U is ordered by decreasing values of the ratio $\frac{d}{w}$, which weighs the patients' target time by their priority weight;

Weighted Longest Processing Time (wLPT). The queue of eligible patients in U is ordered by decreasing values of the ratio $\frac{p}{w}$, which weighs the patients' service time by their priority weight. If two patients have the same service time, the one with the largest weight is selected.

In all three policies, if two patients have the same value of the selected ratio, the one with the largest weight is selected.

Algorithm 5: COST FUNCTION

Input: Y; U; π; t_{now}

1 $WT \leftarrow 0$ `// Weighted tardiness`
2 $\Phi \leftarrow 0$ `// Penalty due to unserved patients`
3 **for** $t \leftarrow t_{now}, t_{now}+1, \ldots, T_{\max}$ **do**
4 $J_{free} \leftarrow$ Set of servers free at time t
5 **foreach** $j \in J_{free}$ **do**
6 Apply police π to select the next patient $i \in U$ to be served by j
7 $WT \leftarrow$ add the weighted tardiness of i, if any
8 $\Phi \leftarrow$ total weighted penalty for all unserved patients

Output: $(WT + \Phi)$

Scenario Recombination

We propose the *Scenario Recombination* step to exploit further the solutions obtained from the sampled scenarios. We aim to avoid relying solely on decisions from a single scenario

solution, e.g., the one with the best score according to the selected consensus function. Decisions based on a single scenario solution are often non-representative of the entire set of scenarios, as a single scenario may overlook important aspects of the problem, such as not including patients who abandon the facility or those requesting less common services.

The recombination step consists of solving an optimization problem, which we call *Hedging-Balancing*, aimed at balancing the configurations across servers to hedge against solutions from all scenarios. Specifically, this model minimizes the number of additional configurations required to cover all requested services in each scenario.

For a given $\bar{t} \in T_\Delta$, the associated hedging-balancing problem is modeled as an ILP in (3.10)-(3.18). This model does not depend on the number of patients but only on the number of servers and configurations, and it uses the following variables:

- an integer variable N_k, giving the number of servers with configuration k, for $k \in K$;
- an integer variable M_s, giving the number of servers offering service s, for $s \in S$;
- an integer variable $\delta_s(\xi)$, giving the number of additional servers with configurations that contain service s that are required, for $s \in S$ and $\xi \in \Xi$;
- a binary variable $y^R_{jk\bar{t}}$, taking the value 1 if configuration k is assigned to server j, 0 otherwise, for $j \in J$ and $k \in K$.

By solving the hedging-balancing problem, we obtain a recombination solution $Y^R_{\bar{t}} = \left\{ y^R_{jk\bar{t}} : j \in J, k \in K \right\}$, associated with the first-stage decision variables for the time interval $[\bar{t},\ \bar{t}+\Delta)$. This solution should be balanced enough to hedge all requested services in all scenarios. Variables $\delta_s(\xi)$ measure the loss of not appropriately hedging service s, if s is requested in scenario ξ and there are no sufficient servers with configurations in K_s.

Formally, the model is:

$$\min \sum_{s \in S} \sum_{\xi \in \Xi} \delta_s(\xi) \tag{3.10}$$

subject to:

$$M_s + \delta_s(\xi) \geq \overline{M_s}(\xi) \qquad \forall s \in S, \xi \in \Xi \tag{3.11}$$

$$M_s = \sum_{k \in K_s} N_k \qquad \forall s \in S \tag{3.12}$$

$$\sum_{k \in K} N_k = |J| \tag{3.13}$$

$$N_k = \sum_{j \in J} y^R_{jk\bar{t}} \qquad \forall k \in K \tag{3.14}$$

$$\delta_s(\xi) \in \mathbb{Z}_{\geq 0} \qquad \forall s \in S, \xi \in \Xi \tag{3.15}$$

$$N_k \in \mathbb{Z}_{\geq 0} \qquad \forall k \in K \tag{3.16}$$

$$M_s \in \mathbb{Z}_{\geq 0} \qquad \forall s \in S \tag{3.17}$$

$$y^R_{jk\bar{t}} \in \{0,1\} \qquad \forall j \in J, \forall k \in K. \tag{3.18}$$

The objective function (3.10) seeks to minimize the sum of the hedging loss variables $\delta_s(\xi)$. Constraints (3.11) impose each service s requested in each scenario ξ on being hedged. If there are sufficient servers with configurations that contain s in the recombination solution $y^R_{jk\bar{t}}$, given the existing number of servers in the scenario solution that offers s (i.e., if $M_s \geq \overline{M_s}(\xi)$), then the hedging loss variable associated with s and ξ is null, otherwise it takes a positive value, whose sum has to be minimized. Constraints (3.12) ensure that each

variable M_s holds exactly the number of servers with configurations that contain s in the recombination-solution. Constraints (3.13) impose the total number of servers with an assigned configuration on being equal to the number $|J|$ of available servers. Constraints (3.14) ensure that the number of servers assigned with configuration k corresponds exactly to that in the recombination solution $y^R_{jk\bar{t}}$, for each k. Constraints (3.15)-(3.18) define the domain of the variables.

FIGURE 3.2: Illustrative example of the *Scenario Recombination* step for a given re-optimization time $t \in T_\Delta$.

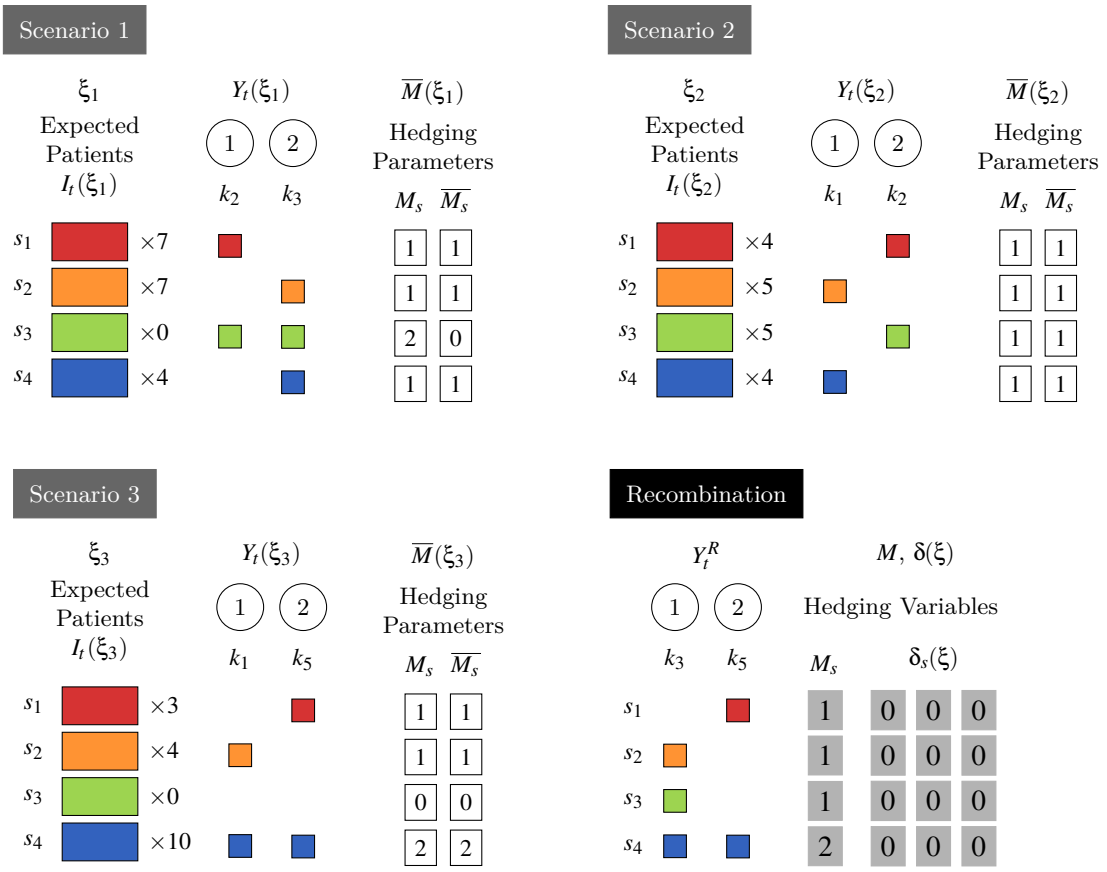


Figure 3.2 presents a solution to the Hedging-Balancing problem, assuming the same servers, services and configurations of Figure 3.1. We consider a set $\Xi = \{\xi_1, \xi_2, \xi_3\}$ with three scenarios. Scenario ξ_1 involves services s_1, s_2 and s_4, requested by, respectively, 7, 7, and 4 fictive patients, all supposed to arrive within the time interval $[t,\ t+\Delta)$. Similarly, scenario ξ_2 involves services s_1, s_2, s_3 and s_4, requested by 4, 5, 5 and 4 patients, respectively, and scenario ξ_3 involves services s_1, s_2 and s_4, requested by 3, 4 and 10 patients, respectively. The scenario solutions are obtained with the RVNS. The first-stage solution $Y_t(\xi_1)$ of scenario ξ_1 assigns configuration k_2 to server 1 and k_3 to server 2. Since only one server is available to attend s_1 and ξ_1 contains fictive patients requesting this service, the hedging parameter $\overline{M_{s_1}}(\xi_1)$ is set to 1. We calculate $\overline{M_{s_1}}(\xi_2)$ and $\overline{M_{s_1}}(\xi_4)$ in a similar way. Note, however, that in this solution, two servers are offering service s_3 but no patient is requiring it, resulting in $\overline{M_{s_3}}(\xi_1) = 0$ (refer to Table 3.1 for the parameter definition). After computing these parameters for all scenarios and solving the Hedging-Balancing problem, the resulting solution assigns k_3 to server 1 and k_5 to server 2. Consequently, the hedging variables associated with the services take the values $M_{s_1} = 1$, $M_{s_2} = 1$, $M_{s_3} = 1$ and $M_{s_4} = 2$. The hedging loss variable $\delta_{s_1}(\xi)$ of service s_1 and concerning all three scenarios is equal to 0 because the

recombination solution can hedge each scenario solution, that is, $M_{s_1} \geq \overline{M_{s_1}}(\xi_i)$ for $i = 1,2,3$. The same holds for the other hedging loss variables associated with the remaining services. Note that this solution is different from any other scenario solution. Moreover, among the possible assignments, it is the only one that guarantees a zero sum of hedging loss variables.

Solution Evaluation and Update

Once the RVNS has produced a solution for each scenario and the recombination solution Y^R has been calculated, we decide which configuration-server assignments are derived from these solutions to apply to the instance being solved. To this aim, we first associate a score with each solution in the set of augmented scenarios solutions $Y^R(\Xi) = \{Y^R, Y(\xi) : \xi \in \Xi\}$ by applying a given consensus function. Then, we select which solution to use (according to line 13 in Algorithm 3).

To this aim, we implemented three consensus functions, denoted as average best, best case, and worst case. In these functions, given the time t, each scenario solution $Y_t \in Y^R(\Xi)$ is fixed as a first-stage decision to all other scenarios $\xi \in \Xi$ and second-stage optimization decisions are taken for each scenario by using Algorithm 5, so that an expected objective cost $z_t(Y_t, \xi)$ is obtained. Then, depending on the specific consensus function, we compute

Average best (AVG): $\omega(Y_t) = \frac{1}{|\Xi|} \sum_{\xi \in \Xi} z_t(Y_t, \xi)$;

Best case (BEST): $\omega(Y_t) = \min_{\xi \in \Xi} z_t(Y_t, \xi)$;

Worst case (WORST): $\omega(Y_t) = \max_{\xi \in \Xi} z_t(Y_t, \xi)$;

and we finally select the solution to be implemented as $\overline{Y_t} = \arg\min \omega(Y_t)$.

3.5 Computational experiments

This section presents the outcome of the computational experiments we performed to evaluate the SBPRA performance and assess the impact of its main components. All algorithms were coded in C++, and the experiments were carried out on a computer equipped with an Intel Xeon CPU E3-1245 v5 processor with 3.50 GHz and 32 GB of RAM, running under Linux Ubuntu 22.04 LTS.

We performed the experiments on 61 realistic instances (described in the next section). For the sake of conciseness, the results we obtained are presented in aggregated form. We solved each instance with the following algorithms: (i) the plain re-optimization version of Algorithm 3 in which $|\Xi| = 0$, called RE-OPT from now on; (ii) SBPA (without the *Recombination Solution* step) for different number of scenarios ($|\Xi|$) and consensus functions (ω), called AVG, BEST, and WORST, respectively; (iii) SBPRA for different number of scenarios and consensus functions, called AVG-R, BEST-R, and WORST-R, respectively.

All mentioned algorithms solved each instance individually. For SBPA and SBPRA, which need in input $|\Xi|$ scenarios, we proceeded as follows. We generated a large number of scenarios by sampling the probability distributions at the facility. Then, assuming we are solving, e.g., instance α, each scenario $\xi \in \Xi$ is randomly selected among all the generated scenarios but by excluding instance α.

The RVNS parameters were calibrated by a trial-and-error procedure considering a subset of instances, balancing solution quality and computing time. After the calibration procedure, we set $It_{\max} = 250$ iterations, and for neighborhood VN_3 we limited the size of the subsets to be in $\{2,3,\ldots,\lfloor |J|/4 \rfloor\}$.

We also solved the instances in their static version by means of the ILP model (3.1)-(3.9). The model was solved with Gurobi Optimizer 11, imposing a single thread and a time limit

of one hour. The results were unsatisfactory, and we do not report them explicitly here. Indeed, the model always reached the time limit of one hour and obtained very large percentage gaps between the reported lower and upper bound values, ranging from an average of 379% for Δ = 60, to 1991% for Δ = 120 and 1911% for Δ = 180. In addition, the solutions obtained were very unrealistic, as the model, in order to minimize the number of abandonments (which cause the largest penalties) selected from the middle of the queues patients with close deadlines even if they had lower priorities and arrived at the facility later than other patients (whose service was just postponed). Such a behavior would be clearly unacceptable in a real system.

3.5.1 Instance generation

The instances used in our computational experiments were collected from an outpatient facility system operating in northern Italy. The collected data corresponds to working days in December 2019. The facility is open from 6 a.m. to 5 p.m. and has 13 servers. In the following, we denote by H the set of instances and use a tuple $h = (I, J, S, K, T_{\max})$ to define each instance $h \in H$.

Analysis of the historical data suggests that the time interval between subsequent arrivals of patients could be modeled by a Weibull distribution. This distribution is defined by a cumulative density function $f(x) = 1 - e^{-\left(\frac{x}{\lambda}\right)^{\kappa}}$, with the so called shape λ and scale κ parameters. The Weibull distribution generalizes the exponential one, because when $\kappa = 1$ it results in an exponential distribution with mean equal to λ. Indeed, Weibull and exponential distributions are commonly used to model the inter-arrival time of patients, especially in the context of hospitals, emergency departments, and outpatient facilities (see, e.g., [6, 62, 63]). We observe that the number of patients varies during the facility's working hours, so we discretized the inter-arrival times by intervals of one hour each, whose shape and scale parameters are given in Table 3.2. Note that this results in an average of 1557 patients visiting the facility daily, with more patients in the morning, fewer in the afternoon, and even fewer at lunchtime.

TABLE 3.2: Average inter-arrival times per hour, following a Weibull distribution.

Time interval	Average arrivals	Shape λ	Scale κ
6 a.m. - 7 a.m.	114	11.78	0.43
7 a.m. - 8 a.m.	134	23.54	0.78
8 a.m. - 9 a.m.	253	13.32	0.87
9 a.m. - 10 a.m.	234	14.56	0.89
10 a.m. - 11 a.m.	173	20.16	0.92
11 a.m. - 12 a.m.	126	27.63	0.92
12 a.m. - 1 p.m.	72	47.65	0.92
1 p.m. - 2 p.m.	102	32.81	0.91
2 p.m. - 3 p.m.	149	22.97	0.86
3 p.m. - 4 p.m.	120	28.31	0.86
4 p.m. - 5 p.m.	80	42.91	0.90

Table 3.3 details the parameters and values associated with the outpatient services from historical data, which we used to generate patient information. For each patient $i \in I$, the requested service s_i was randomly generated following a multinomial distribution with weights given by column *Requests*. Then, the weight w_i, the target time d_i, and the service time p_i were generated in accordance with the patient service s_i. The priority weight w_i was randomly generated with a distribution given by the product between $\overline{w}$ and a uniform random

variable in the range $[0.5, 1.5]$. The abandonment time a_i was generated as an integer value obtained from a uniform random variable in the interval $[d_i + 1,\ T_{\max} - p_i]$.

TABLE 3.3: Information on services requested in the outpatient facility.

s	Service	Requests [%]	p [min]	$d-r$ [min]	$\overline{w}$
1	Outpatient reception	34.48	1.47	20	5
2	Polyclinic reception	19.08	1.15	20	5
3	Blood tests	13.38	2.40	10	10
4	Appointment booking from polyclinics	8.32	4.03	20	5
5	Appointment booking from outpatient clinics	6.99	3.02	20	5
6	Private practice	5.14	4.32	5	20
7	Frequent users' reception	3.64	0.28	20	5
8	Biological material delivery	3.51	2.35	15	6
9	Priority blood tests	2.45	2.40	4	25
10	Influenza vaccination	1.27	2.40	20	5
11	Vaccinations	0.67	1.17	20	5
12	Driver's license certifications	0.55	4.43	20	5
13	Afternoon blood tests	0.41	2.40	20	5
14	Booking and cash desk	0.08	2.90	20	5
15	Outpatient consultations	0.03	2.40	20	5

The outpatient facility had 75 configurations in December 2019, all defined by its managers and following internal policies. One configuration served four services, five configurations served three services, and the remaining ones served either one or two services. Each configuration served at most one of the three high-demand services, i.e., those with a demand exceeding 10% of the total requests. In addition, there was a configuration serving two services and addressing 41% of the total requests. Some services had a low overall request rate, but they were associated with the highest levels of urgency, e.g., services 6 and 9 in Table 3.3. All in all, finding good-quality solutions to the problem is not straightforward for a decision-maker, even if the frequency of requested services is known.

After analyzing the historical data, we created and solved 61 realistic instances, considering up to 90 scenarios similarly generated. It is important to mention that instances do not share information; that is, when solving, e.g., instance α with SBPA, no information about its future patients is known in advance, and no scenario has information about its future patients. Besides that, all patients in each scenario are considered fictive patients and do not count in the final solution of the instance α. When deciding at time t, the algorithm considers only the fictive patients whose arrival time is in the interval $[t,\ t + \Delta)$.

3.5.2 Computational results

In the following discussion, we first study the influence of some parameters on the solution quality and computing time, namely: (i) the policy $\pi \in \{\text{wFIFO}, \text{wEDD}, \text{wLPT}\}$ used to schedule patients to servers; (ii) the size of the time step Δ between two subsequent optimizations; (iii) the number of scenarios $|\Xi|$ for SBPA and SBPRA; (iv) the consensus function ω; and (v) the use of the *Recombination Solution* step. As random choices may influence the computational results, we solve each instance ten times for each parameter combination, reporting the average, best, and worst results.

The first results are related to comparing the three policies $\pi \in \{\text{wFIFO}, \text{wEDD}, \text{wLPT}\}$ to schedule patients. Results are presented in Table 3.4 and consider $|\Xi| = 0$, i.e., they are obtained by running the plain RE-OPT algorithm. We compare the performance of the considered policies over three different re-optimization time step values: $\Delta \in \{60, 120, 180\}$, expressed in minutes. The performance indicators are based on the objective function values:

for each instance $h \in H$, we get the best, worst, and average solution values, i.e., $z_{h,\min}$, $z_{h,\max}$, and $\bar{z}_h$, respectively, given the ten runs. This is considered for each Δ and π combination. We also compute the average indicators considering all 61 instances, resulting in $z_{\min}$, $\bar{z}$, and $z_{\max}$ accordingly. Besides that, for each instance, we report how many times a policy obtained the best values in terms of $z_{h,\min}$, $\bar{z}_h$ and $z_{h,\max}$. We denote these values by $\textsc{Best}(z_{\min})$, $\textsc{Best}(\bar{z})$ and $\textsc{Best}(z_{\max})$, respectively. As a result, summing the corresponding values in the cells of the wEDD policy with the ones in the wFIFO and wLPT policies, in the absence of ties, gives 61 (i.e., the total number of instances). Formally, by using ρ to denote one of the $\rho_{\max} = 10$ runs, we have:

$$z_{h,\min} = \min_{\rho \in \{1,\ldots,\rho_{\max}\}} z_h^{(\rho)}(\Delta,\pi), \quad \bar{z}_h = \frac{1}{\rho_{\max}} \sum_{\rho=1}^{\rho_{\max}} z_h^{(\rho)}(\Delta,\pi), \quad z_{h,\max} = \max_{\rho \in \{1,\ldots,\rho_{\max}\}} z_h^{(\rho)}(\Delta,\pi)$$

$$z_{\min} = \frac{1}{|H|} \sum_{h \in H} z_{h,\min}, \quad \bar{z} = \frac{1}{|H|} \sum_{h \in H} \bar{z}_h, \quad z_{\max} = \frac{1}{|H|} \sum_{h \in H} z_{h,\max}$$

and

$$\begin{aligned}
\textsc{Best}(z_{\min}) &= \textsc{Best}(z_{\min}(\Delta,\pi)) \\
&= |\{h \in H:\ z_{h,\min}(\Delta,\pi) \le z_{h,\min}(\Delta,\pi'),\ \forall \pi' \in \{\text{wFIFO}, \text{wEDD}, \text{wLPT}\}\}| \\
\textsc{Best}(\bar{z}) &= \textsc{Best}(\bar{z}(\Delta,\pi)) \\
&= |\{h \in H:\ \bar{z}_h(\Delta,\pi) \le \bar{z}_h(\Delta,\pi'),\ \forall \pi' \in \{\text{wFIFO}, \text{wEDD}, \text{wLPT}\}\}| \\
\textsc{Best}(z_{\max}) &= \textsc{Best}(z_{\max}(\Delta,\pi)) \\
&= |\{h \in H:\ z_{h,\max}(\Delta,\pi) \le z_{h,\max}(\Delta,\pi'),\ \forall \pi' \in \{\text{wFIFO}, \text{wEDD}, \text{wLPT}\}\}|.
\end{aligned}$$

In other words, the value of $\bar{z}$ is the overall average objective function value for all 61 instances, all 10 runs for each instance, and all parameter configurations. This value gives an overall perspective of the algorithm's performance over different instances and parameter configurations. A small value corresponds to a superior overall algorithm performance compared to the others. The value of $z_{\min}$ corresponds to the overall average value among the minimum objective function values for all instances and parameter configurations, providing insights into the algorithm's average performance for the best solutions found over the ten runs. On the other hand, the value of $z_{\max}$ is the overall average value among the maximum objective function values for all instances and parameter configurations. It gives information about the overall algorithm performance, considering the worst solutions found over the ten runs.

In the results of Table 3.4, we observe that reducing the time step of the re-optimization parameter systematically affects the solution quality. On average, from $\Delta = 180$ to $\Delta = 120$ the value of $\bar{z}$ has a reduction superior to 20%. We notice a similar behavior when passing from $\Delta = 120$ to $\Delta = 60$. When comparing policies, it is evident from the last three columns that both wEDD and wFIFO policies significantly outperform the wLPT policy. For instance, when $\Delta = 180$, wEDD achieves the best solutions in 42 out of 61 instances, while wLPT achieves it in only 14 cases. This behaviour is also evident when considering $z_{\min}$, $\bar{z}$ and $z_{\max}$. The average improvement using either wEDD or wFIFO over wLPT is around 10% for $z_{\min}$ and around 18% for both $\bar{z}$ and $z_{\max}$.

In Table 3.4, we also note the occurrence of ties in the last three columns, primarily due to the comparable performance of wEDD and wFIFO policies. Indeed, when fixing Δ, the sum of the occurrences of the three policies often exceeds 61, for $\textsc{Best}(z_{\min})$, $\textsc{Best}(\bar{z})$ and $\textsc{Best}(z_{\max})$. Moreover, the results of wEDD and wFIFO policies are similar for $z_{\min}$, $\bar{z}$, and $z_{\max}$, with slightly better overall results obtained by wEDD. These findings show that both wFIFO and wEDD policies consistently produce better solutions than those produced by the

TABLE 3.4: Results of the plain RE-OPT algorithm for the three policies π.

Δ	π	$z_{\min}$	$\bar{z}$	$z_{\max}$	BEST $(z_{\min})$	BEST $(\bar{z})$	BEST $(z_{\max})$
60	wEDD	1579926	2178641	2962455	34	38	39
	wFIFO	1583503	2181730	2964863	35	45	39
	wLPT	1595821	2367925	3295606	24	11	20
120	wEDD	2101170	2706187	3428427	38	46	54
	wFIFO	2095694	2706766	3435523	44	51	54
	wLPT	2453185	3471583	4541212	17	2	3
180	wEDD	2992400	3677627	4547705	42	40	48
	wFIFO	2992304	3677449	4547626	44	44	48
	wLPT	3324875	4255414	5092104	14	10	10

wLPT policy. It is important to comment that all policies are relevant as they approximate different queuing situations, and deciding which one to apply is not straightforward in practical situations. Due to the nature and applicability of each policy, we decided to present results for each one of them also in the next tables.

The next results are related to the SBPA heuristic and the new version we propose, SBPRA. Tables 3.5–3.7 show the results obtained with $\Delta = 60, 120$ and 180, respectively. We aim to compare the gap, i.e., the relative difference between the value of the solution produced by an algorithm and a reference value. The gap values are not given in percentages. Each table contains results for different policies of RE-OPT, SBPA, and SBPRA, besides the various combinations of consensus functions and numbers of scenarios. For SBPA, the consensus functions ω are AVG, BEST, and WORST, while for SBPRA they are the consensus functions AVG-R, BEST-R, and WORST-R, as discussed in Section 3.4. The number $|\Xi|$ of scenarios we attempted is in $\{5, 10, 20, 30, 45, 60, 90\}$.

The values in the tables are calculated as follows: for each instance $h \in H$, we assume as reference value $z_{\min,h} = z_{\min,h}(\Delta) = \min_{\pi,\omega,|\Xi|,\rho} z_h^{(\rho)}(\Delta)$, i.e., the minimum among all runs considering all algorithms for the given time step Δ. Then, for each instance solved with each algorithm in each run, with any combination of policy, consensus, and number of scenarios, we calculate the relative gap $\delta z_h^{(\rho)}$ from the reference value. For each consensus function ω and number of scenarios $|\Xi|$, as well as for each instance, we determine the average gap $\overline{\delta z_h}$ over all runs. Finally, we calculate the overall average gap over all 61 instances, denoted by $\overline{\overline{\delta z}}$, where lower values indicate proximity to the reference value (i.e., a better performance). Formally:

$$\delta z_h^{(\rho)} = \delta z_h^{(\rho)}(\Delta, \pi, \omega, |\Xi|) = \frac{z_h^{(\rho)}(\Delta, \pi, \omega, |\Xi|) - z_{\min,h}(\Delta)}{z_{\min,h}(\Delta)}$$

$$\overline{\delta z_h} = \overline{\delta z_h}(\Delta, \pi, \omega, |\Xi|) = \frac{1}{\rho_{\max}} \sum_{\rho=1}^{\rho_{\max}} \delta z_h^{(\rho)}(\Delta, \pi, \omega, |\Xi|)$$

$$\overline{\overline{\delta z}} = \overline{\overline{\delta z}}(\Delta, \pi, \omega, |\Xi|) = \frac{1}{|H|} \sum_{h \in H} \overline{\delta z_h}(\Delta, \pi, \omega, |\Xi|).$$

In Table 3.5, both SBPA and SBPRA have better performance than RE-OPT as the number of scenarios $|\Xi|$ increases. The improvements are substantial for all policies. For example, considering $|\Xi| = 90$ and consensus BEST, the percentage reduction in the gap values compared to RE-OPT is about 70%, while it is more than 80% with consensus BEST-R. Similar findings can be noted for all the policies considered, confirming a significant improvement of both SBPA and SBPRA over RE-OPT. Besides that, we note that the newly proposed

TABLE 3.5: Comparison in terms of overall average gap ($\overline{\overline{\delta z}}$) from the reference value, for $\Delta = 60$.

π	ω	RE-OPT	number of scenarios: $\lvert\Xi\rvert$							*Average*
			5	10	20	30	45	60	90	
	-	1.53								1.53
	AVG		1.08	0.89	0.73	0.67	0.59	0.54	0.47	0.71
	BEST		1.09	0.91	0.76	0.70	0.61	0.56	0.50	0.73
wEDD	WORST		1.08	0.89	0.73	0.67	0.58	0.53	0.45	0.70
	AVG-R		0.68	0.52	0.44	0.40	0.36	0.34	0.30	0.43
	BEST-R		0.69	0.53	0.44	0.40	0.36	0.34	0.30	0.44
	WORST-R		0.69	0.53	0.44	0.40	0.36	0.35	0.30	0.44
	-	1.53								1.53
	AVG		1.10	0.90	0.75	0.68	0.60	0.54	0.47	0.72
	BEST		1.11	0.91	0.77	0.70	0.63	0.57	0.53	0.75
wFIFO	WORST		1.10	0.90	0.73	0.67	0.57	0.52	0.44	0.70
	AVG-R		0.68	0.53	0.45	0.40	0.36	0.34	0.29	0.44
	BEST-R		0.69	0.55	0.47	0.42	0.39	0.36	0.34	0.46
	WORST-R		0.70	0.54	0.45	0.41	0.37	0.34	0.30	0.44
	-	1.75								1.75
	AVG		1.09	0.89	0.75	0.66	0.57	0.51	0.45	0.70
	BEST		1.09	0.91	0.77	0.67	0.58	0.54	0.48	0.72
wLPT	WORST		1.08	0.89	0.75	0.66	0.57	0.52	0.46	0.70
	AVG-R		0.70	0.51	0.42	0.39	0.36	0.34	0.30	0.43
	BEST-R		0.70	0.52	0.43	0.40	0.36	0.34	0.30	0.44
	WORST-R		0.71	0.54	0.43	0.40	0.37	0.34	0.30	0.44

SBPRA always has the best performance, with an overall value of $\overline{\overline{\delta z}}$ that, considering all scenarios, consensus functions, and policies, is around 0.44 against 0.72 of SBPA, with a remarkable decrease of about 38%.

Table 3.6 presents the results for $\Delta = 120$. Similarly to $\Delta = 60$, SBPA and SBPRA have the best results as the number of scenarios increases, with SBPRA achieving the best overall performance. SBPA improves the results of RE-OPT for any combination of consensus function, number of scenarios, and policy, reaching an improvement of 66% when $|\Xi| = 90$. With SBPRA, the improvement is even more significant and amounts to 78%. Considering the three different policies, we observe similar results with wEDD and wFIFO for each combination of the number of scenarios and consensus function. In addition, both policies outperform the results of wLPT. For example, considering all SBPRA consensus functions, the value of $\overline{\overline{\delta z}}$ is around 0.50 for wEDD, while it increases to about 0.56 for wLPT.

Table 3.7 shows the results for $\Delta = 180$. Once again, the proposed SBPRA is superior to the other algorithms. Moreover, the results obtained with the wEDD and wFIFO policies are superior to those of the wLPT policy, with the value of $\overline{\overline{\delta z}}$ around 0.98 for wEDD and wFIFO, and 1.10 for wLPT, considering all SBPRA variants.

The results in Tables 3.5-3.7 reveal that the larger the number of scenarios is, the better the solution is for both SBPA and SBPRA. The improvement is particularly notable when $\Delta = 60$, as shown in Figure 3.3. It is worth mentioning that improvements in the solution quality are observed for all combinations of time step Δ, policy π, and number of scenarios $|\Xi|$. Overall, smaller values of Δ contribute to obtaining higher-quality solutions. Regarding the algorithms, introducing the *Recombination Solution* step in SBPA was fundamental to improving all solutions compared to the classical SBPA. A comparison of the consensus functions AVG-R, BEST-R and WORST-R over AVG, BEST and WORST, respectively, given all combinations of Δ, π, and $|\Xi|$, indicates an average improvement of 38%, ranging

TABLE 3.6: Comparison in terms of overall average gap ($\overline{\overline{\delta z}}$) from the reference value, for $\Delta = 120$.

π	ω	RE-OPT	number of scenarios: $\lvert\Xi\rvert$							*Average*
			5	10	20	30	45	60	90	
	-	2.08								2.08
	AVG		1.61	1.38	1.15	1.06	0.94	0.88	0.78	1.11
	BEST		1.61	1.41	1.17	1.10	0.96	0.92	0.80	1.14
wEDD	WORST		1.61	1.38	1.13	1.05	0.93	0.86	0.76	1.10
	AVG-R		0.97	0.85	0.72	0.65	0.59	0.56	0.50	0.69
	BEST-R		0.97	0.86	0.73	0.65	0.60	0.56	0.50	0.70
	WORST-R		0.98	0.86	0.73	0.65	0.59	0.56	0.50	0.70
	-	2.08								2.08
	AVG		1.62	1.39	1.14	1.07	0.96	0.89	0.77	1.12
	BEST		1.63	1.41	1.16	1.10	0.97	0.92	0.80	1.14
wFIFO	WORST		1.63	1.39	1.13	1.06	0.93	0.87	0.75	1.11
	AVG-R		0.97	0.85	0.73	0.66	0.60	0.56	0.50	0.70
	BEST-R		0.98	0.85	0.73	0.67	0.61	0.56	0.50	0.70
	WORST-R		0.98	0.86	0.73	0.66	0.60	0.56	0.50	0.70
	-	2.94								2.94
	AVG		1.72	1.48	1.25	1.13	1.01	0.94	0.82	1.19
	BEST		1.72	1.49	1.25	1.13	1.02	0.95	0.83	1.20
wLPT	WORST		1.73	1.49	1.26	1.16	1.03	0.96	0.83	1.21
	AVG-R		1.12	0.96	0.79	0.73	0.67	0.62	0.55	0.78
	BEST-R		1.13	0.95	0.79	0.73	0.67	0.62	0.56	0.78
	WORST-R		1.13	0.96	0.80	0.74	0.68	0.64	0.56	0.79

from a minimum of 32% to a maximum of 43%. This enhancement is evident when observing the graphs in Figure 3.3. Besides the superior performance of SBPRA compared to SBPA for the same value of $|\Xi|$, SBPRA still outperforms SBPA with the next value of $|\Xi|$ (i.e., comparing it with the next column of Tables 3.5-3.7). This confirms that adding the recombination solution is significantly more beneficial than only adding more scenarios to Ξ, which is a relevant achievement.

FIGURE 3.3: Values of $\overline{\overline{\delta z}}$ obtained with SBPA and SBPRA for the three time steps over all consensus and policies, summarizing the results in Tables 3.5-3.7.

A common issue for SBPA implementations may concern the computational effort required to attain satisfactory solutions [115]. This is often associated with implementing effective consensus functions and the number of used scenarios. In the proposed SBPRA, we also have an ILP model that is exactly solved (no time limit was imposed), which could make this issue even more evident. Table 3.8 provides an overview of the average computing time (in seconds) required to solve an instance, for the entire time horizon, with the proposed algorithms, assuming different combinations of time step Δ and number of scenarios $|\Xi|$. We decided to aggregate the computing times of the different consensus functions and policies for SBPA and SBPRA because they are quite similar. As expected, the computing times associated with each time step increase as the number of scenarios increases. However, the difference between the time required by SBPA and SBPRA is very small, less than 2 seconds on average, for all values of Δ and $|\Xi|$.

Observing Table 3.8, the computational times of RE-OPT are very low compared to those of SBPA having many scenarios (e.g., $|\Xi| = 90$). On the other hand, computing times get

TABLE 3.7: Comparison in terms of overall average gap ($\overline{\overline{\delta z}}$) from the reference value, for $\Delta = 180$.

π	ω	RE-OPT	number of scenarios: $\lvert\Xi\rvert$							*Average*
			5	10	20	30	45	60	90	
	-	2.65								2.65
	AVG		2.17	1.93	1.68	1.52	1.39	1.30	1.18	1.59
	BEST		2.17	1.93	1.68	1.54	1.40	1.33	1.21	1.61
wEDD	WORST		2.17	1.92	1.68	1.53	1.39	1.30	1.20	1.60
	AVG-R		1.30	1.14	1.03	0.95	0.89	0.83	0.78	0.99
	BEST-R		1.29	1.14	1.03	0.96	0.90	0.84	0.78	0.99
	WORST-R		1.30	1.14	1.03	0.95	0.90	0.84	0.78	0.99
	-	2.65								2.65
	AVG		2.17	1.93	1.64	1.49	1.37	1.28	1.15	1.58
	BEST		2.17	1.94	1.64	1.52	1.37	1.30	1.18	1.59
wFIFO	WORST		2.17	1.94	1.64	1.49	1.37	1.28	1.17	1.58
	AVG-R		1.25	1.13	1.01	0.93	0.88	0.83	0.76	0.97
	BEST-R		1.25	1.12	1.01	0.94	0.88	0.83	0.76	0.97
	WORST-R		1.25	1.13	1.01	0.93	0.87	0.83	0.76	0.97
	-	3.23								3.23
	AVG		2.40	2.12	1.81	1.68	1.58	1.50	1.41	1.79
	BEST		2.40	2.11	1.80	1.67	1.54	1.45	1.33	1.76
wLPT	WORST		2.40	2.12	1.83	1.71	1.59	1.52	1.45	1.80
	AVG-R		1.44	1.27	1.10	1.03	1.01	0.95	0.92	1.10
	BEST-R		1.44	1.26	1.09	1.00	0.96	0.90	0.84	1.07
	WORST-R		1.45	1.27	1.12	1.05	1.03	0.98	0.96	1.12

closer to each other when SBPA and SBPRA have only $|\Xi| = 5$. Notice that these two algorithms are superior to RE-OPT even with 5 scenarios. By increasing Δ, we notice that computing times reduce when $|\Xi| < 10$, since less re-optimization is needed. This does not happen for larger values of $|\Xi|$ because the scheduling of patients is more frequent. Since re-optimization is performed every Δ minutes, this implies a very short computing time for SBPA and SBPRA. In any case, we can conclude that the execution times of both SBPA and SBPRA are compatible with their use in practice (as they are all much shorter than the length of the re-optimization time step Δ), and that SBPRA does not require any significant increase in the computing time with respect to SBPA.

TABLE 3.8: Computing times (in seconds) of the proposed algorithms.

Δ	Heuristics	number of scenarios: $\lvert\Xi\rvert$								*Average*
		0	5	10	20	30	45	60	90	
	RE-OPT	1.55								
60	SBPA		2.52	5.04	10.75	17.16	26.78	38.25	64.68	23.60
	SBPRA		2.60	5.31	11.08	17.75	28.65	40.08	66.28	24.54
	RE-OPT	0.89								
120	SBPA		1.98	4.14	9.26	15.94	27.95	42.01	78.59	25.70
	SBPRA		2.02	4.21	9.80	16.75	28.13	41.90	78.13	25.85
	RE-OPT	0.60								
180	SBPA		1.85	4.23	10.98	19.40	36.91	58.87	116.59	35.55
	SBPRA		1.83	4.41	11.36	20.13	37.64	59.52	114.78	35.67
Average		1.01	2.13	4.56	10.54	17.86	31.01	46.77	86.51	28.48

3.6 Conclusions

Optimizing the flow of patients in outpatient facilities is a hard task. Patients with different needs and priorities wish to be serviced as soon as possible. For different reasons, they may leave the facility without receiving care, negatively impacting the facility's performance. In this study, we optimize the use of servers in an outpatient facility, handling a real-world dynamic and stochastic problem. The objective is to assign configurations (each comprising different services required by the patients) to the servers so as to reduce the number of abandoning patients and the total weighted tardiness. We solve the problem by means of an innovative algorithm that we call the Scenario-Based Planning and Recombination Approach (SBPRA). SBPRA generalizes the well-known Scenario-Based Planning Approach (SBPA) by including a dedicated mathematical model that combines the solutions found for different scenarios in order to find a new, possibly more balanced solution. Both algorithms make use of an inner Reduced Variable Neighborhood Search (RVNS) to decide the assignments of configurations to servers.

In the computational experiments performed on real-world instances, we evaluated how the re-optimization time interval to update the servers, the number of scenarios, the consensus function, and the policy to schedule patients impact the solution quality and the computing time. The results show that large values of the re-optimization time interval affect the solution quality. Thus, a decision-maker should pay attention and act as soon as she detects an increase in tardiness or patients abandoning the facility. The results also show that increasing the number of scenarios improves the solution. This conclusion is clearly observed when the re-optimization time step is set to 60 minutes. Concerning the scheduling of patients, it can be a wise decision to schedule them according to their weighted target time or weighted arrival time instead of the expected weighted processing time.

We observed interesting insights also when comparing the SBPRA against the SBPA and a plain re-optimization algorithm (called RE-OPT) that does not invoke scenarios. The proposed SBPRA is indeed quite efficient. Even with a few scenarios (e.g., 5), we observe satisfactory improvements over the SBPA and RE-OPT results. In particular, the SBPRA outperformed the SBPA by 38% on average on their best configuration, a quite impressive result. Moreover, its computing time is relatively small, especially considering a dynamic and stochastic problem solved over a large time horizon.

Setting the number of scenarios to 90 and the minimum re-optimization time step to 60 minutes proves to be the most efficient choice for the SBPRA, resulting in the best solution values, obtained with an average computing time of around 66 seconds. It is important to mention that using just 30 scenarios already produces very good solutions while reducing the computing times of the SBPRA by two-thirds on average.

After all, we observe that there is still room for future research. First, regarding solution methods, other (sophisticated) heuristics to handle patient scheduling could be used instead of simple policies to improve the solution quality. For example, the very elaborated branch-and-regret (see, e.g., Côté, Queiroz, Gallesi, and Iori [30]) could be implemented and compared with the proposed SBPRA. Another interesting direction could be to investigate the impact of other consensus functions and data-driven neighborhood searches.

Second, the distribution of abandonment times in outpatient facilities can be influenced by various factors, such as queue lengths, the patients' priority weight, and even the overall workload of the facility. Priority weights may increase while patients wait in the facility, indicating that priorities could change over time. Patients may abandon the facility sooner if they observe long waiting times or their priorities increase. Conversely, patients might wait longer in less crowded periods before abandoning the facility. All these indicate that abandonment times are due to the current conditions inside the facility. Understanding the interaction between these dynamic factors and their influence on patient abandonment patterns can provide

valuable insights to facility managers. All information about the facility, including queue lengths, patient waiting times, and priority weights, could be integrated into the optimization process. Thus, decision-makers at the facility could update patient scheduling on the fly to achieve the best overall practice.

Third, introducing new constraints that better model real-world applications is also an important future research direction. This could be the case for time-budget constraints, which involve a maximum amount of time a server or a specific configuration can remain active. This constraint reflects cases in which the available resources must satisfy specific time windows. Another direction could be to investigate conflicting KPIs (e.g., patients' needs and facility costs), so handling a multi-objective objective problem by proposing multi-objective RVNS-based algorithms (as in, e.g., [89]).

Chapter 4

A storage location assignment problem with incompatibility and isolation constraints

In this chapter, we explore a variant of the Storage Location Assignment Problem (SLAP) that incorporates product-cell incompatibility and isolation constraints (SLAP-PCIIC), as found in pharmaceutical warehouses. The objective is to minimize picker travel distances while respecting stringent operational and safety constraints. We propose an Iterated Local Search algorithm, which effectively addresses the problem, as demonstrated through computational experiments on diverse instances. This research is presented also in [73, 74].

4.1 Introduction

Pharmaceutical or equipment shortages in healthcare services strongly influence healthcare services, are often the cause of interruptions and delays in attendance, increasing the chance of putting patients' lives at risk. Constant item replenishment and the use of high inventory levels constitute the traditional approach to avoid this problem [113] [4]. However, this solution has been often considered expensive and difficult to manage, as it requires large dedicated spaces in facilities and the workload of healthcare personnel [117].

Recently, more efficient strategies have been adopted, as acquisitions through *Group Purchasing Organisations* (GPO) and wholesalers. Centralised warehouses sharing to store together products to be distributed to different customers located in the same geographical area are one of the most successful approaches. By allowing healthcare facilities to share a common structure to store a large volume of products, centralised warehouses are particularly useful. Also, they fasten the delivery process when they are needed. This enables a constant material flow, a reduction in the personnel costs, a reduced storage space in the customer facility and a lower work burden over healthcare workers. These benefits strongly depend on the warehouse reliability and capability of delivering the ordered products in the short terms defined by the customers. In turn, this reliability directly requires an efficient warehouse internal organisation, which is a result of a good *storage location policy*.

A storage location policy is a general strategy to assign *Stock Keeping Units* (SKU) to storage positions inside a warehouse. It aims at optimising a metric (e.g. total time or distance travelled to store and retrieve SKUs, congestion, space utilisation, pickers ergonomic), while considering issues like product re-allocations efforts, demand oscillation, picking precedence and storage restrictions.

Commonly, the metric adopted to compare these strategies is the distance travelled by the pickers to retrieve all products in a list of orders. The relevance of this metric is due to the fact that picking operations accounts for around 35% of the total warehouse operational costs [120] and the energy/time spent to reach a product location is a waste of resources

that must be minimised. In other frameworks, the evaluation may also consider issues like congestion, picker ergonomic, product storage conditions and total space utilisation, that can also lower the warehouse operation efficiency. In this study, we address a problem originating from real life operations of a pharmaceutical products distributor. As a main goal, the optimisation of a dedicated storage allocation policy in a picker-to-parts warehouse, i.e., a warehouse where each product is assumed to have a dedicated/fixed position and any time an order request is received, pickers move to reach product location to retrieve order items. In more detail, we deal with a *Storage Location Assignment Problem with Product-Cell Incompatibility and Isolation Constraints* (SLAP-PCIIC). In this problem, some products cannot be assigned to certain locations due to reasons like ventilation or refrigeration (*product-cell incompatibility*), and some other products need to be isolated from the unconstrained ones due to contamination or toxicity concerns (*isolation*).

In our *Iterated Local Search* (ILS) algorithm, a set of orders and a warehouse layout are taken in input and a list of assignments of products to locations that minimises the total distance travelled to fulfill the orders is returned. Our main contribution relies on extensive instance tests of the ILS algorithm proposed in [73] with larger instances, an enriched and updated literature review and a deeper exposition of input warehouse data processing. Inferring the distance matrix from warehouse layout input data, though necessary for most of SLAP algorithms initialization, is not an easy task: in order to improve algorithm usability, we dedicated Section 4.4 to a detailed description of this process. Also, Figures 4.1 and 4.2 are integrated for easier readability and faster comprehension.

The remainder of the paper is organised as follows: in Section 4.2, an extended literature review is presented; Section 4.3 provides a detailed problem description; Section 4.4 presents all the data processing done before the optimisation starts; Section 4.6 describes the various instance sets used to test the ILS algorithm; Section 4.5 presents the ILS algorithm; Section 4.7 describes the numeric experiments carried out, results are provided and then the conclusions are drawn in Section 4.8.

4.2 Literature review

The *Storage Location Assignment Problem* (SLAP) is a generalisation of the well know Assignment Problem in which a set of elements must be assigned to a specif position inside a storage area. It is also related with the Quadratic Assignment Problem (QAP), mainly due to the methods used to evaluate the solutions [102]. Most of times, SLAP variants have complex constraints and objective functions that includes considerations about warehouse layout, picking policy, picker routing policy and order batching [35], that together can make the solution evaluation slow or imprecise.

The SLAP constraints are mostly related with strategic or tactical decisions taken by warehouse managers, as warehouse dimensions and layout, shelves capacities, storage policy, facility function, etc [110]. By their own nature, these decisions are hard to be changed, disregarded or relaxed, due to the costs, training or setup times involved.

The most important constraint in this sense is the warehouse layout. It basically defines the number and position of available storage locations, as well the physical barriers that guide or reduce picker's mobility. It also implicitly defines if the picker will need machines to recover products allocated on higher places, tight corridors [27] or with mobile racks [42], causing a increase on picking time or even internal congestion.

Warehouse layouts are usually classified according two main characteristics: the number of blocks (single block or multi block) and the presence (or absence) of stacked/high level storage. In a warehouse organized in blocks, each block can be defined as a set of identically long, parallel and aligned shelves separated by aisles (i.e. corridors). Each aisle traverses

a pair of shelves along its whole extension without being crossed by any other aisle (called in this case a cross-aisle). Stacked storage (or floor stacking storage), on its turn, is the configuration in which the allocation of products can be organized in stacks with one product directly put over another or in shelves vertically divided [83]. In low level warehouses all the shelves are only low headed and the items can be picked by a walking picker. Consequently, if some items locations are not reachable for a picker without the help of an elevator/stair the warehouse is classified as a high level storage warehouse [129] [31].

The definition of warehouse layout is followed by the decision about the storage policy, that can be divided in three main groups: *random storage*, *dedicated storage* and *class based storage* [120][133]. A random storage policy allocates products in empty positions inside the warehouse using a random criteria (e.g. closest open location), without including any evaluation in the decision process. Dedicated storage goes in the opposite sense, by ranking the products according to some criteria - popularity, turnover, *Cube per Order Index* (COI) - putting the best ranked products close to the accumulation/expedition locations. Finally, a class based storage separates products in groups according to some popularity criteria and tries to optimize the position and size of each class [103].

Several studies report that random storage policies lead to a better space utilisation, due to frequent reuse of storage positions, but increase the travelled distances to pick the products [81] and require higher searching times or control over product locations [91]. Dedicated storage, on its turn, reduces the picking distance, but cause the rise with re-allocations costs (because of demand fluctuations) and space utilisation, as empty spaces can be reserved to products not currently available. Class based policies are a balance between random and dedicated storage strategies, but they require more strategic efforts to define the number of groups and their positions on the warehouse.

Random storage is rarely studied in the literature, being more an operational and practical approach used as a benchmark to evaluate other methods [85]. One noticeable exception is [91], which proposes a heuristic to dynamically allocate pallets in a storage area considering stacking constraints in order to reduce the total area used. Pallet stacking is also discussed in [83], which proposes a bi-objective mathematical model and a constructive algorithm.

A dedicated storage policy is considered in [35], in which exact distance evaluations are used to define the product assignment. [49] proposes a non-linear model and an ILS to address a storage allocation problem in a multi-level warehouse considering the compatibility between product classes. In [120], by departing from an S-shape routing policy and multi-level storage a two-phase algorithm is created to assign items to locations, and a multi-criteria approximation is used to evaluate the solutions. Other notable works regarding dedicated to storage are [9], which propose a storage assignment and travel distance estimation to design and evaluate a manual picking system and [14], which propose a model to describe a storage assignment problem, solving it to proven optimality for small instances.

Class based policies are studied in [93], [81] and [80]. In [93], class boundaries are defined based on the picking travel distance in a two-block and low-level warehouse where returning routing policy is used. The second uses a Simulated Annealing algorithm to define classes and assign locations to them inside a warehouse considering simultaneously space and picking costs. The work in [80] extends [81] by using a branch-and-bound algorithm instead of a heuristic, and by including space use reduction in the considered metrics.

In [87], the authors propose a data-mining based algorithm that uses association rules to define product dedicated positions in order to minimise the travelled distance in a picker-to-parts warehouse. Similar approaches are presented in [128] for optimizing storage with correlations related with *bill of materials*(BOM) picking, [75] for class based policy, [29] for cluster assignment, [41] for product classification and storage, and [58] for frequent item set grouping and slot allocation. In [66], an ABC classification and product affinity method is

used to define the best positions for the products. A set partitioning approach is presented in [60] to allocate entire areas, instead of specific locations, to group of products.

Some warehouses can also adopt some kind of zoning storage, in which each zone contains only a subset of the items stored (defined by any criteria) and each picker only retrieve and transport products stored in a single zone. This approach leads to reduced travel distances, but requires an additional work on order consolidation to put together products picked in the different zones. A more recent approach, more suited to large e-commerce retail warehouses, is the mixed-shelve storage, in which one product can be stored in several locations to be easily retrieved from any region of the warehouse, even if it requires large use of automation to find each item [129] [94] [22].

Another high level decision that affects the storage allocation strategy is the the picking system. The two main systems are pickers-to-parts and parts-to-pickers. In the former, pickers depart from a consolidation/expedition point and perform a tour to pick each product in their location, manually or by using some vehicle. In the latter, an automated storage and retrieve system, composed by one or more automated guided vehicles, picks ordered items and delivers them in the place where they will be prepared to be dispatched [9]. Pickers-to-part systems usually are easier and cheaper to implement and are more robust to changes, as most of times it is operated by humans, that adapt better to new situations than machines. Parts-to-picker however make the picking activity faster and more regular along long working shifts.

Picker-to-parts systems are addressed in [86], [84] and [85]. The first models the warehouse operation as a Markov chain in order to calculate the expected distance travelled by the picker in three different zoning cases. The second propose a heuristic for a case where congestion effects are considered. The last use a genetic algorithm to define the best workload balance among the pickers.

Parts-to-pickers are the subject of a large set of works. In one of the most recent studies [76] describe an integrated cluster allocation (ICA) policy, that considers both affinity and correlation in order to minimize retrieval time. This approach is extended in [77], who include the concept of inventory dispersion and propose mathematical models to define product positions in a warehouse with robot based order picking. In [57], new routing algorithms are proposed based on autonomous vehicles communication capabilities provided by new *Internet of Things (IoF)* technologies;

We can also highlight the existence of the pick-and-pass system (also called progressive zoning system [85]), in which each picker is responsible to retrieve a specific subset of products in an order and then deliver the incomplete order to the next picker, until all the products to be put together and dispatched. This system is commonly associated with a zone storage policy.

Among the metrics commonly used to evaluate assignment quality we can cite: shipping time, equipment downtime, on time delivery, delivery accuracy, product damage, storage cost, labour costs, throughput, turnover and picking productivity [104][96] [132]. By a large margin, however, the most commons are picking travel time and travel distance [96]. Their popularity is directly connected to an easier evaluation, representation and visualization. As disadvantage, we can mention the need of solving one or more instances of the *Travelling Salesman Problem* (TSP) or *Vehicle Routing Problem (VRP)* - depending on the presence of picker capacity constraints or multiple simultaneous picking tours. This additional optimization problem can let the overall process slow, depending of the number of assignments tested and the methods used to solve the TSP/VRP. In this sense, it is worth to mention that the travel distance metric has an additional advantage of allowing disregarding issues related with congestion, handling time, picker speed or searching delays.

In this context, several methods to optimise order picking routes have been proposed,

both inside SLAP variants studies or in independent researches. As pointed out in [35], routing problems in warehouses can be seen as special case of the Steiner Travelling Salesman Problem, that in some layouts can be solved to optimality ([95], [71] and [101], [24]) but in general layouts is mostly solved using heuristics ([32], [27], [97], [109]). For difficult layout warehouse configurations, exact algorithms for optimize picking route do not exist because the dynamic programming approaches used on single-block warehouses are not easy to be generalized for two or more cross-aisles [20]. The few exact methods available are used to solve problems with already defined warehouse allocations (as can be seen also in [48] and [96]) and they are mostly algorithms based on a graph theoretic algorithm for single-block warehouses [71].

Nonetheless, it is known that in real life operations pickers tend to deviate from optimal or non-intuitive routes [39]. To simulate this behaviour, many studies consider simple heuristics for picker routing as return point method, largest gap method, returning point or S-shape routing [31][39] [58]. These heuristics also simplify order picking evaluation as they quickly allow the computation of travelling distances in deterministic problems or the evaluation expected travelling distances in stochastic problems [35]. Some operational constraints can also make pickers deviate from optimal routes. It is the case in [111], which describes a warehouse by imposing a weight precedence rule during the picking, so that the heaviest products are retrieved before lighter ones. Congestion avoidance is either a factor that impacts on route definition. For the interested reader, an extensive analysis of these routing algorithms is presented in [31].

Another commonly used technique to deal with the optimisation of order picking routes is the order batching. It consists in performing the picking of one or more small orders in a single route, with a posterior products separation on the consolidation/expedition area.

More complex problems are surveyed in [45]. In [114], batching, routing and zoning are combined to optimise the warehouse operation. A discussion on storage allocation with product picking precedence can be found in [133], whereas [108] defines a problem where allocation and routing must be decided jointly in order to avoid congestion during the picking.

The SLAP-PCIIC, discussed in these paper is a storage allocation problem in which products have their assignment locations limited by their characteristics and mutual compatibility. Studies that deal with joint optimization of storage location assignment and picking routes, as we propose here, are less common in literature than those ones considering each problem separately. We can cite [102] and [14], who propose *Mixed Integer Programming* (MIP) model to solve their problems.

The SLAP-PCIIC also considers high level storage, non regular blocks, and warehouses divided in one or more interconnected pavilions. Furthermore, instead of performing an indirect evaluation of the order picking travel distance, we directly optimize order picking routes during the location assignment optimization. Constraints similar to the ones discussed in this study can be found in [3] and [12]. The first one, the storage of temperature-sensitive products is discussed, mainly food and pharmaceutical ones, and a dynamic policy to allocate products is proposed, by observing the different temperatures inside the warehouse, as well seasonal variations, aiming to minimise the picking traveled distance and maximise the product safety. The second propose a mathematical model to define a warehouse layout to store hazardous chemicals products. Compatibility constraints are addressed in [49], who proposes an *Iterated Local Search* (ILS) algorithm to optimize product allocation in a multi-layer warehouse.

4.3 Problem description

The SLAP can be briefly described as follows: given a set P of products to be stored, a tuple $\omega = (O_1, \ldots, O_n)$ of (non-necessarily distinct) orders, in which an order $O_i \subseteq P$ is the subset of products to be picked up by a picker on a single route, and a set L of locations in a warehouse, define an assignment (that is, an injective function) $g : P \to L$ in which an evaluation function $z = v(g, \omega)$, to be described next, is minimised. The warehouse is received in input in the form of a graph, and the travel distance d_{ij} between any pair of locations $i, j \in L$ is calculated by invoking the Dijkstra algorithm. In other words, given the evaluation function v that maps a value to each ordered pair consisting of a set of orders and an assignment of products to locations, define the assignment that minimizes v.

In the SLAP-PCIIC, the SLAP variant described here, due to incompatibility and isolation constraints, g is, on the one hand, relaxed to a possible partial assignment, but on the other hand it is subjected to the constraints of the location-product eligibility, i.e., each product is assigned to at most a single location and each location receives at most a single product while respecting the incompatibility and (strong) isolation constraints. In this framework, $v(g, \omega)$ is defined as the sum of minimum travelled distance to pick all the products in each order (designated by $D(g, \omega)$), plus the non negative penalties for non desirable or missing allocations, (designated by $\Phi(g)$). Notice that if all products are allocated, then we have no contribution to the penalty Φ caused by missing allocations, meaning that the lack of product assignment to locations is highly deprecated. Following the company's operational rules, it is assumed that the warehouse uses a picker-to-parts picking policy (the picker visits the locations of the products) and orders splitting / folding are not allowed, making each order an individual and independent route. These assumptions make it possible to decompose the distance $D(g, \omega)$ as the sum of the minimal travelling distances to pick the products in each order O in the ω tuple, designated by $d(g, O)$. With these considerations, and indicating with $\mathcal{G}$ the set of relaxed admissible assignments $g : P \to L$, the SLAP-PCIIC objective function can be described as:

$$z = \min_{g \in \mathcal{G}} \sum_{i=1}^{n} d(g, O_i) + \Phi(g) \tag{4.1}$$

It can be noticed that to evaluate each one of the $d(g, O_i)$ it is necessary to solve another optimisation problem, more specifically a variant of the TSP that calls for the minimization of the travelled distance. Namely, if there is a single accumulation/expedition point, each product is assigned to (at most) a unique location and $O' = \{p_1, ..., p_{|O'|}\} \subseteq O \subseteq P$ is the requested order deprived of those products lacking of location, then $d(g, O)$ is the minimum distance to depart from the accumulation/expedition point, visit all the locations $(g(p_1), ..., g(p_{|O'|}))$ in the best possible sequence and then come back. Conversely, in the SLAP-PCIIC we allow the presence of more than one accumulation/expedition point, so the picker can depart from any of these points and return to another if this operation reduces the total distance travelled $D(g, \omega)$. The algorithms can be easily adapted to deal with the case in which the expedition points are, instead, fixed.

Defining an optimal picker routing in the scenario above is relatively simple if the warehouse is organised in blocks of identical and parallel shelves. However, in the SLAP-PCIIC, the shelves can have different sizes, cell quantities, orientations and positioning and also be located in different pavilions. To deal with this setting, a regular distance matrix containing the distances between each pair of locations is considered as the input of the distance minimisation method, disregarding any further information about the warehouse organisation.

The second part of the objective function, the penalty value $\Phi(g)$, is the sum of two terms: the number of unassigned products $\Phi_1(g)$ and the number of undesired allocations $\Phi_2(g)$. In this sense, the possible configurations of the function g are limited by a set F of

assignment incompatibilities and a set I of isolation constraints. Each assignment incompatibility $f \in F$ is a hard constraint (i.e., it must be strictly respected) composed by a tuple of three values (p,n,c) representing a product p, the nature n of the incompatible location (cell, shelf or pavilion) and the code c of the incompatible location, respectively. For instance, the incompatibility $(p_1, \text{“}shelf\text{”}, k_1)$ defines that product p_1 cannot be allocated on shelf k_1.

An isolation constraint is based on the product classification. Given a set T of types, representing the most relevant product characteristic to the storage (toxic, radioactive, humid, etc...), an isolation constraint $\imath \in I$ is a tuple of three values (t,n,s) specifying that products of type $t \in T$ should be allocated in an isolated $n \in \{\text{“}cell\text{”}, \text{“}shelf\text{”}, \text{“}pavilion\text{”}\}$ with an enforcement $s \in \{\text{“}weak\text{”}, \text{“}strong\text{”}\}$. The enforcement s defines if the isolation is a hard constraint or can be relaxed with a penalty.

4.4 Input data processing

A common assumption in studies about SLAP is the regularity of the warehouse layout. Most of the times, the warehouse is represented by sets of parallel and identical shelves that can be accessed through aisles between them and cross-aisles that allow moving from one aisle to another. These sets are called blocks, and most of the literature about SLAP or picker routing considers the presence of one or more blocks in the warehouse.

However, for several reasons, this assumption creates problems when a proposed algorithm needs to be released to production environment. In many facilities, for example, physical or operational barriers are present (like build columns and machines), so layouts cannot be represented as simple blocks.

In this section, we describe the format we created to represent general warehouse layouts, aiming at allowing the use of our algorithm to a large number of warehouses. Furthermore, we present the data processing performed to get the distance matrix of all the relevant positions in the warehouse, and thus to make it possible to use TSP algorithms to evaluate the total distance travelled by pickers to retrieve all the products in a set of orders.

4.4.1 Warehouse input format

The main objective in proposing a new format to describe the warehouse layout is to allow a quick description of different types of layouts. More specifically, a good format should provide: *(a)* easiness of transcription in a spreadsheet or text file; *(b)* readability; *(c)* direct representation on relational databases; *(d)* a simple and robust representation of internal valid paths; *(e)* possibility of defining pavilions and connections between them; *(f)* possibility of defining multiple accumulation/expedition points; *(g)* possibility of defining heterogeneous shelves and cells.

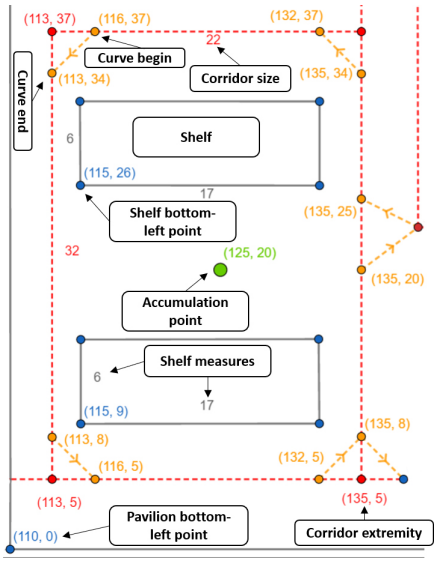
We defined points *(a)*, *(b)* and *(c)* aiming at a future implementation in a decision support system and also at an easy utilisation of the system, as most users are used to text files or spreadsheets and most developers are comfortable in using relational databases.

The robustness to represent paths (point (d)) was considered due to the existence of several operational rules that limit the traffic inside warehouses, mainly when it involves large vehicles or robots. In the proposed format, it is possible to define rectilinear corridors (with a start position, length, direction and sense) and rectilinear segments (with start and end position) to connect two corridors (see Figure 4.1). Non rectilinear corridors or connections were left out due to the variable number of points needed to define them and also because of the difficult evaluation of their length. We also defined that corridors must be parallel to one of the Cartesian axes, not allowing in this way oblique corridors.

Points *(e)* and *(f)* were adopted to take into consideration larger warehouses, in which internal divisions are common and operations can be less centralised. Notwithstanding, the

pavilions must be always represented as rectangles, as allowing other formats would significantly increase the representation complexity.

FIGURE 4.1: Partial warehouse representation with main elements information. The dashed lines are corridors, and the dashed arrows are curves



Point *(g)* is modelled to allow describing warehouses with a very diversified set of stored items, from large machines that are positioned on pallets to small tools that can be stored in cabinet drawers.

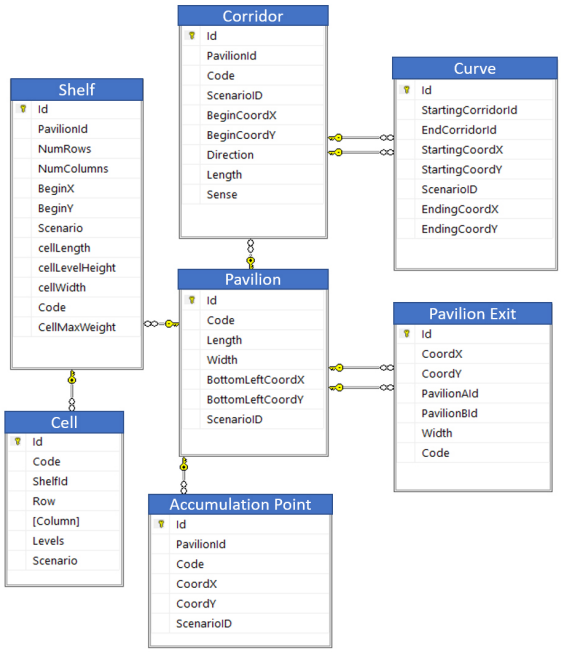
The resulting format is a union of seven tables (Pavilions, Shelves, Cells, Corridors, Curves and Pavilion Exits, Accumulation/Expedition Points), as presented in Figure 4.2. Each pavilion is composed of a code, a point (with coordinates x and y over a Cartesian plane) representing the bottom-left extremity, width and length (the height is not important to our problem, but can be easily included if needed). By its turn, each shelf has either a code, a bottom-left point, the number of rows and columns, the size of the cells (width, length and height) and the block code indicating where it is located. Each cell has a code, width, length, row and column at the shelf, a reference to the shelf and a number of vertical levels (1 or more). Corridors have a code, an initial point, a direction (horizontal or vertical), a sense (up-down, bottom-up, left-to-right, right-to-left, both), a length and a reference to the block where it is located. Each curve has a code, a reference to both corridors it connects, an initial point and a final point (in this case the length is calculated by the application). Each block exit contains a point, a width, a code and a reference to the blocks it connects. Finally, accumulation/expedition points are composed of a code and a bi-dimensional coordinate.

Products and orders use similar structures, but with less data. Each product is represented by a tuple containing code, description and type (size and weight are not considered in this problem) and each order with a tuple containing a code and a deadline. A list of items is assigned to each order, where each item contains a product code and a quantity.

4.4.2 Distance matrix extraction

Once all input information has been loaded, it is necessary to process it to get useful information for the algorithm. Basically, from a warehouse layout description we must extract a distance matrix containing the distances from each delivery point or storage location to all the others, and a compact representation of cells, shelves and blocks to check if the allocation prohibition and isolation constraints are being respected. The representation of warehouse

FIGURE 4.2: Warehouse database



structures (i.e. cells, shelves and blocks) was done simply using standard object-oriented classes, with no relevant improvements, so it is not detailed in the text.

To build the distance matrix we follow three steps: *(1)* transform the warehouse layout in a directed graph; *(2)* separate the vertices that represent storage locations and accumulation/expedition points from those that represent the paths in the warehouse; and *(3)* run iteratively a shortest path algorithm to determine distances between vertices.

The first step, the conversion of the warehouse in a graph, starts by creating vertices to represent accumulation/expedition points and inferior levels of the cells (i.e. those located closer to the ground, that are the connection points between shelves and corridors). The vertices are positioned in the central point of the cell level. After that, vertices to represent high levels in cells are created, always in the level central point. In detail, when there is more than one level in a cell, each vertex representing a cell level is connected by two arcs, one in each sense, to the level immediately above and below. In other words, when a cell is divided in five vertical levels, the third level is connected with the fourth and the second levels, but not with the first or fifth levels. The edge length correspond to the vertical distance between the level centres, that is the height of the cell divided by the number of cells (i.e. the height of a single level). In the instances of our study, the level height is fixed in 1.5 units.

It is important to notice that in this approach vertices representing external adjacent cells in a same shelf are not directly connected by an edge, except in some special cases that are described later. This is based on the fact that, in many cases, a picker needs to do a non-negligible backward movement to go from a cell to another. The length of this backward movement, however, is as small as the distance from the closest corridor path to the shelf, so it depends on the input and can be adjusted by the user.

Once connections between the vertices in the shelves have been completed, the process of connecting these shelves to the corridors begins. We consider that each rectangular shelf can be accessed only by its longest sides (except for squared shelves, that can be accessed by any side). In case of a vertical shelf (i.e. a shelf with the longer side parallel to the y-axis), we consider that a path may reach the shelf through its lateral (left/right) extremities, whereas for a horizontal shelf we consider bottom/up extremities. When the shelf is squared, four

corridors are selected, following the same logic. A corridor can not be selected if it does not go through all the shelf side or if it is located in a different pavilion. If there is no corridor in one of the sides, all the inferior cell levels in that side are connected to the adjacent cell levels in the opposite side, to become accessible from the remaining corridor. This is one of the special cases mentioned in the previous paragraph. If there are no corridors in the laterals of the shelf, this shelf is considered unreachable.

After the adjacent corridors be defined, each inferior cell level vertex is connected by a pair of edges (one in each sense) to a vertex created over the corridor exactly in front of the cell vertex. The new vertices created on corridors in the previous step are then merged with the vertices that represent the extremities of the curves (that are always over corridors) to define all the valid paths in the warehouse. Two consecutive vertices on a corridor are connected by two edges in a two-way corridor (designated by the word "both" in the field "sense" on corridor input data) and one edge otherwise. To avoid confusions, we defined that there is not a curve on corridor interception points, thus it is not right to suppose that it is possible to move from a corridor to another through these points unless a curve passing on it is defined.

Following the procedure, each vertex representing an accumulation/expedition point is connected to the closest non-storage vertex in the pavilion in which it is located by two edges, one in each sense. This connection can be a traversal one if there is no possibility of establishing a horizontal or vertical one.

To connect two different pavilions, we first create a vertex to represent each pavilion exit and then connect it to the closest vertex in each pavilion. The connection with pavilions vertices are done in both senses, with the same rules adopted to connect the accumulation/expedition points described in the previous paragraph.

After all these steps, we obtain a graph that represents the connections between all the relevant points in the warehouse. During the graph building process, all vertices are associated with the location they represent, so it is possible to associate each edge with the distance between its incident vertices in the warehouse. Furthermore, each vertex receives a label that is used to define if it is a storage vertex or a vertex representing an accumulation/expedition point, thus being relevant in the distance matrix. The graph built is then passed to an algorithm that calculates the distance matrix.

The algorithm that calculates the distance matrix is a loop, where at each iteration a Dijkstra shortest path algorithm is executed departing from one storage or accumulation/expedition vertex. In this way, all the graph vertices are considered in the shortest path evaluation.

4.5 Iterated Local Search

In this section, we present the ILS algorithm that we developed to solve the SLAP-PCIIC. First, a constructive algorithm is invoked by ILS to generate an initial solution, and then three different neighborhood structures are explored to attempt improving the solution.

Two main phases constitute the constructive algorithm (see Algorithm 6). In the first phase (described in lines from 6 to 23), locations are assigned only to the products without isolation constraints (i.e., products that do not belong to any tuple $\iota \in I$) or to those with isolation constraints involving an enforcement $s =$ "*weak*". In the second phase (described in lines from 25 to 40), it assigns locations to isolated products, according to their types. This procedure, by helping controlling $\Phi(g)$ value, promotes the allocation of a greater number of products, since the constrained products are usually fewer and could lead to a large number of unusable locations.

Request frequency is the first product sorting criterion used by the algorithm. Initially, all the products are sorted by descending order of popularity, i.e., the products more frequently

required are put first, and those less frequently after, where popularity is inferred from the tuple of orders ω. In a similar way, the storage location sorting criterion is the distance from the closest accumulation/expedition point (from lowest to highest). Then, the most popular product is assigned to the closest empty location, whenever this assignment is allowed. Instead, if an assignment is forbidden, the algorithm explores iteratively the next location, until an allowed position is found or the loop reaches the last position. If a product belongs to a strongly isolated type, it is not assigned during this phase.

In the second phase of the constructive algorithm, all the products presenting strong isolation constraints are divided by type (as shown in line 25 in Algorithm 6) and, similarly, the isolated warehouse locations are grouped by level (block, shelf or cell), in line 26. At this point, the allocation is done according to the structure size (from largest to smaller), starting from the isolated blocks and finishing with the isolated cells. At each step, the product types that have the corresponding isolation level are selected while preserving the frequency order. Then, each type is assigned to the isolated area where it is possible to maximise the total frequency, without worrying with the internal assignment optimisation. The list of available positions and products is updated at each step. The algorithm ends either when all the available isolated spaces are occupied or all the products are assigned.

Algorithm 6: `Greedy algorithm`

```
1  g ← ∅// Start with an empty assignment
2  L* ← distanceSort(L)// Locations are ordered by distance
3  A ← L*// Available locations
4  P ← sortByFrequence(P);
   // First part of greedy algorithm
5  foreach p ∈ P do
6      if isStronglyIsolated(p) then
7          continue;
8      l ← firstAvailable(A);
9      while true do
10         if l == null then
11             break;
12         if isForbidden(l, p) then
13             l ← nextAvailable(A);
14             continue;
15         g ← g ∪ (p, l);
16         A ← A \ {l};
17         break;
   // Second part of greedy algorithm
18 Λ ← stronglyIsolatedProductsByType(P);
19 Ψ ← availableIsolatedStructures(A);
20 μ ← allocateOnIsolatedBlock(Y, Ψ)// First allocation:  assigns products isolated by block
21 g ← g ∪ μ;
22 Ψ ← updateAvailableIsolateStructures(Ψ, μ);
23 Λ ← updateStronglyIsolatedProductsByType(μ, P);
   // Second allocation:  assigns products isolated by shelf
24 μ ← allocateOnIsolatedShelf(Y, Ψ);
25 g ← g ∪ μ;
26 Ψ ← updateAvailableIsolateStructures(Ψ, μ);
27 Λ ← updateStronglyIsolatedProductsByType(μ, P);
   // Third allocation:  assigns products isolated by cells
28 μ ← allocateOnIsolatedCell(Y, Ψ);
29 g ← g ∪ μ;
30 Ψ ← updateAvailableIsolateStructures(Ψ, μ);
```

It is significant to notice that, due to incompatibility and isolation constraints, it may be impossible to assign all the products to the warehouse locations. This may happen even when the number of available locations is higher than the number of products. Moreover, as a heuristic method, the constructive algorithm may be not able to find an initial feasible

assignment even when it exists. In both cases, the solution evaluation procedure penalizes the objective function according to the number of unassigned products (i.e., the value of $\Phi_1(g)$).

However, a valid solution can still report one of the side effects of allocating groups of isolated products together in the warehouse. The first is assigning relatively good positions to several products with a low number of requests due to the existence of some very popular products in the set, that are responsible of a skewed popularity of that group. The second effect is approximately the opposite, i.e., due to a low group average popularity, very popular products can be allocated in bad positions. Both these problems are analogous to those that are reported in class or zone based storage location problems [93] [81].

According to the procedure described in Section 4.5.1, the objective function of an initial solution is calculated, immediately after being created by the greedy algorithm. Then, the ILS heuristic enters in a loop devoted to the exploration of the solution space (see lines 4 to 25 of Algorithm 7). This loop is composed by three neighbourhood structures that are combined as a single local search, and a perturbation method.

Within a loop iteration, a small set of neighbours of the current solution (denoted by g) is evaluated by each neighbourhood structure and the one with lowest objective function value is stored, thus using a best improvement criteria. Then, denoting by g^w the best solution in the loop, and by g^* the best global solution, they are compared, and every time g^w is better, it is assigned to g^* and the number of iterations without improvement (line 22) is reset. Finally, a perturbation of g^* is performed at the end of the iteration. The loop stops if the number of iterations without improvement reaches the value of the input parameter *iterations without improvements* (IWI).

In order to avoid non significant improvements, we enforce the comparison criterion, by assuming that an assignment g_1 is better than an assignment g_2 if and only if the objective function value of g_1 is at least 0.1% lower than that of g_2 (i.e., the ratio of between difference of solution values, $v(g_1,O) - v(g_2,O)$, and the old solution value $v(g_2,O)$ must be smaller than $\delta = -0.001$).

Algorithm 7: `ILS algorithm`

```
1  g* ← initialSolution(L,P) // Get greedy assignment
2  g ← g*;
3  nonImprovingIter ← 0;
4  while nonImprovingIter < IWI do
5      g^w ← g // Initialize the best solution in loop
6      g' ← mostFrequentLocalNeighbourhood(g);
7      if (v(g^w,O) − v(g',O)) / v(g^w,O) ≥ δ then
8          g^w ← g';
9      g' ← insideShelfLocalNeighbourhood(g);
10     if (v(g^w,O) − v(g',O)) / v(g^w,O) ≥ δ then
11         g^w ← g';
12     g' ← insidePavilionNeighbourhood(g);
13     if (v(g^w,O) − v(g',O)) / v(g^w,O) ≥ δ then
14         g^w ← g';
       // Update best global solution
15     nonImprovingIter ← nonImprovingIter + 1;
16     if (v(g*,O) − v(g^w,O)) / v(g*,O) ≥ δ then
17         g* ← g^w;
18         nonImprovingIter ← 0;
19     g ← perturbation(g*);
```

In the local search we use as neighbourhood structures simple location swaps between two products. The first neighbourhood (*mostFrequentLocalNeighbourhood*) is obtained by swapping the assignments of two products belonging to the subset of 20% most required

products. The second (*insideShelfNeighbourhood*) is obtained by swapping the assignments of two products assigned to locations in the same shelf, working as an intensification of the search. The third neighbourhood (*insidePavilionNeighbourhood*) is a wider search: the locations of pairs of products assigned to the same pavilion are swapped.

The neighbourhood structures work following the same steps: (i) randomly choose two products, (ii) check if the swap between these products is valid, (iii) evaluate the change in objective function and store the best solution found.

Three rules are used to check the swap validity: (1) all swaps that assign a product to a forbidden position are not allowed; (2) no swaps are allowed between a product belonging to a strongly isolated type and a product not belonging to a strongly isolated type; (3) no swaps between products of different types are allowed. We remark that validity does not imply feasibility. In fact, while the two first rules were introduced to avoid infeasible solutions in the search, the third was created to filter the moves in order to avoid the recalculation of the penalties related to weak isolated types. It can be noticed that the third rule makes the second redundant: second and third rules are complementary, indeed. We tested two algorithm versions in the numeric tests, one with rules (1) and (2) (*type-free* swap policy), and the other with rules (1) and (3) (*same-type* swap policy).

In order to evaluate the solution after a swap, we reevaluate the distances of the routes affected by that swap and the total of products with weak isolation constraints allocated in altered areas. As the number of swaps performed during the algorithm execution is huge and the number of orders can easily reach some thousands, the procedures to evaluate them must have a strong performance.

When all neighbourhoods have been explored and the global best solution has possibly been updated, the best global solution undergoes a perturbation. This perturbation consists in $|P|/20$ unconstrained and valid swaps, chosen randomly and applied in a loop. The resulting solution is then used as the initial solution in the next ILS iteration.

4.5.1 Solution evaluation

Concerning the evaluation of the routing distance, we combine two ideas: first, we make use of different TSP algorithms depending on the size of the instance and, second, we keep track of routes that passes from each position, and we take advantage from this by recalculating only those picking distances of routes containing products involved in the swap. In this case, initially the algorithm retrieves all the routes affected and checks if a route passes from both locations where the products are currently allocated. If the route has both locations on it, the re-evaluation is not necessary (because the set of locations to be visited does not change), otherwise the route is reevaluated with the new location in place of the old one.

The evaluation is controlled by the parameter $TSP(\alpha,\beta)$. In this parameter, the constants α and β are two integers representing the order size thresholds used to choose each algorithm method for estimating the minimum distance to visit the product locations in a route. Let $|O|$ be the number of items in an order O in the tuple ω, if $|O| \leq \alpha$ an exhaustive search is run (i.e., all the possibilities are tested and then the distance found is optimal). Otherwise, if $\alpha < |O| \leq \beta$, a closest neighbour algorithm is used to initialise the route and a subsequent quick local search is performed. In this local search, $(|O|-1)$ swaps among two consecutive locations are tested in $(|O|-1)$ iterations (always departing from the first to the last product) and the current solution is updated when the swap reduces the smaller distance. Finally, if $|O| > \beta$, only the closest neighbour heuristic is performed.

The complexity of the route evaluation method described above is easily seen to be quadratic, as the exponential time approach is limited depending on the number of visited points. Though not being able to guarantee an optimal route, it is better suited to our purpose

of using non block-based warehouse layouts than algorithms based on the method proposed in [95].

We use Algorithm 8 below in order to explain how the evaluation of penalties is done by breaking weak isolation constraints. This pseudo code shows the process for shelves, but, in practice, it can be easily replicated for cells and pavilions.

Algorithm 8: `Isolation penalty evaluation after a swap`

```
1  totalPenalty ← 0;
2  S ← allShelves(L);
3  T^w ← WeakIsolationTypes();
4  P_s ← productsAllocated(g,s)   ∀s ∈ S;
5  foreach s ∈ S do
6      P_i ← {p | p ∈ P_s, type(p) ∈ T^w};
7      P_f ← {p | p ∈ P_s, type(p) ∉ T^w};
8      T_s ← {type(p) | p ∈ P_s};
       // Group products with isolation constraints by type
9      H_t ← {p ∈ P_i | type(p) = t}   ∀t ∈ T^w;
10     if |P_i| = 0 or |distinct(T_s)| = 1 then
11         continue;
12     penalty ← 0;
13     x ← max_{t∈T^w}(|H_t|)// Type with max cardinality
14     r ← x;
15     if |P_f| ≥ |P_i| then
16         penalty ← W_pen · |P_i|^2 / |P_s|;
17     else
18         penalty ← W_pen · (|P_f|^2 + r) / |P_s|;
19     totalPenalty ← totalPenalty + penalty;
```

First, the algorithm gets all the allocated products and groups them by shelves (line 4). Products are grouped by type for each shelf (line 9) and then the algorithm counts the number of different product types assigned to the shelf. If there is only one type assigned to the shelf or if no assigned product has an isolation constraint $\iota \in I | n =$ "*shelf*" (see isolation constraint definition on Section 4.3), no penalty is applied (lines 11 to 13). Otherwise, the actual penalty evaluation is performed (lines 14 to 22).

The penalty strategy is based on the division of the products assigned to a structure (which can be a shelf, a cell or a pavilion) into two groups, one with isolation constraints and another without. The group with less assignment in this structure is defined as the minority, while the other one is said to be the majority.

The method tries to push the search through the predominant configuration by penalizing minority groups. For example, if the shelf is mostly occupied by products belonging to types without isolation constraints, it penalizes the products with isolation constraints (lines 17 and 18) that are the minority. In the same way, it penalizes products belonging to types without isolation constraints if they are the minority in the shelf (lines 19 and 20). Moreover, instead of simply counting the number of products and to distinguish among similar assignments, the proportion of products belonging to the minority/majority over the total number of products is considered. This decision was taken due to the fact that only counting the products was causing no changes after a swap in the total penalty.

4.6 Instance sets

In order to test the algorithm performances, we built up several instance sets, which are to be described in this section. We created three warehouse layouts W_1, W_2 and W_3 with several differences between them, not only regarding the number of positions, but also regarding how these positions are distributed in the area. A short description of the warehouses is

provided in Table 4.1. Visual representations of the warehouses are provided in Figures 4.3, 4.4 and 4.5. The shelves are defined by the rectangles with circles on the corners. The dashed lines are the corridors and the dashed arrows are the connections between the corridors. The accumulation/expedition points are represented by isolated circles, as well as the connection between two pavilions is represented by a triangle (see the detail in the centre of Figure 4.5). In these figures, it is possible to notice some of the aforementioned characteristics of these generalised warehouses, as the presence of heterogeneous shelves or blocks, the mandatory moving sense and the multiple accumulation/expedition points.

FIGURE 4.3: Layout of the warehouse W_1

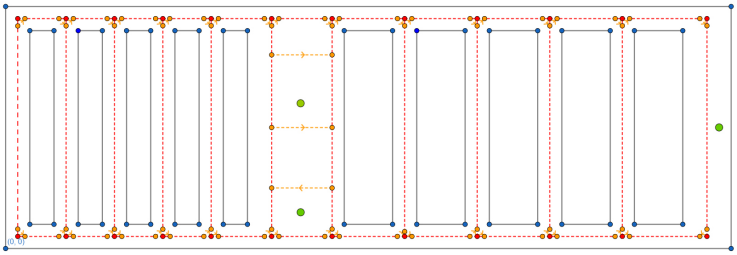


FIGURE 4.4: Layout of the warehouse W_2

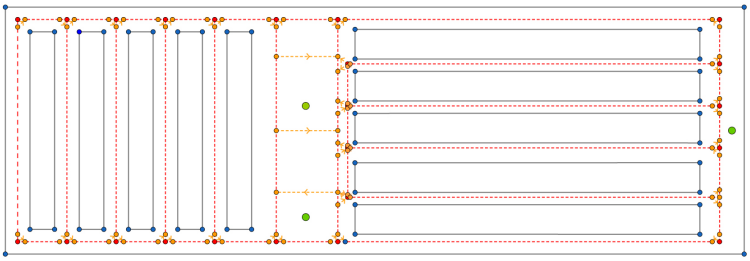
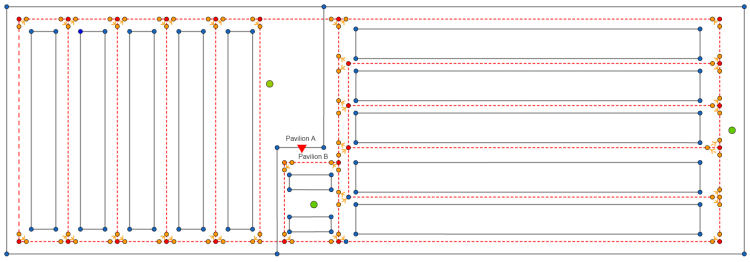


FIGURE 4.5: Layout of the warehouse W_3



The experiments were performed in three steps. In the first step, the goal was the analysis of the algorithm performance by considering only the travelled distance and thus its suitability at dealing with directly calculated distances. In the second step we tested the performance by considering the incompatibility and isolation constraints, in order to check if these constraints were well handled in the algorithm. The third step was aimed at analysing the algorithm performance over a set of large instances, the word large referring to a higher number of products and orders.

By using the instance set I_1, we tested both the insertion of products and the algorithm parameters. We experimented the insertion of two sets of products in the warehouses, one with 100 and the other with 200 products. For each product set, we tested scenarios with $|\omega| = 500$, 1000 and 5000 orders. For each combination of warehouse, number of products

TABLE 4.1: Warehouse layout overview [73].

ID	pavilions	shelves	cells	accum/exp points
W_1	1	10	200	3
W_2	1	10	240	3
W_3	2	12	260	3

and number of orders, five realistic instances were created, for a total of $|I_1| = 3 \cdot 2 \cdot 3 \cdot 5 = 90$ instances. In all these instances, we used cells with only one position/level, as showed in Table 4.1.

Regarding the algorithm parameters, we investigated two values: the maximum number of IWI, used as stopping criteria, and the TSP thresholds described in Section 4.5. Three values of maximum number of IWI were tested, and three different combination of the two TSP thresholds. Additionally, we tested the local search with and without incompatibility of swaps between products of different types. In this way, $3 \cdot 3 \cdot 2 = 18$ algorithm parameter combinations were experimented.

To test the algorithm capabilities in managing incompatibility and isolation constraints, we used a subset of the initial instances, using 3 variations to each combination of warehouse, number of products and number of orders, for a total of $3 \cdot 2 \cdot 3 \cdot 3 = 54$ instances. For each of these 54 instances, we tested 5 incompatibility and isolation constraints, leading to a new instance set I_2, with $|I_2| = 5 \cdot 54 = 270$.

To analyse the algorithm run time growth and also to check its efficiency in a large warehouse, we performed the experiments summarised in Table 4.4. In these instances, there are no isolated types or prohibited allocations and the warehouse is an expanded version of the warehouse W_3, as described in Section 4.7.

In all the instances mentioned above, the number of products by order was determined following a Poisson distribution with average number of events equal to 6. The products inside each order were defined following a uniform distribution, but preventing the same product from being requested twice in the same order.

4.7 Computational evaluation

In this section, we describe the numeric experiments performed to test the efficiency of the proposed ILS algorithm. In these experiments, we observed the algorithm performance on heterogeneous sets of instances, by monitoring features of interest such as the average run time, the local search capacity of improving the initial solution, the influence of the parameters on the final solution, and the solution quality.

The algorithm was implemented in C++17 language, with source code being compiled with -O3 flag. The tests were performed on a Dell Precision Tower 3620 computer with a 3.50GHz Intel Xeon E3-1245 processor and 32GB of RAM memory, running the Ubuntu 16.04 LTS operational system. All the executions were performed on a single thread, without any significant concurrent process.

The results obtained for the I_1 instance set are presented in Table 4.2, grouped by the TSP evaluation parameters. Each line in this table represents the average over five instances of the computational results with a specific algorithm parameter setting. Column z_0 gives the average initial solution value produced by the greedy algorithm, column z_{best} gives the average solution value produced by the ILS, column time(s) reports the average run time of the ILS in seconds, and column $\%G$ shows the average percentage gain of z_{best} with respect to z_0.

We start our analysis from the effect of the TSP evaluation parameters on the solution value. As the constructive algorithm is not affected by these parameters, it provides always the same initial solution to a given instance, but with a different objective function value, due to the difference in the solution evaluation.

As expected, using exact evaluations (and better heuristics) in more routes leads to a decrease in the objective function values, but considering the initial solution value, this variation is inferior to 5% in total from the most precise to the less precise evaluation, which suggests that simple heuristics are sufficiently efficient to check the quality of a storage assignment. This evidence is in accordance with what is stated in the literature and practised in real scenarios, mainly by the pickers, that tend to follow the most intuitive and sub-optimal routes [32] [39]. On the other hand, the improvement on evaluation precision is unequivocally counterbalanced by a significant run time increase if the whole algorithm is considered. The total run time using the $TSP(7,11)$ configuration is over the double of the total run time using the $TSP(5,9)$ configuration.

An interesting effect of the TSP evaluation parameter in the solution is the negative correlation between the evaluation precision and the improvement obtained by the local search. In other words, if we increase the number of routes that are evaluated by an exact method (or better heuristics), we get less improvement over the greedy solution. A possible cause of this result is the higher probability of a travel distance over-estimation when evaluating good-quality solutions if the evaluation precision is low. This could guide the search to a region with low-quality solutions, increasing the convergence time without improving the solution quality.

In the same sense, it is noticeable that an increase in the number of products and orders in the instances also reduces the gains over the initial solution. This is an expected result, as it is harder to balance all picker route distances if there are more routes to balance and more points to visit among the different routes.

When comparing the results of scenarios with the same evaluation parameters but different IWI values, it is possible to notice that a higher IWI only slightly improves the average gains over the initial solution (on average less than 0.7 percentage points increase for each two more IWI), even with significantly higher run times. Among the hypotheses to explain this behaviour, we can cite a bad performance of the perturbation method to escape from local optima, a too quick convergence of the method to values close to an average local optimum, the existence of too many local optima, or the existence of several regions of the solution space with very similar characteristics causing repetitive searches.

It is interesting to notice that when comparing "type-free swap" with "same-type swap" results, the latter on average gets better solutions with a higher run time. We expect that a more restricted swap could lead to a quick conversion and thus a worse solution. As the differences between solutions are relatively small, while between run times are relevant, we can suppose that the method, when using a more restricted local search, performs more but smaller improvements. This explanation is compatible with concerns about meta-heuristic parameter calibration, in which a developer tries to balance the size of intensification and diversification steps in order to find a better algorithm performance.

In Table 4.3, we show the algorithm results for instances with isolation and allocation incompatibility constraints. In the table, Φ_0 gives the average value of the initial penalties, Φ_{best} the average value of the penalties in the best solutions produced by the ILS, $\%G_z$ the average percentage gain over the initial objective function values, and $\%G_\Phi$ the average percentage gain over the initial penalties. In Table 4.3, the block *Isolation 1* refers to instances containing one type of weak isolation constraints, the block *Isolation 2* refers to instances containing one type of weak isolation and one type of strong isolation constraints. Besides that, both the blocks *Isolation 1* and *Isolation 2* in the table are subjected to allocation incompatibilities.

TABLE 4.2: Computational results on instance set I_1 [73]. Each line is an average on 90 instances.

Swap policy	IWI	$TSP(5,9)$				$TSP(6,10)$				$TSP(7,11)$			
		z_0	z_{best}	time(s)	$\%G$	z_0	z_{best}	time(s)	$\%G$	z_0	z_{best}	time(s)	$\%G$
Same-type	6	1675434.0	1025435.0	683.0	41.88	1644702.6	1059339.3	788.0	38.27	1604009.9	1149117.5	1519.7	30.85
	8	1675434.0	1016371.8	830.8	42.70	1644702.6	1047753.8	991.4	39.19	1604009.9	1137799.8	1709.8	31.47
	10	1675434.0	1007464.1	1038.8	43.34	1644702.6	1042574.2	1126.9	39.70	1604009.9	1130933.5	2078.8	32.10
Type-free	6	1675434.0	1035816.2	589.0	41.39	1644702.6	1065611.6	722.7	37.59	1604009.9	1153347.8	1224.9	30.18
	8	1675434.0	1030057.3	709.2	41.85	1644702.6	1059577.0	893.1	38.21	1604009.9	1144094.8	1525.9	30.75
	10	1675434.0	1024599.0	866.0	42.26	1644702.6	1055123.4	1090.6	38.56	1604009.9	1143325.2	1858.5	31.20

TABLE 4.3: Computational results for instances with isolation and incompatibility constraints on instance set I_2 [73]. Each line is an average on 3 instances. Parameters used: 8 IWI, $TSP(7,11)$. Swap policy: *type-free*.

Id	$\|P\|$	W	$\|\omega\|$	Isolation 1							Isolation 2						
				z_0	Φ_0	z_{best}	Φ_{best}	$\%G_z$	$\%G_\Phi$	time(s)	z_0	Φ_0	z_{best}	Φ_{best}	$\%G_z$	$\%G_\Phi$	time(s)
1	100	W_1	500	405287.9	51048.9	225120.3	37610.6	44.32	25.38	276.5	632041.1	344081.8	503372.6	317109.3	20.36	7.84	288.2
2			1000	749806.0	49873.4	430050.9	31858.6	42.63	35.43	713.7	1187816.0	621282.7	977614.8	590412.1	17.64	4.96	499.6
3			5000	3558644.9	49217.2	2193491.7	52373.3	38.36	-6.53	2839.9	5659723.9	2795037.9	4846198.5	2786653.8	14.37	0.30	2119.7
4		W_2	500	383964.6	54282.0	240575.6	42760.2	37.33	20.93	261.0	613230.2	353249.2	534527.6	341856.0	12.82	3.15	203.0
5			1000	704103.1	51699.1	469837.4	42271.7	33.28	17.90	680.6	1162098.5	644019.8	1025637.5	619162.5	11.77	3.90	489.7
6			5000	3332955.4	54780.4	2276342.0	52617.7	31.70	2.89	3005.5	5555117.6	2933844.9	4960398.8	2911717.1	10.71	0.76	2388.9
7		W_3	500	371185.3	58468.7	251748.4	36607.4	32.19	36.99	334.2	575261.8	346403.8	518039.3	334616.3	9.94	3.39	402.6
8			1000	673638.7	49932.3	504716.8	38462.8	25.07	21.97	611.3	1096946.6	632364.3	1001514.8	621692.1	8.70	1.73	519.2
9			5000	3221278.3	58714.3	2444756.2	46692.6	24.10	19.00	3519.7	5296059.8	2928326.8	4907968.1	2915739.4	7.32	0.43	2897.5
10	200	W_1	500	438229.7	27666.7	326534.0	26333.3	25.48	4.87	281.8	728354.9	363274.5	576015.8	297922.2	20.98	18.17	278.5
11			1000	854100.0	32666.7	633261.7	24000.0	25.83	26.33	766.9	1223601.2	503051.5	1022193.0	412985.7	16.44	17.91	622.5
12			5000	4208520.8	51535.1	3145456.1	59535.1	25.26	-25.94	2824.4	5413919.5	1718497.8	4499844.1	1673331.1	16.88	2.62	3092.1
13		W_2	500	425402.5	30307.5	306401.6	25061.0	27.97	14.48	254.7	711924.2	362643.5	546403.6	277711.3	23.24	23.43	282.3
14			1000	814109.8	28141.2	622948.2	23876.2	23.47	15.01	760.5	1210574.2	515456.9	993881.1	447135.4	17.88	13.18	678.9
15			5000	4046685.1	32227.3	2971407.6	30121.8	26.57	6.28	3593.7	5265289.3	1724671.7	4412597.4	1637695.8	16.19	5.02	3621.2
16		W_3	500	431234.1	32694.1	356368.3	23407.3	17.36	28.40	441.0	711317.5	360032.5	611663.2	297630.6	13.98	17.25	229.6
17			1000	833826.9	31042.3	689633.4	24334.8	17.29	21.15	775.5	1208464.6	504536.9	1055921.7	426959.7	12.61	15.21	532.4
18			5000	4165773.8	30477.8	3403163.1	31035.4	18.31	-2.27	3409.6	5349024.8	1733807.5	4761622.8	1645191.5	10.98	5.11	3864.5
Totals				1731243.0	42656.8	1257468.8	42760.2	37.33	20.93	1486.4	2523801.4	1100055.3	2180204.3	341856.0	12.82	3.15	1362.0

We can first observe a satisfactory improvement over the initial solution obtained by the local search, although smaller than that observed in the previous results. Nevertheless, in instances without hard isolation constraints, this metric raises to 19.4% and 15.1% in blocks *Isolation 1* and *Isolation 2*, respectively. These results may suggest that if the constructive algorithm does not handle well isolation constraints, then local search has troubles in improving the solution.

We notice that the local search reduces proportionally less the penalty value than the overall objective function value, except in instances with 200 products in warehouse W_3 (lines 16 to 18 of Table 4.3), suggesting that the method could be giving more relevance to travel distance.

The average run time is smaller than one hour for almost all instance settings, except on instances on lines 15 and 18 and *Isolation 2*. It suggests that the method converges in an acceptable time even in more realistic and complex mid-size instances.

In Table 4.4, z_0, z_{best}, $\%G_z$ and time represent the average values for five different instances. In this case, the gain over the initial solution obtained by the search is between 2% and 4%. This interval is far inferior to those observed in smaller instances, as expected, but still relevant in a real life operation. The run time rises approximately in a linear way with respect to the number of orders (6.37 times for instances with 800 products and 7.18 times for instances with 1600 products), but very fast with respect to the number of products to be allocated, getting to an average of 48 hours for instances with 1600 products. It suggests that the amount of order data is not a critical factor in algorithm scalability.

An interesting result can be noticed when observing the initial solution value. While this

TABLE 4.4: Computational results on large instances. Parameters used: 8 IWI, $TSP(7,11)$. Each line is an average on 5 instances.

Id	$\|P\|$	$\|\omega\|$	z_0	z_{best}	$\%G_z$	time(s)
1	400	5000	4150015.0	3438747.4	3.43	6727.4
2	800	2000	1737677.0	1467340.2	3.11	7035.9
3	800	10000	8945296.0	7596591.4	3.02	44700.2
4	1600	2000	1722730.4	1518573.6	2.37	23162.5
5	1600	10000	9025347.8	7749001.4	2.83	166274.7

seems to be directly proportional to the number of orders (as more orders mean more routes), it does not change with the number of products. This can be explained by the proportional growth of warehouse capacity per area, which makes the average picking density to remain similar, causing a similar route distances.

4.8 Conclusions

In this paper, we study the Storage Location Assignment Problem with Product-Cell Incompatibility and Isolation Constraints. This generalises the classic Storage Location Problem, by introducing the possibility of preventing the placement of certain products in certain positions and keeping certain product families isolated. In a pharmaceutical logistic operators context, this problem is often encountered when aiming at improving warehouse performance while maintaining flexibility in defining warehouse layout. Additionally, the work also described how to process the warehouse input data, which can be useful in several other studies that, as this one, need to deal with non-conventional warehouse layouts or for situations where design changes also need to be evaluated.

To solve the problem, an Iterated Local Search method is proposed and tested on a large heterogeneous set of instances, demonstrating its suitability in providing good quality solutions within a satisfactory run time. Analysis enlighten that variations on the stopping criteria involve significant changes in the run time, but just slight changes in the solution quality. On the other hand, the variations in the parameters related to the TSP solutions (to determine the pickers' routes) prove to be very influential in both run time and solution quality. Large instances are well handled by the ILS algorithm, though with higher run times, with this raise strongly related to the number of products to be allocated and less to the number of orders considered. Instances with isolation and incompatibility constraints present lower improvements in the local search phase, but still relevant for commercial purposes.

In the future, we aim to investigate more deeply the influence of initial allocation of strongly isolated types on the overall performance of the optimization method. General weighting of the different terms in the objective function 4.1 should be addressed, after an extensive analysis of several factors, e.g. incidence of the distribution of the various types of penalties among products, or the ratio between warehouse dimensions and the number of orders. This could also lead to the development of interesting multi-objective optimization algorithms. Also, the suitability of the method to more dynamic situations should be investigated, mainly when different sources of information are available for orders. It could be interesting to create stronger strategies to avoid route re-evaluation or discard non-improving swaps, as this could significantly reduce the run time and allow a broader search.

Chapter 5

Conclusions and future research

In this thesis, we address three distinct optimization problems arising in production and healthcare contexts: (1) an integrated workforce allocation and scheduling problem in a two-stage flexible flow shop for perishable products, aiming to minimize unscheduled orders, weighted tardiness, and production costs; (2) a dynamic assignment of server configurations in healthcare facilities to minimize patient waiting times; and (3) a storage location assignment problem with incompatibilities and isolation constraints to minimize picker travel distances in pharmaceutical warehouses.

The first problem involves allocating workers and scheduling orders in a two-stage production environment. A constructive heuristic was developed to prioritize the minimization of unscheduled orders, tardiness, and costs. This heuristic was embedded into three metaheuristics: Random Multi-Start (MR), Biased Random Key Genetic Algorithm (BRKGA), and Variable Neighborhood Search (VNS). Computational results demonstrated that BRKGA consistently outperformed the other approaches, achieving high-quality solutions efficiently.

The second problem focuses on assigning configurations to servers in healthcare facilities under dynamic and uncertain conditions. We proposed the Scenario-Based Planning and Recombination Approach (SBPRA), which extends the traditional Scenario-Based Planning Approach (SBPA) by recombining solutions across scenarios. SBPRA, combined with a Reduced Variable Neighborhood Search (RVNS), significantly improved solution quality compared to SBPA and other baseline methods, achieving a 38% average improvement. It balances computational efficiency with solution effectiveness, particularly when using 30–90 scenarios and re-optimization intervals of 60 minutes.

The third problem addresses the Storage Location Assignment Problem with Product-Cell Incompatibility and Isolation Constraints (SLAP-PCIIC), a variant of the classical Storage Location Assignment Problem (SLAP), incorporating constraints such as product-cell incompatibility (e.g., refrigeration) and isolation (e.g., toxicity). An Iterated Local Search (ILS) algorithm was proposed and tested on diverse instances, demonstrating strong performance. The ILS effectively handled large-scale instances and provided robust solutions within reasonable computational times, even under stringent operational constraints.

Future research can build on these findings by exploring dynamic extensions of the studied problems. For example, dynamic changes in worker assignments could be incorporated into the flexible flow shop model, while real-time data integration could enhance the effectiveness of SBPRA in outpatient facilities. A comparative study of these proposed methods with established stochastic optimization techniques, such as the Progressive Hedging Algorithm (PHA), would also be highly valuable. Such comparisons could provide deeper insights into the performance trade-offs and scalability of these algorithms, as well as offer opportunities for integrating more sophisticated approaches to stochastic decision-making. Multi-objective optimization is another promising direction, as many practical scenarios involve conflicting objectives such as cost minimization and service quality maximization. Developing algorithms that explicitly address these trade-offs would further enhance the applicability of the proposed methods.

Another area of interest is the incorporation of additional real-world constraints to better reflect operational realities. For instance, time-budget limitations and dynamic abandonment behaviors in healthcare facilities could provide more realistic models. Similarly, enhanced decision-making frameworks that combine predictive analytics with optimization algorithms could significantly improve operational performance, particularly when applied at earlier planning stages.

Data-driven techniques, such as machine learning, also hold promise for advancing optimization methodologies. These approaches could be used to refine neighborhood searches, improve consensus functions, or enhance initial solution quality in iterative methods like ILS. By leveraging such techniques, future research could reduce computational times and improve solution quality, making these methods even more practical for real-world applications.

Across these three problems, our work highlights the large demand of advanced optimization techniques for addressing real-world challenges involving resource allocation, scheduling, and logistics. While focused on distinct applications, the problems share commonalities in their need to balance complex constraints and dynamic conditions. Future work could explore integrating real-time data, hybrid algorithmic approaches, and multi-objective optimization to address evolving challenges in production, healthcare, and logistics.

Bibliography

[1] M. Abaalkhayl, H. Al-Najjar, and N. Abukraym. "Patient satisfaction study in outpatient setting in tertiary academic hospital patient satisfaction in outpatient clinics". In: *Journal of Population Therapeutics and Clinical Pharmacology* 30 (17 2023), 259–267.

[2] Z. A. Abdalkareem, A. Amir, M. A. Al-Betar, P. Ekhan, and A. I. Hammouri. "Healthcare scheduling in optimization context: a review". In: *Health and Technology* 11.3 (2021), pp. 445–469.

[3] R. Accorsi, G. Baruffaldi, and R. Manzini. "Picking efficiency and stock safety: A bi-objective storage assignment policy for temperature-sensitive products". In: *Computers & Industrial Engineering* 115 (2018), pp. 240–252.

[4] R. Aldrighetti, I. Zennaro, S. Finco, and D. Battini. "Healthcare supply chain simulation with disruption considerations: a case study from northern Italy". In: *Global Journal of Flexible Systems Management* 20.1 (2019), pp. 81–102.

[5] T. Alves de Queiroz, B. Bolsi, V. L. de Lima, M. Iori, and A. Kramer. "Assigning Multi-skill Configurations to Multiple Servers with a Reduced VNS". In: *Variable Neighborhood Search*. Ed. by A. Sleptchenko, A. Sifaleras, and P. Hansen. Cham: Springer Nature Switzerland, 2023, pp. 97–111.

[6] T. Alves de Queiroz, M. Iori, A. Kramer, and Y.-H. Kuo. "Dynamic scheduling of patients in emergency departments". In: *European Journal of Operational Research* 310.1 (2023), pp. 100–116. ISSN: 0377-2217.

[7] N. T. J. Bailey. "A Study of Queues and Appointment Systems in Hospital Outpatient Departments with Special Reference to Waiting Times". In: *Journal of the Royal Statistical Society* 14 (2 1952), 185–199.

[8] R. Bargetto, T. Garaix, and X. Xie. "A branch-and-price-and-cut algorithm for operating room scheduling under human resource constraints". In: *Computers & Operations Research* 152 (2023), p. 106136. ISSN: 0305-0548.

[9] D. Battini, M. Calzavara, A. Persona, and F. Sgarbossa. "Order picking system design: the storage assignment and travel distance estimation (SA&TDE) joint method". In: *International Journal of Production Research* 53.4 (2015), pp. 1077–1093.

[10] R. W. Bent and P. Van Hentenryck. "Scenario-Based Planning for Partially Dynamic Vehicle Routing with Stochastic Customers". In: *Operations Research* 52.6 (2004), pp. 977–987.

[11] B. P. Berg, B. T. Denton, S. Ayca Erdogan, T. Rohleder, and T. Huschka. "Optimal booking and scheduling in outpatient procedure centers". In: *Computers & Operations Research* 50 (2014), pp. 24–37.

[12] D. Bo, L. Yanfei, R. Haisheng, L. Xuejun, and L. Cuiqing. "Research on Optimization for Safe Layout of Hazardous Chemicals Warehouse Based on Genetic Algorithm". In: *IFAC-PapersOnLine* 51.18 (2018). 10th IFAC Symposium on Advanced Control of Chemical Processes ADCHEM 2018, pp. 245–250. ISSN: 2405-8963.

[13] E. Bolandifar, N. DeHoratius, and T. Olsen. "Modeling abandonment behavior among patients". In: *European Journal of Operational Research* 306.1 (2023), pp. 243–254.

[14] J. Bolaños Zuñiga, J. A. Saucedo Martínez, T. E. Salais Fierro, and J. A. Marmolejo Saucedo. "Optimization of the Storage Location Assignment and the Picker-Routing Problem by Using Mathematical Programming". In: *Applied Sciences* 10.2 (2020), p. 534.

[15] B. Bolsi, V. de Lima, T. Alves de Queiroz, and M. Iori. "Integrated Workforce Scheduling and Flexible Flow Shop Problem in the Meat Industry". In: *Advances in Production Management Systems. Artificial Intelligence for Sustainable and Resilient Production Systems*. Ed. by A. Dolgui, A. Bernard, D. Lemoine, G. von Cieminski, and D. Romero. Springer International Publishing, 2021, pp. 594–602.

[16] B. Bolsi, A. Kramer, T. A. de Queiroz, and M. Iori. "Optimizing a dynamic outpatient facility system with multiple servers". In: *Optimization in Artificial Intelligence and Data Sciences: ODS, First Hybrid Conference, Rome, Italy, September 14-17, 2021*. Springer. 2022, pp. 247–258.

[17] B. Bolsi, V. L. de Lima, T. Alves de Queiroz, and M. Iori. "Heuristic algorithms for integrated workforce allocation and scheduling of perishable products". In: *International Journal of Production Research* 61.20 (2023), pp. 7048–7063.

[18] B. Bolsi, V. L. de Lima, T. A. de Queiroz, and M. Iori. "Integrated workforce scheduling and flexible flow shop problem in the meat industry". In: *IFIP International Conference on Advances in Production Management Systems*. Springer. 2021, pp. 594–602.

[19] B. Bolsi, T. A. de Queiroz, V. L. de Lima, A. Kramer, and M. Iori. "Assigning multi-skill configurations to multiple servers with a Scenario-Based Planning and Recombination Approach". In: *Computers & Operations Research* (2024), p. 106719.

[20] E. Bottani, G. Casella, and T. Murino. "A hybrid metaheuristic routing algorithm for low-level picker-to-part systems". In: *Computers & Industrial Engineering* 160 (2021), p. 107540.

[21] T. R. Bovim, M. Christiansen, A. N. Gullhav, T. M. Range, and L. Hellemo. "Stochastic master surgery scheduling". In: *European Journal of Operational Research* 285.2 (2020), pp. 695–711. ISSN: 0377-2217.

[22] N. Boysen, R. De Koster, and F. Weidinger. "Warehousing in the e-commerce era: A survey". In: *European Journal of Operational Research* 277.2 (2019), pp. 396–411.

[23] J. Brimberg, S. Salhi, R. Todosijević, and D. Urošević. "Variable Neighborhood Search: The power of change and simplicity". In: *Computers & Operations Research* 155 (2023), p. 106221.

[24] H. Cambazard and N. Catusse. "Fixed-parameter algorithms for rectilinear steiner tree and rectilinear traveling salesman problem in the plane". In: *European Journal of Operational Research* 270.2 (2018), pp. 419–429.

[25] F. Camiat, M. I. Restrepo, J.-M. Chauny, N. Lahrichi, and L.-M. Rousseau. "Productivity-driven physician scheduling in emergency departments". In: *Health Systems* 10.2 (2021), pp. 104–117.

[26] T. Cayirli and E. Veral. "Outpatient scheduling in health care: a review of literature". In: *Production and Operations Management* 12.4 (2003), pp. 519–549.

[27] F. Chen, G. Xu, and Y. Wei. "Heuristic routing methods in multiple-block warehouses with ultra-narrow aisles and access restriction". In: *International Journal of Production Research* 57.1 (2019), pp. 228–249.

[28] V. Choudhary, A. Shastri, S. Silswal, and A. Kulkarni. "A Comprehensive Review of Patient Scheduling Techniques with Uncertainty". In: *Handbook of Formal Optimization*. Ed. by A. J. Kulkarni and A. H. Gandomi. Singapore: Springer Nature Singapore, 2023, pp. 1–21. ISBN: 978-981-19-8851-6.

[29] Y.-F. Chuang, H.-T. Lee, and Y.-C. Lai. "Item-associated cluster assignment model on storage allocation problems". In: *Computers & Industrial Engineering* 63.4 (2012), pp. 1171–1177.

[30] J.-F. Côté, T. Alves de Queiroz, F. Gallesi, and M. Iori. "A branch-and-regret algorithm for the same-day delivery problem". In: *Transportation Research Part E: Logistics and Transportation Review* 177 (2023), p. 103226.

[31] R. De Koster, T. Le-Duc, and K. J. Roodbergen. "Design and control of warehouse order picking: A literature review". In: *European Journal of Operational Research* 182.2 (2007), pp. 481–501.

[32] R. De Santis, R. Montanari, G. Vignali, and E. Bottani. "An adapted ant colony optimization algorithm for the minimization of the travel distance of pickers in manual warehouses". In: *European Journal of Operational Research* 267.1 (2018), pp. 120–137.

[33] N. Dellaert and J. Jeunet. "A variable neighborhood search algorithm for the surgery tactical planning problem". In: *Computers & Operations Research* 84 (2017), pp. 216–225. ISSN: 0305-0548.

[34] M. Di Mascolo, C. Martinez, and M.-L. Espinouse. "Routing and scheduling in Home Health Care: A literature survey and bibliometric analysis". In: *Computers & Industrial Engineering* 158 (2021), p. 107255.

[35] A. S. Dijkstra and K. J. Roodbergen. "Exact route-length formulas and a storage location assignment heuristic for picker-to-parts warehouses". In: *Transportation Research Part E: Logistics and Transportation Review* 102 (2017), pp. 38–59.

[36] C. Dosi, M. Iori, A. Kramer, and M. Vignoli. "Successful implementation of discrete event simulation: integrating design thinking and simulation approach in an emergency department". In: *Production Planning & Control* 34.13 (2023), pp. 1233–1247.

[37] A. Duarte and E. G. Pardo. "Special issue on recent innovations in variable neighborhood search". In: *Journal of Heuristics* 26 (2020), pp. 335–338.

[38] A. Elalouf and G. Wachtel. "Queueing Problems in Emergency Departments: A Review of Practical Approaches and Research Methodologies". In: *Operations Research Forum* 3.1 (2022), p. 2.

[39] R. M. Elbert, T. Franzke, C. H. Glock, and E. H. Grosse. "The effects of human behavior on the efficiency of routing policies in order picking: The case of route deviations". In: *Computers & Industrial Engineering* 111 (2017), pp. 537–551.

[40] H. Emmons and G. Vairaktarakis. "The Hybrid Flow Shop". In: *Flow Shop Scheduling: Theoretical Results, Algorithms, and Applications*. Boston, MA: Springer US, 2013, pp. 161–187. ISBN: 978-1-4614-5152-5.

[41] M. E. Fontana and V. S. Nepomuceno. "Multi-criteria approach for products classification and their storage location assignment". In: *The International Journal of Advanced Manufacturing Technology* 88.9-12 (2017), pp. 3205–3216.

[42] A. Foroughi, N. Boysen, S. Emde, and M. Schneider. "High-density storage with mobile racks: Picker routing and product location". In: *Journal of the Operational Research Society* 72.3 (2021), pp. 535–553.

[43] Y. Fu, M. Zhou, X. Guo, and L. Qi. "Scheduling Dual-Objective Stochastic Hybrid Flow Shop With Deteriorating Jobs via Bi-Population Evolutionary Algorithm". In: *IEEE Transactions on Systems, Man, and Cybernetics: Systems* 50.12 (2020), pp. 5037–5048.

[44] M. Gen and L. Lin. "Multiobjective evolutionary algorithm for manufacturing scheduling problems: state-of-the-art survey". In: *Journal of Intelligent Manufacturing* 25.5 (2014), pp. 849–866.

[45] T. van Gils, K. Ramaekers, A. Caris, and R. B. de Koster. "Designing efficient order picking systems by combining planning problems: State-of-the-art classification and review". In: *European Journal of Operational Research* 267.1 (2018), pp. 1–15.

[46] J. Gonçalves and M. Resende. "Random-Key Genetic Algorithms". In: *Handbook of Heuristics*. Ed. by R. Martí, P. M. Pardalos, and M. Resende. Springer International Publishing, 2016. ISBN: 978-3-319-07153-4.

[47] L. Green. "Queueing Analysis in Healthcare". In: *Patient Flow: Reducing Delay in Healthcare Delivery*. Ed. by R. W. Hall. Boston, MA: Springer US, 2006, pp. 281–307.

[48] J. Gu, M. Goetschalckx, and L. F. McGinnis. "Research on warehouse operation: A comprehensive review". In: *European Journal of Operational Research* 177.1 (2007), pp. 1–21.

[49] F Guerriero, R Musmanno, O Pisacane, and F Rende. "A mathematical model for the Multi-Levels Product Allocation Problem in a warehouse with compatibility constraints". In: *Applied Mathematical Modelling* 37.6 (2013), pp. 4385–4398.

[50] M. R. M. Habibi, F. M. Abadi, H. Tabesh, H. Vakili-Arki, A. Abu-Hanna, and S. Eslami. "Evaluation of patient satisfaction of the status of appointment scheduling systems in outpatient clinics: Identifying patients' needs". In: *Journal of Advanced Pharmaceutical Technology & Research* 9 (2 2018), 51–55.

[51] P. Hansen and N. Mladenović. "Variable neighborhood search: Principles and applications". In: *European Journal of Operational Research* 130.3 (2001), pp. 449–467.

[52] P. Hansen, N. Mladenović, and J. A. Moreno Pérez. "Variable neighbourhood search: methods and applications". In: *Annals of Operations Research* 175.1 (2010), pp. 367–407.

[53] P. Hansen, N. Mladenović, R. Todosijević, and S. Hanafi. "Variable neighborhood search: basics and variants". In: *EURO Journal on Computational Optimization* 5 (2017), pp. 423–454.

[54] J. Huang, B. Carmeli, and A. Mandelbaum. "Control of patient flow in emergency departments, or multiclass queues with deadlines and feedback". In: *Operations Research* 63.4 (2015), pp. 892–908.

[55] F. Hwang and B. Lin. "Survey and extensions of manufacturing models in two-stage flexible flow shops with dedicated machines". In: *Computers & Operations Research* 98 (2018), pp. 103–112.

[56] M. A. Kaboudan. "A dynamic-server queuing simulation". In: *Computers & Operations Research* 25.6 (1998), pp. 431–439.

[57] K. Keung, C. Lee, and P. Ji. "Industrial internet of things-driven storage location assignment and order picking in a resource synchronization and sharing-based robotic mobile fulfillment system". In: *Advanced Engineering Informatics* 52 (2022), p. 101540. ISSN: 1474-0346.

[58] J. Kim, F. Méndez, and J. Jimenez. "Storage Location Assignment Heuristics Based on Slot Selection and Frequent Itemset Grouping for Large Distribution Centers". In: *IEEE Access* 8 (2020), pp. 189025–189035.

[59] K. J. Klassen and R. Yoogalingam. "Improving Performance in Outpatient Appointment Services with a Simulation Optimization Approach". In: *Production and Operations Management* 18.4 (2009), pp. 447–458.

[60] D. Kress, N. Boysen, and E. Pesch. "Which items should be stored together? A basic partition problem to assign storage space in group-based storage systems". In: *IISE Transactions* 49.1 (2017), pp. 13–30.

[61] Y. H. Kuo. "Integrating simulation with simulated annealing for scheduling physicians in an understaffed emergency department". In: *HKIE Transactions* 21.4 (2014), pp. 253–261.

[62] Y.-H. Kuo, J. M. Leung, C. A. Graham, K. K. Tsoi, and H. M. Meng. "Using simulation to assess the impacts of the adoption of a fast-track system for hospital emergency services". In: *Journal of Advanced Mechanical Design, Systems, and Manufacturing* 12.3 (2018), JAMDSM0073.

[63] Y.-H. Kuo, O. Rado, B. Lupia, J. M. Leung, and C. A. Graham. "Improving the efficiency of a hospital emergency department: a simulation study with indirectly imputed service-time distributions". In: *Flexible Services and Manufacturing Journal* 28 (2016), pp. 120–147.

[64] S. Lan, W. Fan, S. Yang, P. M. Pardalos, and N. Mladenovic. "A survey on the applications of variable neighborhood search algorithm in healthcare management". In: *Annals of Mathematics and Artificial Intelligence* 89 (2021), pp. 741–775.

[65] A. Legrain, J. Omer, and S. Rosat. "An online stochastic algorithm for a dynamic nurse scheduling problem". In: *European Journal of Operational Research* 285.1 (2020), pp. 196–210. ISSN: 0377-2217.

[66] J. Li, M. Moghaddam, and S. Y. Nof. "Dynamic storage assignment with product affinity and ABC classification—a case study". In: *The International Journal of Advanced Manufacturing Technology* 84.9-12 (2016), pp. 2179–2194.

[67] Z. Li, Q. Chen, N. Mao, X. Wang, and J. Liu. "Scheduling rules for two-stage flexible flow shop scheduling problem subject to tail group constraint". In: *International Journal of Production Economics* 146.2 (2013), pp. 667–678.

[68] D. V. Lindley. "The theory of queues with a single server". In: *Mathematical Proceedings of the Cambridge Philosophical Society* 48.2 (1952), 277–289.

[69] R. Linn and W. Zhang. "Hybrid flow shop scheduling: A survey". In: *Computers & Industrial Engineering* 37.1 (1999), pp. 57–61.

[70] H. R. Lourenço, O. C. Martin, and T. Stützle. "Iterated local search". In: *Handbook of metaheuristics*. Springer, 2003, pp. 320–353.

[71] W. Lu, D. McFarlane, V. Giannikas, and Q. Zhang. "An algorithm for dynamic order-picking in warehouse operations". In: *European Journal of Operational Research* 248.1 (2016), pp. 107–122.

[72] M. López-Ibáñez, J. Dubois-Lacoste, L. Pérez Cáceres, M. Birattari, and T. Stützle. "The irace package: Iterated racing for automatic algorithm configuration". In: *Operations Research Perspectives* 3 (2016), pp. 43–58.

[73] N. Mendes, B. Bolsi, and M. Iori. "An Iterated Local Search for a Pharmaceutical Storage Location Assignment Problem with Product-cell Incompatibility and Isolation Constraints". In: *Proceedings of the 24th International Conference on Enterprise Information Systems - Volume 1: ICEIS,* INSTICC. SciTePress, 2022, pp. 17–26. ISBN: 978-989-758-569-2.

[74] N. F. Mendes, B. Bolsi, and M. Iori. "A Storage Location Assignment Problem with Incompatibility and Isolation Constraints: An Iterated Local Search Approach". In: *International Conference on Enterprise Information Systems*. Springer. 2022, pp. 22–47.

[75] D. Ming-Huang Chiang, C.-P. Lin, and M.-C. Chen. "Data mining based storage assignment heuristics for travel distance reduction". In: *Expert Systems* 31.1 (2014), pp. 81–90.

[76] M. Mirzaei, N. Zaerpour, and R. De Koster. "The impact of integrated cluster-based storage allocation on parts-to-picker warehouse performance". In: *Transportation Research Part E: Logistics and Transportation Review* 146 (2021), p. 102207.

[77] M. Mirzaei, N. Zaerpour, and R. B. de Koster. "How to benefit from order data: correlated dispersed storage assignment in robotic warehouses". In: *International Journal of Production Research* 60.2 (2022), pp. 549–568.

[78] N. Mladenović and P. Hansen. "Variable neighborhood search". In: *Computers & Operations Research* 24.11 (1997), pp. 1097–1100.

[79] N. Mladenović, M. Souza, and K. Sörensen. "Special issue on "Advances in Variable Neighborhood Search"". In: *International Transactions in Operational Research* 25.1 (2018), pp. 427–427.

[80] V. R. Muppani and G. K. Adil. "A branch and bound algorithm for class based storage location assignment". In: *European Journal of Operational Research* 189.2 (2008), pp. 492–507.

[81] V. R. Muppani and G. K. Adil. "Efficient formation of storage classes for warehouse storage location assignment: a simulated annealing approach". In: *Omega* 36.4 (2008), pp. 609–618.

[82] F. Nikzad, J. Rezaeian, I. Mahdavi, and I. Rastgar. "Scheduling of multi-component products in a two-stage flexible flow shop". In: *Applied Soft Computing* 32 (2015), pp. 132–143.

[83] Ö. Öztürkoğlu. "A bi-objective mathematical model for product allocation in block stacking warehouses". In: *International Transactions in Operational Research* 27.4 (2020), pp. 2184–2210.

[84] J. C.-H. Pan, P.-H. Shih, and M.-H. Wu. "Storage assignment problem with travel distance and blocking considerations for a picker-to-part order picking system". In: *Computers & Industrial Engineering* 62.2 (2012), pp. 527–535.

[85] J. C.-H. Pan, P.-H. Shih, M.-H. Wu, and J.-H. Lin. "A storage assignment heuristic method based on genetic algorithm for a pick-and-pass warehousing system". In: *Computers & Industrial Engineering* 81 (2015), pp. 1–13.

[86] J. C.-H. Pan and M.-H. Wu. "A study of storage assignment problem for an order picking line in a pick-and-pass warehousing system". In: *Computers & Industrial Engineering* 57.1 (2009), pp. 261–268.

[87] K.-W. Pang and H.-L. Chan. "Data mining-based algorithm for storage location assignment in a randomised warehouse". In: *International Journal of Production Research* 55.14 (2017), pp. 4035–4052.

[88] M. L. Pinedo. *Scheduling: Theory, Algorithms, and Systems*. New York, USA: Springer-Verlag, 2012.

[89] T. A. Queiroz and L. R. Mundim. "Multiobjective pseudo-variable neighborhood descent for a bicriteria parallel machine scheduling problem with setup time". In: *International Transactions in Operational Research* 27.3 (2020), pp. 1478–1500.

[90] T. A. de Queiroz, B. Bolsi, V. L. de Lima, M. Iori, and A. Kramer. "Assigning Multi-skill Configurations to Multiple Servers with a Reduced VNS". In: *International Conference on Variable Neighborhood Search*. Springer Nature Switzerland Cham. 2022, pp. 97–111.

[91] S. Quintanilla, Á. Pérez, F. Ballestín, and P. Lino. "Heuristic algorithms for a storage location assignment problem in a chaotic warehouse". In: *Engineering Optimization* 47.10 (2015), pp. 1405–1422.

[92] A. Rais and A. Viana. "Operations Research in Healthcare: a survey". In: *International Transactions in Operational Research* 18.1 (2011), pp. 1–31.

[93] S. S. Rao and G. K. Adil. "Optimal class boundaries, number of aisles, and pick list size for low-level order picking systems". In: *IIE Transactions* 45.12 (2013), pp. 1309–1321.

[94] S. A. B. Rasmi, Y. Wang, and H. Charkhgard. "Wave order picking under the mixed-shelves storage strategy: A solution method and advantages". In: *Computers & Operations Research* 137 (2022), p. 105556. ISSN: 0305-0548.

[95] H. D. Ratliff and A. S. Rosenthal. "Order-picking in a rectangular warehouse: a solvable case of the traveling salesman problem". In: *Operations Research* 31.3 (1983), pp. 507–521.

[96] J Reyes, E Solano-Charris, and J Montoya-Torres. "The storage location assignment problem: A literature review". In: *International Journal of Industrial Engineering Computations* 10.2 (2019), pp. 199–224.

[97] K. J. Roodbergen and R. Koster. "Routing methods for warehouses with multiple cross aisles". In: *International Journal of Production Research* 39.9 (2001), pp. 1865–1883.

[98] A. Rossi, A. Puppato, and M. Lanzetta. "Heuristics for scheduling a two-stage hybrid flow shop with parallel batching machines: application at a hospital sterilisation plant". In: *International Journal of Production Research* 51.8 (2013), pp. 2363–2376.

[99] R. Ruiz and J. Vázquez-Rodríguez. "The hybrid flow shop scheduling problem". In: *European Journal of Operational Research* 205.1 (2010), pp. 1–18.

[100] S. Saghafian, G. Austin, and S. J. Traub. "Operations research/management contributions to emergency department patient flow optimization: Review and research prospects". In: *IIE Transactions on Healthcare Systems Engineering* 5.2 (2015), pp. 101–123.

[101] A. Scholz, S. Henn, M. Stuhlmann, and G. Wäscher. "A new mathematical programming formulation for the single-picker routing problem". In: *European Journal of Operational Research* 253.1 (2016), pp. 68–84.

[102] A. Silva, L. C. Coelho, M. Darvish, and J. Renaud. "Integrating storage location and order picking problems in warehouse planning". In: *Transportation Research Part E: Logistics and Transportation Review* 140 (2020), p. 102003.

[103] A. Silva, K. J. Roodbergen, L. C. Coelho, and M. Darvish. "Estimating optimal ABC zone sizes in manual warehouses". In: *International Journal of Production Economics* 252 (2022), p. 108579.

[104] F. H. Staudt, G. Alpan, M. Di Mascolo, and C. M. T. Rodriguez. "Warehouse performance measurement: a literature review". In: *International Journal of Production Research* 53.18 (2015), pp. 5524–5544.

[105] Q. Su, X. Yao, P. Su, J. Shi, Y. Zhu, and L. Xue. "Hospital registration process reengineering using simulation method". In: *Journal of Healthcare Engineering* 1.1 (2010), pp. 67–82.

[106] Y.-T. L. T.-S. Lee. "A review of scheduling problem and resolution methods in flexible flow shop". In: *International Journal of Industrial Engineering Computations* 10.1 (2019), pp. 67–88.

[107] Y. Tan, L. Mönch, and J. Fowler. "A hybrid scheduling approach for a two-stage flexible flow shop with batch processing machines". In: *Journal of Scheduling* 21.2 (2018), pp. 209–226.

[108] E. Thanos, T. Wauters, and G. Vanden Berghe. "Dispatch and conflict-free routing of capacitated vehicles with storage stack allocation". In: *Journal of the Operational Research Society* (2019), pp. 1–14.

[109] C. Theys, O. Bräysy, W. Dullaert, and B. Raa. "Using a TSP heuristic for routing order pickers in warehouses". In: *European Journal of Operational Research* 200.3 (2010), pp. 755–763.

[110] S. Trab, E. Bajic, A. Zouinkhi, M. N. Abdelkrim, H. Chekir, and R. H. Ltaief. "Product allocation planning with safety compatibility constraints in IoT-based warehouse". In: *Procedia Computer Science* 73 (2015), pp. 290–297.

[111] M. A. Trindade, P. S. Sousa, and M. R. Moreira. "Ramping up a heuristic procedure for storage location assignment problem with precedence constraints". In: *Flexible Services and Manufacturing Journal* 34.3 (2022), pp. 646–669.

[112] H. Tsubone, M. Suzuki, T. Uetake, and M. Ohba. "A Comparison between Basic Cyclic Scheduling and Variable Cyclic Scheduling in a Two-stage Hybrid Flow Shop". In: *Decision Sciences* 31.1 (2000), pp. 197–222.

[113] R Uthayakumar and S Priyan. "Pharmaceutical supply chain and inventory management strategies: Optimization for a pharmaceutical company and a hospital". In: *Operations Research for Health Care* 2.3 (2013), pp. 52–64.

[114] T. Van Gils, K. Ramaekers, K. Braekers, B. Depaire, and A. Caris. "Increasing order picking efficiency by integrating storage, batching, zone picking, and routing policy decisions". In: *International Journal of Production Economics* 197 (2018), pp. 243–261.

[115] P. van Hentenryck and R. Bent. *Online stochastic combinatorial optimization*. Cambridge: MIT Press, 2006.

[116] L. Vanbrabant, K. Braekers, K. Ramaekers, and I. Van Nieuwenhuyse. "Simulation of emergency department operations: A comprehensive review of KPIs and operational improvements". In: *Computers & Industrial Engineering* 131 (2019), pp. 356–381. ISSN: 0360-8352.

[117] J. Volland, A. Fügener, J. Schoenfelder, and J. O. Brunner. "Material logistics in hospitals: A literature review". In: *Omega* 69 (2017), pp. 82 –101. ISSN: 0305-0483.

[118] S. Voß. "The Two — Stage Hybrid — Flowshop Scheduling Problem with Sequence — Dependent Setup Times". In: *Operations Research in Production Planning and Control*. Ed. by G. Fandel, T. Gulledge, and A. Jones. Berlin, Heidelberg: Springer Berlin Heidelberg, 1993, pp. 336–352. ISBN: 978-3-642-78063-9.

[119] J. Wang, Y. F. Chen, and M. Xu. "Optimization and approximation methods for dynamic appointment scheduling with patient choices". In: *Computers & Operations Research* 92 (2018), pp. 65–76.

[120] M. Wang, R.-Q. Zhang, and K. Fan. "Improving order-picking operation through efficient storage location assignment: A new approach". In: *Computers & Industrial Engineering* 139 (2020), p. 106186.

[121] S. Wang, X. Wang, F. Chu, and J. Yu. "An energy-efficient two-stage hybrid flow shop scheduling problem in a glass production". In: *International Journal of Production Research* 58.8 (2020), pp. 2283–2314.

[122] E. N. Weiss and C. Tucker. "Queue management: Elimination, expectation, and enhancement". In: *Business Horizons* 61.5 (2018), pp. 671–678.

[123] J. Welch and N. T. Bailey. "Appointment systems in hospital outpatient departments". In: *The Lancet* 259.6718 (1952). Originally published as Volume 1, Issue 6718, pp. 1105–1108.

[124] J. Wen, N. Geng, and X. Xie. "Optimal insertion of customers with waiting time targets". In: *Computers & Operations Research* 122 (2020), p. 105001.

[125] T. Wong, M. Xu, and K. Chin. "A two-stage heuristic approach for nurse scheduling problem: A case study in an emergency department". In: *Computers & Operations Research* 51 (2014), pp. 99–110. ISSN: 0305-0548.

[126] D. Worthington. "Reflections on queue modelling from the last 50 years". In: *Journal of the Operational Research Society* 60.sup1 (2009), S83–S92.

[127] D. Worthington, M. Utley, and D. Suen. "Infinite-server queueing models of demand in healthcare: A review of applications and ideas for further work". In: *Journal of the Operational Research Society* 71.8 (2020), pp. 1145–1160.

[128] J. Xiao and L. Zheng. "Correlated storage assignment to minimize zone visits for BOM picking". In: *The International Journal of Advanced Manufacturing Technology* 61.5 (2012), pp. 797–807.

[129] X. Xu and C. Ren. "A novel storage location assignment in multi-pickers picker-to-parts systems integrating scattered storage, demand correlation, and routing adjustment". In: *Computers & Industrial Engineering* 172 (2022), p. 108618. ISSN: 0360-8352.

[130] S. Youn, H. N. Geismar, and M. Pinedo. "Planning and scheduling in healthcare for better care coordination: Current understanding, trending topics, and future opportunities". In: *Production and Operations Management* 31.12 (2022), pp. 4407–4423.

[131] J. Yu, R. Huang, and D. Lee. "Iterative algorithms for batching and scheduling to minimise the total job tardiness in two-stage hybrid flow shops". In: *International Journal of Production Research* 55.11 (2017), pp. 3266–3282.

[132] R.-Q. Zhang, M. Wang, and X. Pan. "New model of the storage location assignment problem considering demand correlation pattern". In: *Computers & Industrial Engineering* 129 (2019), pp. 210–219.

[133] I. Žulj, C. H. Glock, E. H. Grosse, and M. Schneider. "Picker routing and storage-assignment strategies for precedence-constrained order picking". In: *Computers & Industrial Engineering* 123 (2018), pp. 338–347.