European Journal of Operational Research xxx (xxxx) xxx

Contents lists available at ScienceDirect

European Journal of Operational Research

journal homepage: www.elsevier.com/locate/eor

Discrete optimization

A unified solution framework for flexible job shop scheduling problems with multiple resource constraints

Gregory A. Kasapidis [a,*], Dimitris C. Paraskevopoulos [b], Ioannis Mourtos [c], Panagiotis P. Repoussis [d]

[a] University of Liverpool Management School, University of Liverpool, Liverpool, UK
[b] Bayes Business School (formerly Cass), City St George's, University of London, London, UK
[c] Department of Management Science and Technology, School of Business Athens University of Economics and Business, Athens, Greece
[d] Department of Marketing and Communication, School of Business Athens University of Economics and Business, Athens, Greece

ARTICLE INFO

Keywords:
Scheduling
Flexible Job Shop Scheduling
Constraint Programming
Adaptive Large Neighbourhood Search
Resource constraints

ABSTRACT

This paper examines flexible job shop scheduling problems with multiple resource constraints. A unified solution framework is presented for modelling various types of non-renewable, renewable and cumulative resources, such as limited capacity machine buffers, tools, utilities and work in progress buffers. We propose a Constraint Programming (CP) model and a CP-based Adaptive Large Neighbourhood Search (ALNS-CP) algorithm. The ALNS-CP uses long-term memory structures to store information about the assignment to machines of both individual operations and pairs of operations, as encountered in high-quality and diverse solutions during the search process. This information is used to create additional constraints for the CP solver, which guide the search towards promising regions of the solution space. Numerous experiments are conducted on well-known benchmark sets to assess the performance of ALNS-CP against the current state-of-the-art. Additional experiments are conducted on new instances of various sizes to study the impact of different resource types on the makespan. The computational results show that the proposed solution framework is highly competitive, while it was able to produce 39 new best solutions on well-known problem instances of the literature.

1. Introduction

The flexible job shop scheduling problem (FJSSP) has attracted significant attention in the literature (Brucker & Schlie, 1990) due to its importance and wide applicability. The FJSSP provides a flexible base model that can be used to model industry driven (Li et al., 2020) and rich (Mokhtari & Dadgar, 2015) production scheduling problems. Nevertheless, resource constraints are missing from the archetypal FJSSP, although such constraints play a critical role when modelling real life scheduling problems. Li and Gao (2020) mention that the shop scheduling literature with resource and buffer constraints is scarce, and it is thus a promising research area.

Resources can be classified as renewable, non-renewable and cumulative. The resources that are not permanently consumed are referred to as renewable. For example, utility resources, like electricity, are consumed and renewed at any time, while a maximum consumption limit is imposed. Also, tools and special equipment can be considered as renewable resources; however, they can only be consumed as discrete units. On the other hand, non-renewable resources are permanently consumed, and they are replenished only at specific times following a given policy. Lastly, cumulative resources, such as inventory buffers, have a fixed capacity that is consumed when finished or semi-finished items are added to the storage space and replenished when items are removed.

Researchers have been studying the effect of resource constraints on scheduling problems for a long time (Błażewicz et al., 1986; Słowiński, 1980, 1981; Węglarz, 1981). One of the most prevalent problems of the literature is the resource-constrained project scheduling problem (RCPSP) that examines the consumption of several renewable resources required for processing a set of activities (Brucker et al., 1999; Debels & Vanhoucke, 2007; Patterson et al., 1990). The combination of renewable and non-renewable resources is prevalent in papers that examine the multi-mode RCPSP (Chakrabortty et al., 2020; Coelho & Vanhoucke, 2011; Elloumi et al., 2017; Muritiba et al., 2018; Van Peteghem & Vanhoucke, 2014). On the other hand, the parallel production or consumption of non-renewable resources has attracted less attention in the

* Corresponding author.
E-mail addresses: gkasapid@liverpool.ac.uk (G.A. Kasapidis), dimi@city.ac.uk (D.C. Paraskevopoulos), mourtos@aueb.gr (I. Mourtos), prepousi@aueb.gr (P.P. Repoussis).

https://doi.org/10.1016/j.ejor.2024.08.010
Received 7 November 2022; Accepted 11 August 2024
Available online 12 August 2024

RCPSP literature (van der Beek et al., 2024; Hanzálek & Šůcha, 2017), while cumulative or storage resources have also been studied (Gehring et al., 2022; Hanzálek & Šůcha, 2017). A comprehensive survey on RCPSP variants is given by Hartmann and Briskorn (2022).

The RCPSP can be considered as a generalization of various shop scheduling problems, including the FJSSP. Regarding shop scheduling problems with resource constraints, the existing body of literature is not sufficiently organized. Indicatively, one can find different names and terminologies in different papers for describing the same type of resources. The vast majority of papers consider only one specific type of resource, while (to our knowledge) there is no model or method that can take into account different types of resources. In this paper, we organize the literature of shop scheduling with various types of resource constraints and present a unified solution framework for solving the FJSSP with resource constraints.

This paper contributes to the existing body of literature in numerous ways. First, a detailed literature review is performed, the different types of resources are identified and clarified and the literature of shop scheduling problems with resource constraints is classified with respect to the resource types. Second, a Constraint Programming (CP) formulation is proposed for modelling the problem with all known types of resources; this CP model is used to derive optimal solutions and lower bounds. Third, a novel CP-based adaptive large neighbourhood search (ALNS-CP) is presented, which imposes additional constraints based on operation-to-machine and operation-pair-to-machine assignments. Fourth, a thorough experimentation on well-known benchmark data sets of the literature showed that the ALNS-CP is highly competitive compared to the current state-of-the-art, while it outperforms the off-the-shelf CP solver. Lastly, various computational experiments have been conducted on new problem instances with resource constraints in order to study the effect of various resource types on the makespan.

The remainder of the paper is organized as follows: Section 2 reviews the recent literature on shop scheduling problems. Next, Section 3 presents a CP formulation of the FJSSP with resource constraints. The proposed solution framework and all the key algorithmic components are presented in Section 4. Computational results from experiments on various benchmark data sets are presented in Section 5. We conclude in Section 6, where pointers for future research are also provided.

2. Literature review

Table 1 provides a summary of the papers from the shop scheduling literature, that consider cumulative, renewable and non-renewable resources. Regarding cumulative resources, the most prominent types are the Work-in-Progress (WIP) inventories and Limited Capacity Buffers (LCB). Both of them refer to storage capacity restrictions on the production floor. In the case of renewable resources, we consider human, equipment, tools and other resource types. The latter includes resources, such as utilities, that cannot be classified either as human, equipment or tool resources. The last column represents non-renewable resources, such as supplies and raw materials.

We have reviewed 47 papers published from 2000 to 2022. One may observe that the vast majority of papers focuses on a specific resource type, while only a handful of papers address more than one resource type. In particular, Liu et al. (2017) and Sha et al. (2021) examine both renewable and non-renewable resources, while Wong et al. (2009) and Novas and Henning (2014) study both cumulative and renewable resources. There exist no papers that consider simultaneously LCB resources with other types of resources, and the combination of non-renewable and cumulative resources.

Out of the 47 reviewed papers, 17 consider renewable resources in the form of personnel (Agnetis et al., 2014; Campos Ciro et al., 2016; Elmaraghy et al., 2000; Latorre-Núñez et al., 2016; Lei & Guo, 2014; Li et al., 2016; Zhang et al., 2021) and manufacturing equipment, such as tools, masks or fixtures (Beezão et al., 2017; Bitar et al., 2016; Chan et al., 2006; Fan et al., 2022; Figielska, 2014; Latorre-Núñez et al., 2016; Novas & Henning, 2014; Soares & Carvalho, 2020; Wong et al., 2009; Zeballos et al., 2010), while eight consider utility renewable resources (e.g., electricity) and other generic resources (Fanjul-Peyro et al., 2017; Figielska, 2018; Fleszar & Hindi, 2018; Li et al., 2019; Liu et al., 2017; Nguyen et al., 2019; Sha et al., 2021; Waldherr & Knust, 2017). Cumulative resources are mostly encountered in the form of LCB buffers (Brucker et al., 2006; Liu et al., 2018; Trabelsi et al., 2012; Yaurima et al., 2009). In this category, we also include papers that address the so-called Blocking Job Shop Scheduling Problem (BJSSP) (Dabah et al., 2019; Mascis & Pacciarelli, 2002; Meloni et al., 2022; Mogali, Barbulescu, & Smith, 2021; Mogali, Smith, & Rubinstein, 2021) as a special case of LCB resources with zero capacity. Papers that consider non-renewable resources, mostly include contributions that focus on the single machine scheduling problem (Grigoriev et al., 2005; Györgyi & Kis, 2017). In most cases, non-renewable resources refer to specific supply and consumption rates (Györgyi & Kis, 2019; Hashimoto & Mizuno, 2021).

Fig. 1(a) provides an overview of the literature regarding the most prevalent solution methodologies. Specifically, 26 out of 47 papers present exact solution approaches. The majority of these papers are MIP-based models, while very few present CP-based algorithms (Novas, 2019; Novas & Henning, 2014; Zeballos et al., 2010). There exist some approximation algorithms (Györgyi & Kis, 2017, 2018; Hashimoto & Mizuno, 2021) and Column Generation solution methods (Figielska, 2018). Regarding heuristic methodologies, 11 papers present heuristic solution frameworks that consider problem-specific information (Andrade-Pineda et al., 2020; Azzi et al., 2012; Herr & Goel, 2016; Mascis & Pacciarelli, 2002; Trabelsi et al., 2012; Waldherr & Knust, 2017) as well as dispatching rules (Elmaraghy et al., 2000). Regarding meta-heuristic algorithms, the majority of papers present genetic algorithms (GA) (Chan et al., 2006; Elmaraghy et al., 2000; Latorre-Núñez et al., 2016; Wong et al., 2009) and other GA-based solution frameworks (Fan et al., 2022; Li et al., 2016; Soares & Carvalho, 2020; Zhang et al., 2021). Tabu search is another popular solution method that can be found in Gröflin et al. (2011), Aschauer et al. (2017, 2018) and Dabah et al. (2019). Lastly, there is a small amount of papers that present other types of meta-heuristics, such as particle swarm optimization (Wong et al., 2009), ant colony optimization (Belaid et al., 2012; Campos Ciro et al., 2016), local search algorithms (Bitar et al., 2016; Boufellouh & Belkaid, 2020; Mogali, Barbulescu, & Smith, 2021; Pranzo & Pacciarelli, 2016) as well as large neighbourhood search (Beezão et al., 2017).

Fig. 1(b) provides an overview of the popular problem variants. The Job Shop Scheduling Problem (JSSP), the Flow Shop Scheduling Problem (FSSP) as well as the FJSSP, are the most popular problem variants. There exists only a handful of papers that study the original JSSP (Agnetis et al., 2014; Brucker et al., 2006). On the other hand, the Dual-Resource Constrained Job Shop Scheduling Problem (DRCJSSP) and the Blocking Job Shop Scheduling Problem (BJSSP) are the most widely used variants. More specifically, the BJSSP is mostly used to study cumulative LCB resources with zero capacity (Dabah et al., 2016, 2019; Mogali, Barbulescu, & Smith, 2021; Oddi et al., 2012), while the DRCJSSP considers only human resources (Elmaraghy et al., 2000; Lei & Guo, 2014; Li et al., 2016). The FSSP (Belaid et al., 2012; Figielska, 2014, 2018; Trabelsi et al., 2012; Waldherr & Knust, 2017) is also popular, since it can be used to model assembly line environments with resource constraints. Furthermore, other FSSP variants like the Hybrid Flow Shop Scheduling Problem (Azzi et al., 2012; Latorre-Núñez et al., 2016; Yaurima et al., 2009) and the Permutation Flow Shop Scheduling Problem (Boufellouh & Belkaid, 2020) have attracted significant research interest. The generalized FJSSP (Brucker & Schlie, 1990) comes third (Aschauer et al., 2017, 2018; Chan et al., 2006; Liu et al., 2017; Novas, 2019). Next, the so-called Flexible Manufacturing System (FMS) problem is often used to study renewable resources (Soares & Carvalho, 2020; Zeballos et al., 2010), while the

Table 1
Literature overview.

	Cumulative		Renewable			Non-renewable
	WIP	LCB	Human	Equipment	Other	
Elmaraghy et al. (2000)	–	–	✓	–	–	–
Mascis and Pacciarelli (2002)	–	✓	–	–	–	–
Grigoriev et al. (2005)	–	–	–	–	–	✓
Brucker et al. (2006)	–	✓	–	–	–	–
Chan et al. (2006)	–	–	–	✓	–	–
Wong et al. (2009)	✓	–	–	✓	–	–
Yaurima et al. (2009)	–	✓	–	–	–	–
Zeballos et al. (2010)	–	–	–	✓	–	–
Gröflin et al. (2011)	–	✓	–	–	–	–
Azzi et al. (2012)	✓	–	–	–	–	–
Trabelsi et al. (2012)	–	✓	–	–	–	–
Belaid et al. (2012)	✓	–	–	–	–	–
Novas and Henning (2014)	✓	–	–	✓	–	–
Lei and Guo (2014)	–	–	✓	–	–	–
Agnetis et al. (2014)	–	–	✓	–	–	–
Figielska (2014)	–	–	–	✓	–	–
Bitar et al. (2016)	–	–	–	✓	–	–
Latorre-Núñez et al. (2016)	–	–	✓	✓	–	–
Li et al. (2016)	–	–	✓	–	–	–
Campos Ciro et al. (2016)	–	–	✓	–	–	–
Herr and Goel (2016)	–	–	–	–	–	✓
Aschauer et al. (2017)	–	✓	–	–	–	–
Beezão et al. (2017)	–	–	–	✓	–	–
Waldherr and Knust (2017)	–	–	–	–	✓	–
Fanjul-Peyro et al. (2017)	–	–	–	–	✓	–
Liu et al. (2017)	–	–	–	–	✓	✓
Györgyi and Kis (2017)	–	–	–	–	–	✓
Liu et al. (2018)	–	✓	–	–	–	–
Aschauer et al. (2018)	–	✓	–	–	–	–
Fleszar and Hindi (2018)	–	–	–	–	✓	–
Figielska (2018)	–	–	–	–	✓	–
Györgyi and Kis (2018)	–	–	–	–	–	✓
Novas (2019)	✓	–	–	–	–	–
Dabah et al. (2019)	–	✓	–	–	–	–
Li et al. (2019)	–	–	–	–	✓	–
Nguyen et al. (2019)	–	–	–	–	✓	–
Györgyi and Kis (2019)	–	–	–	–	–	✓
Soares and Carvalho (2020)	–	–	–	✓	–	–
Boufellouh and Belkaid (2020)	–	–	–	–	–	✓
Andrade-Pineda et al. (2020)	–	–	–	–	–	✓
Mogali, Barbulescu, and Smith (2021)	–	✓	–	–	–	–
Mogali, Smith, and Rubinstein (2021)	–	✓	–	–	–	–
Zhang et al. (2021)	–	–	✓	–	–	–
Hashimoto and Mizuno (2021)	–	–	–	–	–	✓
Sha et al. (2021)	–	–	–	–	✓	✓
Fan et al. (2022)	–	–	–	✓	–	–
Meloni et al. (2022)	–	✓	–	–	–	–

Unrelated Parallel Machine Scheduling (UPM) problem serves as a base model to study renewable resources where operations have different resource requirements depending on the machine that they are assigned to Fanjul-Peyro et al. (2017), Fleszar and Hindi (2018), Li et al. (2019). Lastly, the Single-machine Scheduling (SSP) problem has also received some attention (Györgyi & Kis, 2017, 2018; Hashimoto & Mizuno, 2021; Herr & Goel, 2016) and has been used as the test bed to examine the properties of non-renewable resources.

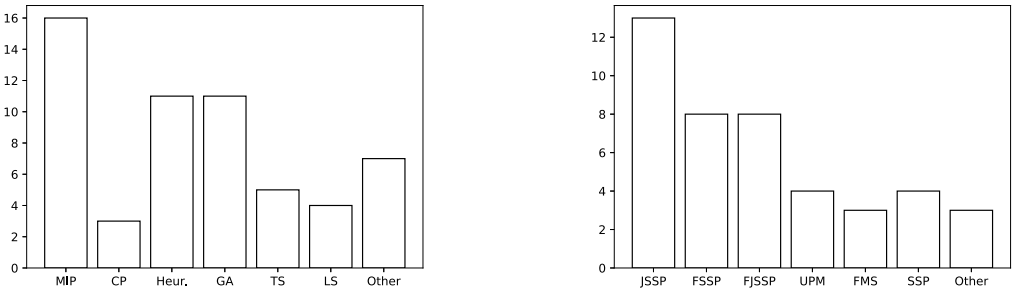
In terms of objective functions, 34 out of 47 papers consider the makespan as the primary objective. Despite the addition of resource constraints, the main concern remains to minimize the completion time of the production schedule. Some papers consider the maximum lateness and tardiness and these objectives are encountered when due and release dates of jobs are present (Grigoriev et al., 2005; Györgyi & Kis, 2017; Herr & Goel, 2016; Nguyen et al., 2019). Alternative objectives include the carbon footprint minimization (Liu et al., 2017), the machine idle cost (Chan et al., 2006), the maintenance cost (Boufellouh & Belkaid, 2020) and also the minimization of the amount of tools used (Zeballos et al., 2010). Lastly, we also notice that 36 out of 47 papers consider only a single objective, while only 11 papers consider multiple objectives.

3. Modelling framework

3.1. Notation

This section presents the CP model of the FJSSP with multiple types of resource constraints. For consistency, we adopt the notation used by Kasapidis et al. (2021) for the FJSSP with arbitrary precedence graphs. Let $J = \{1, \dots, l\}$ denote the set of jobs, $M = \{1, \dots, m\}$ denote the set of machines, and $\Omega = \{1, \dots, n\}$ denote the set of all the operations. The set M_i denotes the set of available machines, where an operation i can be processed. Additionally, let ω_i^- and ω_i^+ denote the single job predecessor and successor operations of an operation i (if any). For completeness, let $\Omega_i^- = \{\omega_i^-\}$ and $\Omega_i^+ = \{\omega_i^+\}$ denote two sets that include the predecessor and successor operation of an operation i, respectively. If an operation i has no job predecessor and/or successor operations, then $\Omega_i^- = \emptyset$ and/or $\Omega_i^+ = \emptyset$, respectively. The flexibility f of the problem describes the average number of machines available per operation of the problem and can be calculated as: $f = \frac{1}{n}\sum_{i=1}^{n} |M_i|$. Lastly, symbols i_j° and i_j^* are used to denote the first and the last operations of a job $j \in J$, respectively.

The above notation is extended by introducing new nomenclature for modelling renewable, non-renewable and cumulative resources. Let



(a) Solution methods (b) Popular problem variants

Fig. 1. Overview of the shop scheduling literature with resource constraints.

$R = \{1, \ldots, L_R\}$ denote a set of arbitrary non-renewable resources, with zero initial inventory, and let $T = \{1, \ldots, L_T\}$ be a set of renewable tool resources that may be required during the processing of an operation. The number of available instances of a tool $r \in T$ is denoted by $\bar{T}_r$. This means that no more than $\bar{T}_r$ instances of r can be used simultaneously for the entire duration of the schedule. Let $T_i \subseteq T$ denote the set of required tools for an operation i. Also, let $U = \{1, \ldots, L_U\}$ be a set of renewable utility resources. This type of resource is consumed by a machine $k \in M$ for processing an operation i (e.g., consumption of a utility). Similarly, each resource $r \in U$ has a hard consumption limit denoted by $\bar{U}_r$. The requirements of a resource $r \in U$ for the processing of an operation $i \in \Omega$ on a machine $k \in M_i$, is denoted by $u_{i,k,r}$. Note that, $u_{i,k,r} \geq 0 \quad \forall r \in U, \forall i \in \Omega, \forall k \in M_i$. Given the fact that the tool and utility resources are both renewable, tools could be modelled using the formulation of utility resources with $u_{i,k,r} = 1, \forall k \in M_i$. Nevertheless, for the sake of simplicity, we choose to keep their representation separate.

Let $W = \{1, \ldots, L_W\}$ be a set of cumulative resources in the form of WIP buffers. The main motivation behind this resource type is that it can be used to represent work-in-progress buffers that store non-renewable resources. In particular, for each non-renewable resource $r \in R$, the associated cumulative resource is denoted by W_r, while the maximum capacity of a cumulative resource $z \in W$ is denoted by $\bar{W}_z$. Also, every machine is equipped with a limited capacity buffer. This is used to temporarily "hold" operations that have been processed on the corresponding machine, but cannot leave the machine, until the processing of their job successor has started. In other words, the job operation currently completed at a given machine can proceed when the machine of their job successor becomes available, and this is the so-called blocking effect. The size of the limited capacity buffer of a machine $k \in M$ is denoted by y_k.

Every job $j \in J$ can be associated with two sets of resources $R_j^-, R_j^+ \subseteq R$ that correspond to the resources required to initiate the processing of a job, and produced once the job processing finishes. Note that, there cannot exist a resource that is both required and produced by a job, i.e., $R_j^- \cap R_j^+ = \emptyset, \forall j \in J$. Nevertheless, there can exist jobs $j \in J$ that do not require or produce resources, i.e., $R_j^+ = \emptyset \wedge R_j^- = \emptyset$. The amount of a resource $r \in R_j^- \cup R_j^+$ required or produced by a job j is denoted by $Q_{j,r}$. The required job resources are consumed when the first operation of a job j, i_j° starts being processed, while the resources are produced when the last operation of the job i_j^* finishes being processed. A Table with the symbols and their corresponding descriptions can be found on Appendix A of the online companion.

3.2. A motivating example

To demonstrate the effect of the resource constraints on the makespan, we present a small example of an FJSSP problem with resource constraints. Let us consider a problem with $l = 4$, $n = 5$, $m = 3$, $L_T = 2$ and $L_U = 1$. Also, $|M_i| = 1$, $\forall i \in \Omega$, which means that every operation can be assigned to exactly one machine, and $|T_i| = 1$, $\forall i \in \Omega$, which means that every operation requires a single tool. The information of the problem is presented in Table 2. We also assume that the limited capacity buffer of every machine $k \in M$ has zero capacity, i.e. $y_k = 0$, and that the available instances for the tool resources are $\bar{T}_1 = 3$ and $\bar{T}_2 = 1$.

Fig. 2(a) presents the Gantt chart of an optimal schedule and the consumption of U_1 when no resource constraints are imposed. The value of the objective C_{max} is 15, while the maximum consumption of U_1 is 6 units. If a tighter consumption limit is considered for U_1, i.e., $\bar{U}_1 = 4$, the operations $OP(2)$ and $OP(3)$ have to be executed after operation $OP(5)$, so that the resource consumption limit is respected. This results in an increased value of the objective by 4 units, i.e. $C_{max} = 19$ (see Fig. 2(b)). Furthermore, if limited capacity buffers of zero capacity are also considered, some operations are prevented from starting right after their machine predecessors. As shown in Fig. 2(c), $OP(4)$ is forced to start five time units later, increasing the makespan to $C_{max} = 20$. Lastly, if tool resources are also present, the execution of $OP(4)$ (and subsequently the makespan) is further shifted by 3 units, since operations $OP(2)$ and $OP(4)$ require the same tool available in one instance only, as shown in Fig. 2(d). Evidently, even for very small problems, resource constraints may cause a significant increase to the makespan, and the impact is even more significant when combinations of resource constraints are considered.

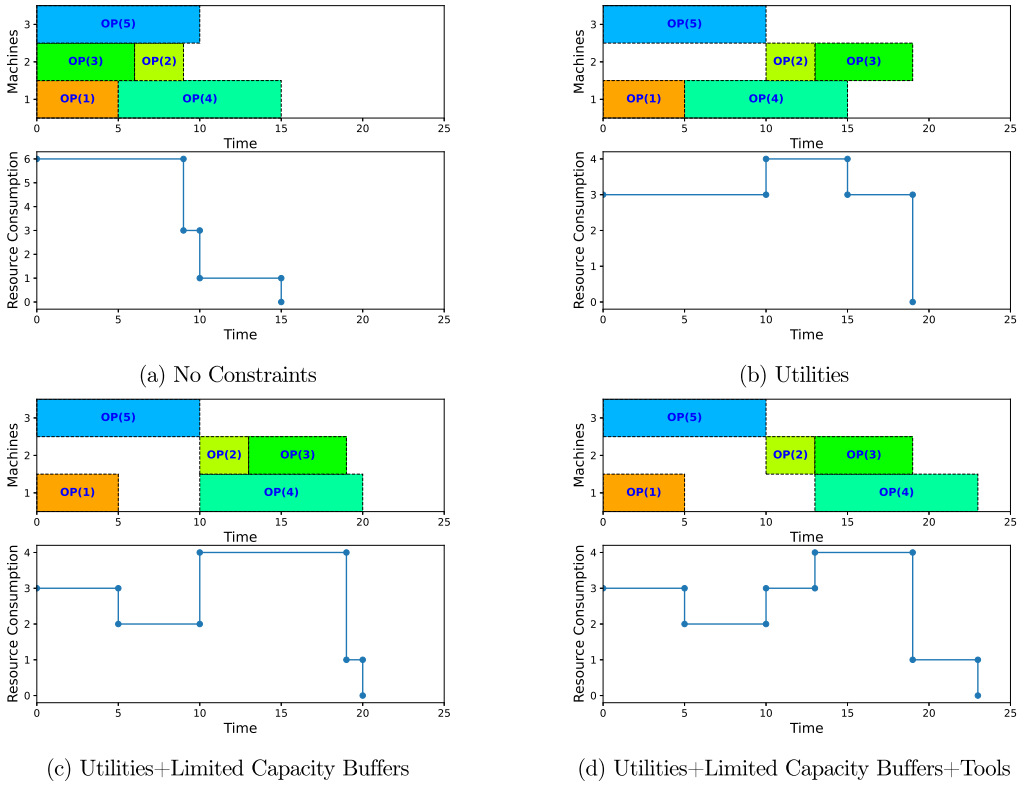
3.3. A constraint programming formulation

The literature has shown that CP is particularly effective for solving not only variants of the FJSSP (Kasapidis et al., 2021; Latorre-Núñez et al., 2016) but also variants of the RCPSP which can be considered as a generalization of the FJSSP (Gómez Sánchez et al., 2023). CP methods use global constraints, which encapsulate sets of simpler constraints. These constraints enable the design of simpler and more compact formulations, while they are usually connected to efficient filtering algorithms that can aid the domain filtering process and the constraint propagation. Specifically for scheduling problems with resource constraints, there exist the 'cumulative' (Aggoun & Beldiceanu, 1993) and the 'disjunctive' (Carlier, 1982) global constraints. The common basis between our CP-model and the majority of CP methodologies that have been developed for FJSSP variants with resource constraints and the RCPSP is the set of these global constraints.

The effectiveness of the latest CP frameworks on production scheduling applications is highlighted in the detailed literature review of Prata et al. (2024), who show that the majority of the publications study variants of shop-scheduling problems and more generic scheduling problems with resource constraints, like the RCPSP. In particular, Novas (2019) developed a CP model to solve a complex resource constrained FJSSP with lot/sub-lot streaming, no-wait constraints and a shared WIP buffer. The proposed methodology was adopted and adapted by Zeballos et al. (2010) for solving a resource

Table 2
Example instance of a FJSSP with resource constraints.

Operation	Job	Predecessor	Machine	Processing time	Utility consumption	Tool
1	1	–	1	5	1	1
2	1	1	2	3	3	2
3	2	–	2	6	3	1
4	3	–	1	10	1	2
5	4	–	3	10	1	1



(a) No Constraints (b) Utilities

(c) Utilities+Limited Capacity Buffers (d) Utilities+Limited Capacity Buffers+Tools

Fig. 2. Gantt charts and resource consumption plots for different resource constraint combinations.

constrained Flexible Manufacturing System problem that focuses on the allocation of production jobs to machines and the allocation of tools to the available workstations. With regards to the RCPSP, researchers have been using these global constraints to develop models for more generalized scheduling problems. More specifically, Trojet et al. (2011) propose a CP model based on the cumulative constraint to solve an RCPSP with additional energy consumption constraints. Kreter et al. (2018, 2017) propose new CP models that incorporate novel constraint propagators for the RCPSP and the resource availability cost problem (RACP) with calendar constraints, respectively. The authors conducted extensive computational experiments that showed how lazy clause generation combined with tailor-made constraint propagators can be used to significantly improve the performance of the CP, compared not only to other state-of-the-art CP solvers, but also to MILP formulations.

In this paper, we adopt the IBM CP Optimizer nomenclature. The IBM CP Optimizer employs the so-called "interval" variables that can be used to represent a task or an activity. The interval variables are defined by three integer attributes, a starting time, an ending time and a duration of the interval. In the context of CP solvers, the interval variables may be defined as optional variables, i.e., they may or may not be present in a feasible solution of a problem. According to Laborie et al. (2018), the domain of an interval variable a is defined as: $dom(a) = \{\perp\} \cup \{[s, e) : s, e \in \mathbb{Z}, s \leq e\}$, where s, e are the earliest start and latest completion times of a, respectively. The symbol $\perp$, is used to denote the truth value "False", i.e., $\perp$ signifies a proposition that cannot be true under any circumstances. The CP Optimizer provides a large set of function expressions, constraint expressions and cumulative function expressions that can be applied on interval variables. This rich framework lends itself well to model complex scheduling problems with a wide variety of constraints. More specifically, function expressions can be used to access information regarding interval variables during the search, constraint expressions can be used to construct constraints that involve designated model variables by utilizing function expressions, while cumulative function expressions can be used to describe the contribution of interval variables to discrete linear functions over time. We also refer the reader to Novas (2019) and Kasapidis et al. (2021) for a more comprehensive description of interval and sequence interval variables of the CP Optimizer.

During the search process, the CP optimizer progressively reduces the domains of all the variables of the problem. When the domain of an interval variable a becomes a singleton set, i.e., $|dom(a)| = 1$, the variable is labelled as "fixed" and is denoted by $\underline{a}$. Subsequently, if the domain of all the decision interval variables is reduced to a singleton set, a valid solution for the problem has been found. At this point, an interval variable $a : \perp \cap dom(\underline{a}) = \emptyset$, is also labelled as "present" in the solution, while a variable $a : dom(\underline{a}) = \{\perp\}$ is labelled as "absent". Absent variables are not considered by any constraint expression of the problem as if they were never defined. On the other hand, when a fixed interval variable a is present, several function expressions can be used to calculate the attributes associated with the interval variable. The function expressions that are used in this paper are presented below:

i. $PresenceOf(a)$ defines a function expression that returns true when the interval variable a is present in the incumbent solution of the CP Optimizer, and false otherwise.
ii. $EndOf(a)$ defines a function expression that returns the completion time of the interval variable a in the incumbent solution of the CP Optimizer if a is present.
iii. $StartOf(a)$ defines a function expression that returns the start time of the interval variable a in the incumbent solution of the CP Optimizer if a is present.

Moreover, the following constraint expressions are used to define constraints that involve interval and/or sequence interval variables, i.e., ordered lists of interval variables:

i. $EndBeforeStart(a, b)$ denotes a constraint between two interval variables a, b that ensures the start time of variable b will be greater or equal to the end time of variable a.
ii. $Alternative(a, B)$ denotes a constraint that sets the domain of interval variable a to the set B, i.e., if a is present in the incumbent solution, then exactly one of the interval variables $b \in B$ is also present in the incumbent solution. Also, the attributes (start time, completion time and length) of a and b will be identical.
iii. $NoOverlap(\sigma)$ denotes a constraint on a set of variables σ that ensures there are no overlapping time windows between any interval variables included in σ.

Discrete linear functions in the IBM CP Optimizer are modelled using the notion of step-wise functions, which are special cases of piece-wise linear functions. A piece-wise linear function is a function comprised of a number of segments, so that the function per segment is linear (Leenaerts & Bokhoven, 1998). A step-wise function is a special case of a piece-wise linear function, where the derivative of the function per segment is equal to zero, i.e., the function has a constant value per segment. Given that, the IBM CP Optimizer offers the so-called cumulative function expressions that can be used to define step-wise functions over the time-span of interval variables. Moreover, using constraint expressions, the values of these functions can be constrained as required. In this paper, we use the following expressions to model the consumption and production of resources:

i. $Pulse(a, h)$ defines a cumulative function whose value is equal to h during the execution time window of an interval variable a and 0 otherwise, if a is present in the solution.
ii. $StepAtStart(a, h)$ defines a cumulative function whose value is equal to h at the start of the interval variable a, if a is present in the solution.
iii. $StepAtEnd(a, h)$ defines a cumulative function whose value is equal to h after the end of the interval variable a, if a is present in the solution.
iv. $AlwaysIn(E, a, v_{min}, v_{max})$ denotes a constraint that limits the values of the cumulative function expression E within the domain $[v_{min}, v_{max}]$ for the time interval defined by the interval variable a, if a is present in the solution.

Given the above, the examined problem can be modelled as follows. For every operation $i \in \Omega$, a decision interval variable τ_i is defined. In addition, a decision interval variable $\phi_{i,k}$ is defined for every machine $k \in M_i$, with its length set to $\eta_{i,k}$, where $\eta_{i,k}$ is used to denote the processing time of operation i on the machine k. In order to calculate the waiting times of operations in limited capacity buffers, for every operation $i \in \Omega$ and a corresponding machine $k \in M_i$, we define the decision variable $\phi^b_{i,k}$. The length of the decision interval variable $\phi^b_{i,k}$ is not fixed, since the waiting time of any operation on its machine is not known beforehand. We also define a set $\mu_i = \{\phi_{i,k}, \forall k \in M_i\}$ to represent all the available execution modes per operation i, i.e., all the different machines where an operation i can be processed. The set μ_i is also used to denote the domain set of variable τ_i. Lastly, a sequence interval decision variable σ_k is defined per machine k for the set of interval variables $\sigma_k = \{\phi_{i,k}, \forall i \in \Omega\}$. The cumulative function that accumulates the usage of the limited capacity buffer of a machine k is denoted by Y_k. The CP formulation is given below:

$$\text{minimize} C_{max} \tag{1}$$

subject to the conditions given by Eqs. (2) to (14) in Box I.

The objective (1) refers to the minimization of the makespan. Constraints (2) are used to enforce a unique selection of the available modes for the interval variable τ_i out of the set μ_i. Constraints (3) are used to cover the precedence relationships, i.e., each operation i can start as soon as its single job predecessor $\omega_i^- \in \Omega_i^-$ (if any) has finished. Constraints (4) ensure that the interval variables included in σ_k do not overlap, since a machine can execute only one operation at a time. They also ensure that each operation starts after its machine predecessor has finished. Constraints (5) and (6) are used to accumulate the consumption of utility and tool resources, respectively. They also make sure that the utility resource consumption limits as well as the available tool instances are respected. Constraints (7)–(11) are used to describe the usage of limited capacity buffers. In more detail, Constraints (7) impose a waiting time on a buffer that belongs to the machine where the operation was processed. Constraints (8) accumulate the usage of the machine waiting buffers, while Constraints (9) make sure that processing on a machine can start only when the corresponding buffer is not full. Constraints (10) and (11) define the duration of the waiting of an activity to the machine buffer. More specifically, Constraints (10) ensure that waiting on the buffer starts immediately after processing on the machine finishes, while Constraints (11) ensure that the activity is removed from the buffer, when the job successor of an operation starts being processed. Constraints (12) are used to accumulate the production and consumption of each generalized resource, while Constraints (13) accumulate the usage of resources on their corresponding work in progress buffer. Lastly, Constraint (14) is responsible for the calculation of the makespan.

4. The adaptive large neighbourhood search-constraint programming method

Large neighbourhood search (LNS) is a meta-heuristic algorithm that was first introduced by Shaw (1998), which is similar to the ruin and re-create mechanism proposed by Schrimpf et al. (2000). More specifically, at each iteration, a part of the solution is destroyed and then repaired using destruction and repair operators, respectively. The adaptive LNS is an extension of LNS first introduced by Ropke and Pisinger (2006). ALNS uses multiple destruction and repair operators, which are favoured based on their performance during the search. The literature has shown that ALNS is effective in solving a wide variety of problems like vehicle routing (Hellsten et al., 2020; Kovacs et al., 2012) and shop scheduling (Rifai et al., 2016; Song et al., 2022). We refer the reader to Pisinger and Ropke (2007) for a more comprehensive description of the ALNS.

There exists literature on solution methodologies that combine ALNS and CP frameworks. Palpant et al. (2004) developed an LNS-CP algorithm for the Resource Constrained Project Scheduling Problem (RCPSP), which decomposes the main problem into subproblems and subsequently calls a CP solver to solve the sub-problems and compose a complete solution. Hojabri et al. (2018) developed a similar scheme for the VRP with multiple synchronization constraints. Their CP-based ALNS uses well-established destruction operators from the VRP literature, while a CP model is used to solve the main problem. Sacramento et al. (2020) propose a more generic math-heuristic ALNS for solving the Port Scheduling Problem. The authors used a CP solver as a search intensification mechanism to further improve local optima produced by a local search heuristic. Recently, similar hybrid algorithms have been proposed for shop scheduling problems. Abreu

$$Alternative(\tau_i, \mu_i) \quad \forall i \in \Omega \quad (2)$$

$$StartOf(\tau_i) \geq EndOf(\tau_j) \quad \forall i \in \Omega, \forall j \in \Omega_i^- \quad (3)$$

$$NoOverlap(\sigma_k) \quad \forall k \in M \quad (4)$$

$$\sum_{i=1}^{n} \sum_{k \in M_i} Pulse(\phi_{i,k}, u_{i,k,r}) \leq \bar{U}_r \quad \forall r \in U \quad (5)$$

$$\sum_{i=1 | r \in T_i}^{n} Pulse(\tau_i, 1) \leq \bar{T}_r \quad \forall r \in T \quad (6)$$

$$PresenceOf(\phi_{i,k}^b) = PresenceOf(\phi_{i,k}) \quad \forall i \in \Omega, \forall k \in M_i \quad (7)$$

$$Y_k = \sum_{i=1}^{n} Pulse(\phi_{i,k}^b, 1) \quad \forall k \in M, k \in M_i \quad (8)$$

$$AlwaysIn(Y_k, \phi_{i,k}, 0, y_k) \quad \forall i \in \Omega, \forall k \in M_i \quad (9)$$

$$StartOf(\phi_{i,k}^b) = EndOf(\phi_{i,k}) \quad \forall i \in \Omega, \forall k \in M_i \quad (10)$$

$$EndOf(\phi_{i,k}^b) = StartOf(\tau_j) \quad \forall i \in \Omega, \forall j \in \Omega_i^+, \forall k \in M_i \quad (11)$$

$$\sum_{j \in J | r \in R_j^+} StepAtEnd(\tau_{i_j^*}, Q_{j,r}) - \sum_{j \in J | r \in R_j^-} StepAtStart(\tau_{i_j^\circ}, Q_{j,r}) \geq 0 \quad \forall r \in R \quad (12)$$

$$\sum_{r \in R | \bar{W}_r = z} \left\{ \sum_{j \in J | r \in R_j^+} StepAtEnd(\tau_{i_j^*}, Q_{j,r}) - \sum_{j \in J | r \in R_j^-} StepAtStart(\tau_{i_j^\circ}, Q_{j,r}) \right\} \leq \bar{W}_z \quad \forall z \in W \quad (13)$$

$$C_{max} \geq EndOf(\tau_i) \quad \forall i \in \Omega \quad (14)$$

Box I.

and Nagano (2022) proposed an ALNS-CP algorithm for a complex variant of the open-shop scheduling problem that incorporates non-anticipatory sequence-dependent setup times. The authors highlight the efficiency of the hybrid scheme, especially when large scale problem instances are considered. Lastly, Yunusoglu and Yildiz (2023) studied a variant of the FJSSP with lot streaming, setup and transport resource constraints. A CP-based LNS algorithm was proposed that uses a single destruction operator to remove a set of variables from the solution. The authors proposed a custom branching strategy to guide the CP towards the near-optimal repair of the partial solution. One of the major differences of our ALNS-CP is that we use information gathered from the search history to extract useful constraints that are fed into the CP model. These constraints are used to purposefully fix part of the solution, that belongs to high quality solutions encountered during the search, while re-solving the remaining part of the solution by calling the CP solver. We describe, in the following of the paper, the reasons that make this scheme particularly efficient in solving job shop scheduling problems with resource constraints.

The aim of the ALNS-CP is to solve efficiently a wide variety of FJSSP variants with resource constraints. In particular, the ALNS-CP uses well-known destruction operators that are effective for a wide range of scheduling problems, according to the relevant literature. In addition, we propose new operators that use the search history to guide the search to promising regions of the solution space. Traditionally, the destruction operators in ALNS are used to remove some elements from a solution, or in other words to destruct a part of a solution (Ropke & Pisinger, 2006), and subsequently heuristic procedures are used to repair that semi-complete solution. To the contrary, in our ALNS-CP, the destruction operators produce a set of extra constraints to preserve part of the solution, while at the same time reconstruct the remaining unfixed part of the solution by calling the CP solver. Based on this, instead of destruction operators we use the term constraint extraction operators throughout the paper, as it is more appropriate. It is worth mentioning that the proposed scheme can easily be adapted to solve even more enhanced shop scheduling variants by simply altering the underlying CP model.

Algorithm 1 provides an overview of the proposed ALNS-CP. First, the solution sets $\mathcal{R}^E, \mathcal{R}^T$ are initialized. The set $\mathcal{R}^E$ stores the 'elite' solutions encountered during the search, while $\mathcal{R}^T$ is used to store the solutions of the current search trajectory. Then, an initial solution s is obtained by the CP solver using the $CPSolveFull$ meta-method. In general, the CP solver is initialized using three arguments, (a) a start solution, (b) a set of constraints to satisfy, and (c) a time limit. In this case, no starting solution is used, the default set of constraints C (Constraints (1)—(14)) is provided, while a self-tuned parameter θ is used as the time limit. The main loop of the algorithm consists of four phases: (a) the operator selection, (b) the constraint generation, (c) the solution reconstruction and (d) the improvement phase.

At the operator selection phase, a constraint extraction operator p is selected from the group of available operators $\mathcal{P}$. Note that, the selection is done in a roulette wheel fashion, which favours the operators that perform better at each iteration (see Section 4.3 for more details). The framework uses seven constraint extraction operators that are presented in Section 4.1. At the constraint generation phase, a set of constraints $\acute{C}$ is generated as a result of the application of p. Every constraint $c \in \acute{C}$ refers to an operation-to-machine assignment or an operation-to-operation relationship that could be fixed or neglected in the current iteration of the algorithm. The additional constraints significantly narrow the solution space and as a result the CP Solver can find an optimal solution, given the remaining non-fixed variables, in short computational times. The set of constraints $\acute{C}$ is appended with additional constraints that are created by using the solutions of $\mathcal{R}^T$. These constraints ensure that every time the CP solver is called, the search typically avoids revisiting solutions that have been encountered in the past. This rationale is inspired by the Tabu Search principle, introduced by Glover and Laguna (1997) according to which solution elements (or solution structures) are stored in a "tabu" list (our trajectory list) so that the local search algorithm (our CP solver) avoids

revisiting them during the search. This process is described in more detail in Section 4.2.2.

At the reconstruction phase, the CP solver solves the problem using the $CPSolveSequential$ meta-method. This method is used to denote that the CP-solver iteratively explores the search space for improved solutions. The search continues until no further improvement can be found within the specified time limit. Upon completion, the method returns a set of solutions S that includes all the encountered solutions during the search. This time the CP solver is initialized with solution s as the starting point of the search, the combined set of constraints $C \cup \acute{C}$, and θ as the time limit per iteration of the solver. Next, every solution $s \in S$ is inserted into $\mathcal{R}^E$ as long as it satisfies specific criteria as described in Section 4.2. Note that the size of the population $\mathcal{R}^E$ is controlled by the parameter N^E. Lastly, s is replaced by the best solution of S in terms of the makespan.

The improvement phase focuses on further improving the quality of the solution s obtained earlier if $C^s_{max} < C^{\hat{s}}_{max}$, where $\hat{s}$ denotes the best solution found so far in terms of the makespan. The aim is to intensify the search in the region of the solution space around the new best solution s. For this purpose, the CP solver is called by using the following parameters: s as the starting solution, the original set of constraints C and the time limit θ. Next, the solution s is inserted to $\mathcal{R}^E$ and, $\mathcal{R}^T$ if possible, the adaptive parameters ρ and θ are updated (see Section 5.1 for more details) and the framework continues to the next iteration. Lastly, the overall solution framework terminates when no improvement to the best solution found is observed for a number of G iterations.

Algorithm 1 ALNS - CP

Require: G, N^E, N^T, $\mathcal{P}$
$\mathcal{R}^E \leftarrow \emptyset$, $\mathcal{R}^T \leftarrow \emptyset$, $g \leftarrow 0$, $\hat{s} \leftarrow \emptyset$, $\acute{C} \leftarrow \emptyset$
s = CPSolveFull($\emptyset$, C, θ)
while $g < G$ **do**
  p = SelectOperator($\mathcal{P}$) ▷ 1. Operator Selection
  $\acute{C} = p(s, \rho, \mathcal{R}^T)$ ▷ 2. Constraint Extraction
  $S \leftarrow$ CPSolveSequential(s, $\acute{C} \cup C$, θ) ▷ 3. Solution Reconstruction
  for each $\acute{s} \in S$ **do**
    AddSolutionToPool($\acute{s}$, $\mathcal{R}^E$)
  end for
  $s = \underset{\forall \acute{s} \in S}{\operatorname{argmin}} C^s_{max}$
  if $C^s_{max} < C^{\hat{s}}_{max}$ **then** ▷ 4. Improvement Phase
    $s \leftarrow$ CPSolveFull(s, C, θ)
    $g \leftarrow 0$, $\mathcal{R}^T \leftarrow 0$, $\hat{s} \leftarrow s$
  end if
  UpdateAdaptiveParameters(ρ, θ)
  AddSolutionToPool(s, $\mathcal{R}^T$)
  AddSolutionToPool(s, $\mathcal{R}^E$)
  $g \leftarrow g + 1$
end while
return $\hat{s}$

4.1. The constraint extraction operators

A core component of our ALNS-CP is the set of the constraint extraction operators. The basic version of ALNS removes some elements or structures of the solution by using destruction operators, and subsequently re-inserts the removed elements back into the solution by using repair operators. In our proposed ALNS-CP, the constraint extraction operators play the role of the destruction operators. At each iteration, a set of operations $D \subset \Omega$ is selected. Instead of removing these operations and trying to re-insert them with a heuristic mechanism, we form a set of constraints $\acute{C}$ that is appended to the original set of constraints C. In this way, we preserve a part of the solution, while "destructing" and "rebuilding" the remaining part. The number of operations selected by every operator is equal to ρn, where $0 < \rho \leq 1$ is a self-tuned parameter. In the following, we describe in detail the proposed constraint extraction operators.

4.1.1. The time window operator

Given the current solution s, the time window operator selects the operations that are being processed within a time window $[w_{start}, w_{end}]$, and stores them in a set D. Subsequently, the machine assignments and the machine precedence relationships of the remaining operations are fixed. The size of the time window is calculated based on the number of selected operations ρn. In particular, the operator assumes that every operation $i \in \Omega$, has a contribution of $\frac{C^s_{max}}{n}$ to the makespan. Therefore, since ρn operations are removed, the corresponding time span is ρC^s_{max} and it is described in Eq. (15) below:

$$\bar{w} = |w_{end} - w_{start}| = \rho C^s_{max}. \tag{15}$$

The start of the time window $w_{start} \in [0, C^s_{max} - \bar{w}]$ is selected at random. Then, the set D is populated as follows:

$$D = \{i : i \in \Omega \wedge StartOf(\tau^s_i) \leq w_{end} \wedge EndOf(\tau^s_i) \geq w_{start}\}. \tag{16}$$

Note that, τ^s_i is used to denote the value of the variable τ_i in a solution s.

For every operation $i \in \Omega \setminus D$, its machine assignment is enforced to the corresponding value in s by imposing the following set constraints:

$$\acute{C}_M = \{PresenceOf(\phi_{i,k}) = 1, \quad \forall k \in M_i : PresenceOf(\phi^s_{i,k}) = 1 \quad \forall i \in \Omega \setminus D\} \tag{17}$$

The machine precedence relationships for all the operations of $\Omega \setminus D$ are enforced using the following set of constraints:

$$\begin{aligned} \acute{C}_R = \{&EndBeforeStart(\tau_i, \tau_j) \quad \forall k \in M_i \cap M_j : \\ &PresenceOf(\phi^s_{i,k}) = PresenceOf(\phi^s_{j,k}) \wedge EndOf(\tau^s_i) \\ &\leq StartOf(\tau^s_j), i \neq j \quad \forall i, j \in \Omega \setminus D\} \end{aligned} \tag{18}$$

Constraints (17) and (18) ensure that all the operations that are processed outside the time window $[w_{start}, w_{end}]$, are processed following the same order and on the same machine as denoted by the solution s. Lastly, the set of constraints $\acute{C}$ is defined as follows:

$$\acute{C} = \acute{C}_M \cup \acute{C}_R \tag{19}$$

4.1.2. The block operator

This operator was first introduced by Palpant et al. (2004) for the RCPSP. In this paper, we adapt it for the FJSSP. The goal of this operator is to select ρn operations that are processed in parallel in a solution s. Algorithm 2 presents an overview of the block operator. At first, an operation $i \in \Omega$ is selected at random and it is inserted into the set D and a queue structure Q. In an iterative fashion, the operator removes the first operation j from Q. Then, every operation i that satisfies $StartOf(\tau^s_i) \leq EndOf(\tau^s_j) \subsetneq EndOf(\tau^s_i) \geq StartOf(\tau^s_j)$ is added to D and appended to Q. This process is repeated until the required number of operations ρn is selected. The constraint generation process for this operator is identical to the time window extraction operator and therefore, Eqs. (17)–(19) can be used to populate the set of constraints $\acute{C}$.

4.1.3. The machine removal operator

This operator focuses on a random subset of machines $\bar{M} \subseteq M$, where $|\bar{M}| = \rho m$. Next, all the operations that are scheduled on any machine $k \in \bar{M}$ in a solution s are stored in D. More specifically, the operation set D is populated as follows:

$$D = \{i : i \in \Omega \wedge \exists k \in \bar{M} : PresenceOf(\phi^s_{i,k}) = 1\} \tag{20}$$

Similarly to the time window and the block operators, the Eqs. (17)–(19) can be used to extract the required constraints stored in set $\acute{C}$.

Algorithm 2 The block operator

Require: s, ρ, n
Select at random $i \in \Omega$
Create queue Q, $D \leftarrow \emptyset$
Q.push(i)
$D = \{i\}$
while $|D| < \rho n$ **do**
  $j \leftarrow Q$.pop()
  $V \leftarrow \emptyset$
  for each $i \in \Omega : i \notin D$ **do**
    if $StartOf(\tau_i^s) \leq EndOf(\tau_j^s) \varsigma EndOf(\tau_i^s) \geq StartOf(\tau_j^s)$ **then**
      $V = V \cup \{i\}$
    end if
  end for
  Shuffle V
  for each $i \in V$ **do**
    Q.push(i)
    $D = D \cup \{i\}$
  end for
end while
return

4.1.4. The Machine Assignment Operator I

The main aim of the Machine Assignment Operator I (MAO-I) is to identify operation-to-machine assignments that typically exist in high quality solutions. To extract this information, we use the set of elite solutions $\mathcal{R}^E$ and a memory structure OM, which we call a frequency map. This memory structure is populated using the information from a selected subset of solutions $S \subset \mathcal{R}^E$, with $|S| = \lambda N^E$, where λ is a parameter (see Sections 5.1 and 5.2 for more details). In particular, OM is represented by a two-dimensional matrix and holds the information regarding the assignment of single operations to machines. Every element $OM_{i,k}$ reflects the preference of an operation $i \in \Omega$ to be scheduled at the machine $k \in M_i$.

The frequency map OM is initialized as follows: At first, the set of elite solutions S is populated by randomly selecting λN^E solutions from $\mathcal{R}^E$. Next, all elements of OM are calculated using the formula below:

$$OM_{i,k} = \frac{1}{|S|} \sum_{s \in S} PresenceOf(\phi_{i,k}^s). \tag{21}$$

Note that, $\phi_{i,k}^s$ is used to denote the value of the variable $\phi_{i,k}$ in a solution s, while the normalization of Eq. (21) using the number of selected solutions, ensures that $0 \leq OM_{i,k} \leq 1$.

After the initialization of the frequency map OM, the operator MAO-I populates the set D with a random selection of ρn operations. Next, the operator identifies a set of operation-to-machine assignments $A_i \subseteq M_i : OM_{i,k} \leq \xi \quad \forall k \in M_i, \forall i \in D$. The parameter ξ is used to implicitly control the size of A_i, i.e., the smaller the value of ξ is, the fewer elements of OM are selected. More information about the values of the parameter ξ are presented in Section 5.1. At a next step, the set of constraints $\acute{C}$ is populated as follows:

$$\acute{C} = \{PresenceOf(\phi_{i,k}) = 0 \quad \forall k \in A_i, \forall i \in D\} \tag{22}$$

In the special case where $A_i = \emptyset$, no constraints are extracted. The elements of OM that have lower values typically represent rare operation-to-machine assignments, and by imposing the associated constraints, we enforce the CP model to exclude them from any new solution obtained during the search.

4.1.5. The machine assignment operator II

Similar to MAO-I, MAO-II populates the set D by using a random selection of ρn operations. This operator identifies a set of assignments $A_i \subseteq M_i : OM_{i,k} \geq 1 - \xi, \forall k \in M_i, \forall i \in D$. As a result, the smaller the value of ξ is, the more elements of OM are selected. Next, the set of constraints $\acute{C}$ is populated as follows.

$$\acute{C} = \{PresenceOf(\phi_{i,\acute{k}}) = 1 \quad \acute{k} = \underset{\forall k \in A_i}{\operatorname{argmax}} (OM_{i,k}), \forall i \in D\}. \tag{23}$$

Again, if $A_i = \emptyset$, no constraints are extracted. The elements of OM that have high values represent frequent operation-to-machine assignments, which are considered to be desirable solution features, and therefore, we enforce the CP model to include them in any new solution obtained during the search.

4.1.6. The machine assignment operator III

MAO-III combines the operators MAO-I and MAO-II. Specifically, for every operation $i \in D$, two sets of operation-to-machine assignments are defined: $A_i^1 \subseteq M_i : OM_{i,k} \leq \xi, \quad \forall k \in M_i$ and $A_i^2 \subseteq M_i : OM_{i,k} \geq 1 - \xi, \quad \forall k \in M_i$. Next, two sets of constraints are generated as follows:

$$\acute{C}_1 = \{PresenceOf(\phi_{i,k}) = 0, \quad \forall k \in A_i^1, \forall i \in D\} \tag{24}$$

$$\acute{C}_2 = \{PresenceOf(\phi_{i,\acute{k}}) = 1, \quad \acute{k} = \underset{\forall k \in A_i^2}{\operatorname{argmax}} (OM_{i,k}), \forall i \in D\} \tag{25}$$

Lastly, the set of constraints $\acute{C}$ is defined as follows.

$$\acute{C} = \acute{C}_1 \cup \acute{C}_2. \tag{26}$$

4.1.7. The Operation Relationship Operator

Similarly to operators MAO-I, MAO-II, and MAO-III, the Operation Relationship Operator (ORO) also uses a frequency map OR which stores the frequency of operation pair-to-machine assignments. In particular, the OR frequency map is represented by a three-dimensional matrix that holds the information regarding the precedence relationship between a pair of operations scheduled at the same machine. Every element $OR_{i,j,k}$ includes the frequency that an operation $i \in \Omega$ is scheduled (not necessarily immediately) before an operation $j \in \Omega$, where $j \neq i$, at a particular machine $k \in M_i \cap M_j$:

$$OR_{i,j,k} = \frac{1}{|S|} \sum_{s \in S} PresenceOf(\phi_{i,k}^s) \wedge PresenceOf(\phi_{j,k}^s) \wedge \left(EndOf(\phi_{i,k}^s) \leq StartOf(\phi_{j,k}^s) \right) \tag{27}$$

Note that the normalization of Eq. (27) using the number of selected solutions $|S| = \lambda N^E$, ensures that $0 \leq OR_{i,j,k} \leq 1$.

Given the initialization of the frequency map OR, ORO proceeds as follows: Let (i, j) denote a pair of operations $i, j \in \Omega$. Also, let set B be the set of all pairs of the operations of the problem:

$$B = \bigcup_{\forall i,j \in \Omega, i \neq j} (i, j). \tag{28}$$

Given the set B, ORO identifies a set of operation pairs (i, j) and a machine k, D_k that is defined as follows:

$$D_k = \{(i, j) : OR_{i,j,k} \geq 1 - \xi\} \quad \forall (i, j) \in B. \tag{29}$$

The smaller the value of ξ is, the more constraints that are extracted. The elements of OR with high values represent precedence relationships that have been frequently encountered between two operations i and j at the machine k. These precedence relationships are typically considered to be components of high quality solutions, and therefore, we choose to include them into the new solutions obtained. In our implementation, the set of constraints $\acute{C}$ is defined as follows:

$$\acute{C} = \{EndBeforeStart(\phi_{i,k}, \phi_{j,k}) \quad \forall (i, j) \in D_k, \forall k \in M\} \tag{30}$$

It is worth mentioning that the resulting constraints in $\acute{C}$ may be conflicting with one another. Let us assume the operations $\{i_1, i_2, \ldots, i_{o-1}, i_o\} \in \Omega$ that can be scheduled at a machine k, with $D_k = \{(i_1, i_2), (i_2, i_3), \ldots, (i_{o-1}, i_o), (i_o, i_1)\}$ and $o \geq 2$. The set of constraints $\{EndBeforeStart(\phi_{i_1,k}, \phi_{i_2,k}), EndBeforeStart(\phi_{i_2,k}, \phi_{i_3,k}), \ldots, EndBeforeStart(\phi_{i_{o-1},k}, \phi_{i_o,k}), EndBeforeStart(\phi_{i_o,k}, \phi_{i_1,k})\}$, compose a CP model

that is infeasible. In order to restore feasibility, we choose to neglect the constraint that corresponds to the operation relationship with the smallest frequency, i.e. $min(OR_{i_1,i_2,k}, OR_{i_2,i_3,k}, \dots, OR_{i_{o-1},i_o,k}, OR_{i_o,i_1,k})$. This procedure is repeatedly applied until all conflicts are resolved.

4.2. Population management

The proposed ALNS-CP uses a set of elite solutions $\mathcal{R}^E$ and an ordered list of solutions $\mathcal{R}^T$. The former is used to store the high quality solutions encountered throughout the search history, and the latter is used to store the trajectory of solutions encountered throughout the iterations of the algorithm. In this section, we describe the criteria that we use in order to maintain and update $\mathcal{R}^E$ and $\mathcal{R}^T$.

4.2.1. The elite set update criteria

The set $\mathcal{R}^E$ is used to store high quality and diverse solutions encountered during the search. The following criteria are used to update the set $\mathcal{R}^E$: Let s_b and s_w denote the best and the worst solutions of $\mathcal{R}^E$ in terms of the makespan, respectively. Let $d_h(s_1, s_2)$ denote the Hamming distance between two solutions s_1 and s_2 (Kasapidis et al., 2021). Also, let $\bar{d}_h(s)$ denote the average Hamming distance of a solution s from the solutions of $\mathcal{R}^E$ calculated as:

$$\bar{d}_h(s) = \frac{1}{|\mathcal{R}^E|} \sum_{\acute{s} \in \mathcal{R}^E} d_h(s, \acute{s}) \tag{31}$$

Given the above, a solution s is always inserted into $\mathcal{R}^E$, when $C^s_{max} < C^{s_b}_{max} \vee |\mathcal{R}^E| < N^E$. On the other hand, a solution s is always discarded when $s \in \mathcal{R}^E \vee \bar{d}_h(s) < \min(\{\bar{d}_h(\acute{s}) \quad \forall \acute{s} \in \mathcal{R}^E\})$. If there exists a solution $\acute{s} \in \mathcal{R}^E$ where $(C^{s_b}_{max} < C^s_{max} < C^{s_w}_{max}) \wedge (C^s_{max} \leq C^{\acute{s}}_{max}) \wedge (\bar{d}_h(s) \geq \bar{d}_h(\acute{s}))$, then $\acute{s}$ is replaced by s. If more solutions exist that satisfy the above criteria, we choose to replace one that has the largest makespan. To break ties between solutions $\acute{s}$ in terms of the makespan, we remove the solution that has the smallest Hamming distance $\bar{d}_h(\acute{s})$. At any other case, s_w is replaced.

4.2.2. The trajectory list update criteria

The list $\mathcal{R}^T$ is used to store the N^T most recently visited solutions during the search. Its main goal is to prevent the CP solver from revisiting the same solution during subsequent runs of the CP solver. A new solution s is always appended to the end of the list, while a solution $\acute{s}$ is removed from the top of the list if $|\mathcal{R}^T| = N^T$.

During the constraint extraction phase, a constraint c_s is constructed for each solution $s \in \mathcal{R}^T$ as follows:

$$c^1_s = \bigwedge_{i \in \Omega} PresenceOf(\phi_{i,k}) = 1, \quad \forall k \in M_i : PresenceOf(\phi^s_{i,k}) = 1 \tag{32}$$

$$c^2_s = \bigwedge_{i \in \Omega} \bigwedge_{j \in \Omega : i \neq j} \left(EndOf(\phi_{i,k}) \leq StartOf(\phi_{j,k})\right) \wedge \left(EndOf(\phi^s_{i,k}) \leq StartOf(\phi^s_{j,k})\right), \quad \forall k \in M_i \cap M_j : PresenceOf(\phi^s_{i,k}) \wedge PresenceOf(\phi^s_{j,k}) = 1 \tag{33}$$

$$c_s = c^1_s \wedge c^2_s \tag{34}$$

The constraint c_s prevents the CP solver from obtaining the solution s. The set of constraints $\acute{C}_T$ is used to store the corresponding constraints for all the solutions of $\mathcal{R}^T$, which are appended to the set of constraints $\acute{C}$ during the constraint generation phase:

$$\acute{C}_T = \{c_s, \quad \forall s \in \mathcal{R}^T\} \tag{35}$$

$$\acute{C} = \acute{C} \cup \acute{C}_T \tag{36}$$

4.3. Constraint extraction operator selection

A common feature of ALNS algorithms is the adaptive selection of destruction or repair operators. In accordance with the literature, the proposed ALNS-CP uses a roulette wheel selection mechanism for choosing the most suitable constraint extraction operator at each iteration. The probability of selecting each operator is calculated based on its performance during the search. In particular, every operator $p \in \mathcal{P}$ is associated with a non-negative score υ_p. All scores are initialized at a value (e.g. 100) at the start of the algorithm. Each score is increased by one when a solution s is successfully inserted to $\mathcal{R}^E$ and decreased by one otherwise. Therefore, the selection probability ζ_p of an operator p is calculated as follows:

$$\zeta_p = \frac{\upsilon_p}{\sum_{\acute{p} \in \mathcal{P}} \upsilon_{\acute{p}}} \tag{37}$$

As described in Section 4.2.1, the elite set of solutions $\mathcal{R}^E$ aims to maintain a balance between the quality and diversity of elite solutions. Therefore, this adaptive mechanism rewards the operators not only in cases when they produce high quality solutions, but also when they effectively diversify the search.

5. Computational results

This section presents the computational results derived by experiments on benchmark problem instances of the literature for assessing the performance of the proposed solution framework. Additionally, we have generated new benchmark problem instances to explore the effect of different types of resources on the solution schedules.

At first, Section 5.1 describes the adaptive tuning mechanism adopted. Next, Section 5.2 provides implementation details, as well as the parameter tuning of the proposed solution framework. Section 5.3 presents the computational results produced by applying the ALNS-CP and the CP model alone to solve benchmark problem instances of the literature. Lastly, Section 5.4 presents the results of a thorough experimentation on new problem instances with different types of resources.

5.1. The adaptive parameter tuning mechanism

The proposed ALNS-CP algorithm uses six used-defined parameters $N_E, N_T, G, \delta, \gamma, \lambda$ that remain constant for the entire run-time of the algorithm and three self-tuned parameters ρ, ξ, ι that are adaptively tuned during the search.

The parameter N^E is used to control the size of the elite solution set $\mathcal{R}^E$. High values of N^E may delay the convergence, whereas low values of N^E may lead to a premature convergence. As described in Section 4.2.2, the parameter N^T directly controls the size of the $\mathcal{R}^T$, which keeps the trajectory of solutions encountered throughout the iterations of the algorithm. Low values of N^T may cause the algorithm to get trapped in local minima, while higher values may cause a premature diversification of the search. Note that, the trajectory list $\mathcal{R}^T$ plays the role of a tabu tenure list, which is a core component in a Tabu Search algorithm (Glover & Laguna, 1997). The parameter G controls the number of maximum iterations of the algorithm without improvement, while the parameter λ controls the portion of $\mathcal{R}^E$ that is used by the extraction operators MAO I–III and ORO, for the initialization of the frequency maps OM and OR.

The self-tuned parameter ρ, that controls the size of the set D, is adaptively tuned as follows: In a span of γ iterations, we record the number of iterations γ^{opt} where the CP solver was able to solve a problem to optimality. Then, the parameter ρ is updated using the following recursive equation:

$$\rho = \rho + \frac{2\gamma^{opt} - \gamma}{\gamma}\delta \tag{38}$$

where the user-defined parameter δ controls the maximum increment of the parameter ρ. When the value of ρ and therefore the size of D is very small, the CP solver may be able to solve the vast majority of problems to optimality, i.e., $\gamma^{opt} > \frac{\gamma}{2}$ and therefore, ρ increases. This

allows the construction of larger sub-problems in the next iterations of the algorithm. On the other hand, if the values of ρ are very high, the CP solver may not be able to solve a lot of problems to optimality. Following Eq. (38), when $\gamma^{opt} < \frac{\gamma}{2}$, ρ decreases so that smaller sub-problems are explored in the next iterations of the algorithm. Additionally, the self-tuned parameter θ controls the time allocated to the CP solver. This parameter is automatically increased by one when the CP solver is able to solve a sub-problem to optimality, and decreased by one otherwise (Hojabri et al., 2018). This adaptive tuning process, controls parameters ρ and θ so that the CP solver is able to optimally solve the largest possible sub-problems, in the shortest amount of time. Note that, very small values of δ may significantly slow down the tuning process.

Lastly, the self-tuned parameter ξ is selected based on the problem flexibility f. For example, a low value of f indicates that every operation can be assigned to a small number of machines. This means that the corresponding values for the elements of the OM and OR frequency maps are relatively high. Similarly, high values of f mean that an operation can be scheduled to a larger number of machines and therefore the associated values of the frequency maps are expected to be relatively low. Based on the above, the rate ξ should be in principle higher for problems with high values of f, and lower for problems with lower values of f to enable the operators MAO-I, MAO-II, MAO-III and ORO to select a sufficient number of operation-to-machine assignments and operation pair-to-machine assignments. In our implementation, three ranges of values for the parameter ξ are used: $A^{\xi} = \{0.01, 0.025, 0.05, 0.075, 0.10\}$ where $f < 2$, $B^{\xi} = \{0.05, 0.075, 0.10, 0.125, 0.15\}$ where $2 \leq f < 3$ and $C^{\xi} = \{0.10, 0.125, 0.15, 0.175, 0.20\}$ where $f \geq 3$. A random value for ξ is selected from the corresponding range at each iteration.

5.2. Implementation details and parameter tuning

All computational experiments of this paper were conducted by using the following parameter values: $N^E = 200$, $N^T = 25$, $G = 50$, $\delta = 0.1$, $\gamma = 20$ and $\lambda = 0.5$. These values have been chosen after preliminary experiments and we describe in the following the rationale and the process followed. Regarding the parameter δ, several values were tested. Higher values deteriorated the performance of the search, since the size of the generated sub-problems fluctuated between very large and very small sizes. On the other hand, very low values cause a very slow convergence of the adaptive tuning process. Similarly, low values for γ, caused a premature change on the sub-problem size and therefore, compromise the intensification of the search. On the other hand, high values of γ may deteriorate the performance of the algorithm, since the convergence is also delayed. Preliminary computational experiments that varied the values of N^T from 20 to 100 did not show a significant impact on the performance of the algorithm. Nevertheless, if this parameter is neglected, the ALNS-CP tends to revisit past solutions. As described above, the parameter N^T resembles the tabu tenure of Tabu Search algorithms where the commonly accepted range typically falls between 20 and 30 (Glover & Laguna, 1997), and we therefore set $N^T = 25$. To determine a well performing set of values for the parameter N^E, but also the portion λ of $\mathcal{R}^E$ used for the initialization of the frequency maps, the following experiment was performed. We selected a total of ten hard-to-solve instances of the FJSSP and the BJSSP literature. More specifically, for the FJSSP the problem instances $mk06$ and $mk10$ from $BRData$ data set (Brandimarte, 1993), and the problem instances $16a, 17a$ and $18a$ from the $DPData$ (Dauzère-Pérès & Paulli, 1997) data set were selected. Regarding the BJSSP, five instances were chosen at random from the benchmark set provided by Lawrence (1984), namely the instances $la18, la25, la30, la35$ and $la38$. All problems were solved by using 16 combinations of parameter settings: $N^E \in \{50, 100, 200, 300\}$ and $\lambda \in \{0.25, 0.50, 0.75, 1.0\}$. For each combination, ten runs of the ALNS-CP were performed. The remaining parameters were initialized as described above. The best computational results were derived by using $N^E = 200$ and $\lambda = 0.5$.

Regarding all experiments conducted in this paper, for every problem instance, we used ten runs of the ALNS-CP, and we report the best solution in terms of the makespan. A time limit of 1000 seconds is applied for every run of the ALNS-CP, if the maximum number of G iterations is not reached. Note that, a time limit of 1800 seconds was used regarding the experiments conducted using the CP optimizer. The ALNS-CP was developed in C++ and uses IBM ILOG CP Solver 20.1.0 as the CP solver. All experiments were executed using an Intel Xeon E5-2650 processor with 32 GB of RAM. Lastly, the CP optimizer was used with the default search configuration.

5.3. Computational experiments on existing benchmark datasets

In this section, we present computational results derived by experiments of the ALNS-CP and the CP model on existing problem instances of the literature. In particular, Sections 5.3.1–5.3.3 present computational results for the FJSSP, the BJSSP and the Unrelated Parallel Machine scheduling problem with Resources (UPMR), respectively.

5.3.1. Comparative performance analysis for the FJSSP

This section presents computational results of experiments on well-known problem instances of the FJSSP literature. More specifically, the following benchmark sets are used: the BRData (Brandimarte, 1993), the BCData (Barnes & Chambers, 1996), the HUData (Hurink et al., 1994) and the DPData (Dauzère-Pérès & Paulli, 1997). Note that, three problem groups are included in the benchmark set HUData, namely edata, vdata and rdata. The performance of the ALNS-CP is compared to the state-of-the-art algorithms for the FJSSP, namely, the Evolutionary Algorithm (EA) by Kasapidis et al. (2021), the Scatter Search and Path Relinking (SSPR) algorithm by González et al. (2015), the Harmony Search algorithm (HHS) by Yuan et al. (2013), the Hybrid Genetic Algorithm (hGA) by Gao et al. (2008) and the Discrepancy Search Algorithm (CDDS) by Ben Hmida et al. (2010).

Table 3 summarizes the computational results for the BRData, DPData and BCData, respectively. The first column contains the name of the problem instance, while the second column lists the performance indicators, i.e., the Average makespan (Avg. C_{max}), the Average gap % (Avg. Gap (%)) from the known lower bounds, the number of optimal solutions (# Opt) and the Computer Independent CPU time (CICPU) time, for each benchmark set. Note that, we used the best known lower bound, according to Mastrolilli and Gambardella (2000), to calculate the average gap. The remaining columns include the computational results of the ALNS-CP and the CP model and the computational results of the state-of-the-art algorithms. The CICPU is calculated based on the normalization coefficients from Dongarra (1992). Since our CPU is not included in Dongarra (1992), we use the single thread CPU rating as reported in *cpubenchmark.net* in order to calculate the corresponding normalization coefficient for our machine. As indicated by both Yuan et al. (2013) and González et al. (2015), "the comparison between CI-CPU times is meant to be indicative, because we do not have access to other information that influences the computation time, such as the operating systems, the programming language, the compiler selection, and the overall code quality". The average gap (%) per instance is calculated as $\frac{C_{max}-LB}{LB}$, where LB denotes the available lower bound per instance. The results for the problem instances included in the $HUData$ benchmark set are presented in Table 4. Table 4 includes a summary of the average gap values, i.e., the first column contains the name of the algorithm, and the next columns include the average gap values for the $edata$, $rdata$ and $vdata$ problem groups, respectively.

Overall, the results showcase the effectiveness of the ALNS-CP compared to the state-of-the-art algorithms proposed for the FJSSP. The ALNS-CP produced solutions with an average gap of 3.59% and 1.75% for the BRData and the DPData benchmark sets, respectively, which makes it the fourth and third best performing algorithm. Also, the EA is the best performing algorithm for the BRData and DPData, whereas it comes second for BCData. Regarding the BCData benchmark set, the

Table 3
Summary of computational results for the BRData, DPData and BCData benchmark sets.

Dataset		ALNS-CP	CP	EA	SSPR	CDDS	hGA	HHS
BRData	Avg. C_{max}	172.70	172.90	172.20	172.40	172.70	172.60	173.20
	Avg. Gap (%)	3.59	3.70	**3.17**	3.27	3.60	3.54	4.02
	# Opt.	6	6	6	6	6	6	6
	CICPU	1204	9882	824	383	96	91	–
DPData	Avg. C_{max}	2209.44	2216.83	2202.00	2203.50	2211.56	2215.56	2210.22
	Avg. Gap (%)	1.75	2.09	**1.40**	1.47	1.84	2.02	1.78
	# Opt.	3	3	4	3	1	0	2
	CICPU	8622	29 855	6320	1934	2890	6206	6279
BCData	Avg. C_{max}	995.19	995.19	995.33	995.52	997.10	929.60	–
	Avg. Gap (%)	**0.00**	**0.00**	0.02	0.03	0.19	0.25	–
	# Opt.	21	21	20	18	7	9	–
	CICPU	997	275	968	1138	253	821	–

Table 4
Average gap (%) of various solutions methods for the HUData benchmark set (problem instances la01 to la40 and mt06/10/20).

Dataset	ALNS-CP	CP	EA	SSPR	HHS	hGA	CDDS
edata	2.01	2.07	**1.99**	2.03	2.11	2.13	2.32
rdata	0.95	1.16	**0.94**	1.03	1.18	1.17	1.34
vdata	**0.02**	0.05	0.05	0.04	0.11	0.08	0.11

Table 5
Summary of computational results for the Lawrence (1984) benchmark set.

	ALNS-CP	CP	PTS	PBB	IGM	IFS
Avg. C_{max}	1509.03	1604.98	1501.25	1551.25	1589.13	1602.83
Avg. RPD (%)	1.17	7.23	0.70	2.58	5.11	5.72
# Best. (NB)	20 (10)	1 (0)	15	21	7	6
CICPU	31 169	100 080	–	–	24 000	85 553

ALNS-CP managed to find optimal solutions for all problem instances, and ranked as the best performing algorithm. Similarly, regarding the HUData benchmark set, the ALNS-CP is the second best performing algorithm for the *edata* and *rdata* problem sets (after the EA), while it performs better than all the state-of-the-art algorithms for the *vdata* problem set. In total, the ALNS-CP produced 14 new best solutions for problem instances of the *rdata* and the *vdata* problem sets. Detailed results of the ALNS-CP algorithm and the proposed CP model for all examined FJSSP problem instances, as well as a detailed performance comparison to the state-of-the-art, are provided in Appendix B of the online companion.

5.3.2. Comparative performance analysis for BJSSP

This section assesses the performance of the ALNS-CP for the BJSSP. The main difference between the BJSSP and the JSSP is the presence of the blocking constraints (see Section 3.1). Two variants of the BJSSP exist in the literature, namely the Blocking-With-Swap (BWS) and the Blocking-No-Swap (BNS). The BWS variant can be considered as a special case of the FJSSP with resource constraints, where $|M_i| = 1$, $\forall i \in \Omega$ and $y_k = 0$, $\forall k \in M$, and this is the variant we focus on this section.

The problem instances of Lawrence (1984) for the JSSP were used as the test bed for the BJSSP. The performance of the proposed ALNS-CP and the CP model are compared with the Parallel Tabu Search (PTS) algorithm by Dabah et al. (2019), the Parallel Branch and Bound method (PBB) of Dabah et al. (2016), the Iterated Greedy Meta-heuristic (IGM) of Pranzo and Pacciarelli (2016) and the Iterative Flattening Search (IFS) algorithm of Oddi et al. (2012).

The results of these computational experiments are presented in Table 5. The structure of the table is similar to the structure of Table 3. Note that, Table 5 includes the average relative percentage deviation (Avg. RPD (%)) from the best known solutions instead of the average gap (%), since lower bound values are not available. The RPD per instance is calculated as $\frac{C_{max}-C^b_{max}}{C^b_{max}}$, where C^b_{max} denotes the best known solution per instance. We calculated the corresponding RPD values using the best known solutions reported by Pranzo and Pacciarelli (2016), Oddi et al. (2012), Dabah et al. (2016, 2019). Also, since lower bound values are not available, the number of best solutions is reported instead of the number of optimal solutions, i.e., the number of instances that the corresponding algorithm obtained the current best known solution. Overall, one can observe that the ALNS-CP produced solutions with an average RPD value of 1.17%, which makes it the second best performing algorithm compared to the state-of-the-art. The ALNS-CP managed to obtain ten new best solutions (indicated by a parenthesis on the table) for the problem instances la14, la15, la26, la27, la28, la30, la31, la32, la34 and la39 (see Appendix C for detailed results).

5.3.3. Comparative performance analysis for UPMR

This section assesses the performance of the proposed ALNS-CP algorithm for the UPMR. The UPMR is an extension of the unrelated parallel machines scheduling problem (UPM). The main difference between the UMP and the FJSSP is that operations can be processed on all the available machines $k \in M$, with $M_i = M, \forall i \in \Omega$. The UPMR extends the UPM by considering additional renewable resources. This problem is also a special case of the FJSSP with resource constraints, and the resources used in UPMR are a representative example of utility resources.

For this purpose, the problem instances of Fanjul-Peyro et al. (2017) were used. This benchmark data set includes two groups of problem instances. The first includes small-scale problem instances with $8 \leq l \leq 16$, while the second includes medium-scale problem instances with $20 \leq l \leq 30$. Since the small-scale problem instances are easy to solve, in this paper we have only used the medium-scale problem instances, and we compare our results with the algorithms of Fanjul-Peyro et al. (2017) and Fleszar and Hindi (2018).

The medium-scale benchmark set includes 450 problem instances in total and Table 6 presents a summary of the computational results. The first column includes the number of jobs, and the second column includes the number of instances that share the same number of jobs. The next four multi-columns include the computational results of the algorithms presented by Fleszar and Hindi (2018) (FH), Fanjul-Peyro et al. (2017) (FP), the ALNS-CP and the CP model, respectively. Each multi-column includes the number of optimal solutions and the number of new best solutions obtained by the ALNS-CP in a parenthesis. The second column presents the average gap (%) per instance group from the lower bounds reported in Fleszar and Hindi (2018). The last three rows of the table present the total optimal solutions obtained, the total average gap (%) and the CICPU time, respectively.

One can observe that the ALNS-CP improves the total average gap of 450 problem instances by 0.01%, compared to Fleszar and Hindi (2018), but it needed more computational time. The results also show that the off-the-shelf CP solver is unable to produce high quality solutions in short computational times, without the high level guidance of our ALNS framework. In total, the ALNS-CP obtained 407 optimal

Table 6
Summary of computational results for the UPMR.

l	#	ALNS-CP		CP		FH		FP	
		# Opt (NB)	Avg.Gap (%)	# Opt	Avg.Gap (%)	# Opt	Avg.Gap (%)	# Opt	Avg.Gap (%)
20	150	134 (5)	0.08	119	0.21	134	0.10	92	1.08
25	150	138 (5)	0.08	112	0.39	139	0.09	76	2.21
30	150	135 (5)	0.08	88	0.45	135	0.09	79	2.22
# Opt. (NB)		407 (15)		319		408		247	
Avg.Gap (%)		0.08		0.35		0.09		1.97	
CICPU (s)		25 934		224 503		17 848		–	

solutions, including 15 new best solutions (see Appendix D for detailed results).

5.4. Computational experiments on new data sets with multiple types of resources

In this section, the aim is to study the effect of various resources types on the solution schedules, and for this purpose, we have generated new small and large-scale problem instances. Sections 5.4.1–5.4.3 present an analysis of experiments on problems that feature limited capacity buffers, tool renewable resources and arbitrary resources and WIP buffers, respectively. For these experiments, the CP model is used to solve the corresponding problems. Lastly, Section 5.4.4 presents results on new large-scale problem instances that include all available resource types. The generated benchmark sets are available in the following repository: https://github.com/gkasapidis/RCFJSSP.

5.4.1. Computational experiments on problem instances with limited capacity buffers

To study the effect of the limited capacity buffer constraints to the makespan, a set of ten representative problem instances have been generated. These problem instances were produced so that they are large and complex enough, but at the same time solvable to optimality in a reasonable computational time. The generated problem instances share the same number of jobs and the same number of machines, but differ in terms of machine assignments and processing times, while the buffer capacities are set ≤ 2. We consider three different cases: (a) zero capacity buffers, (b) random capacity buffers $0 \leq y_k \leq 1, \forall k \in M$ and (c) unlimited capacity buffers. The computational results are presented in Table 7. The first four columns present the name of the problem instance, the number of jobs l, the number of machines m and the number of operations n. The next three multi-columns include the computational results for each case. Each multi-column includes four columns, that present the lower bound (LB), the makespan (C_{max}), the gap from LB ($Gap(\%)$) and the time required to solve the instance by the proposed CP model (Time (s)). The last row provides the average values.

All problem instances were solved to optimality via the proposed CP model. As the overall capacity of buffers increases, we notice a decrease of the average makespan as well as the average time required to solve all problem instances. More specifically, we observe an average makespan of 95.90 for problem instances with zero capacity buffers. In the case of random capacity buffers, the average makespan decreases to 93.80, which can be attributed to the fact that machines can store job operations in the corresponding buffers without blocking the machines. The same effect is prominent in the case of unlimited buffers, where no blocking is imposed to any machine, and therefore, the smallest average makespan of 91.30 is obtained. We can lastly observe that the average computational time increases as the capacity of the buffers decreases.

5.4.2. Computational experiments on problem instances with tool constraints

In this section, we study the effect of the tool resource constraints with respect to the makespan. For this purpose, ten new problem instances were generated with a variable number of jobs and machines per instance. The number of available tools L_T was set to 10, whereas the amount of instances per tool was controlled by the parameter β that varied according to $\beta \in \{10\%, 25\%, 50\%, 75\%, 100\%\}$. For example, if ten operations require a tool $r \in T$ and $\beta = 100\%$, then all operations can be processed in parallel. In a different case, if $\beta = 10\%$ the amount of tool instances would be limited to 1, and therefore, the operations must compete for the available tool instances, which leads to waiting times and prolonged schedules.

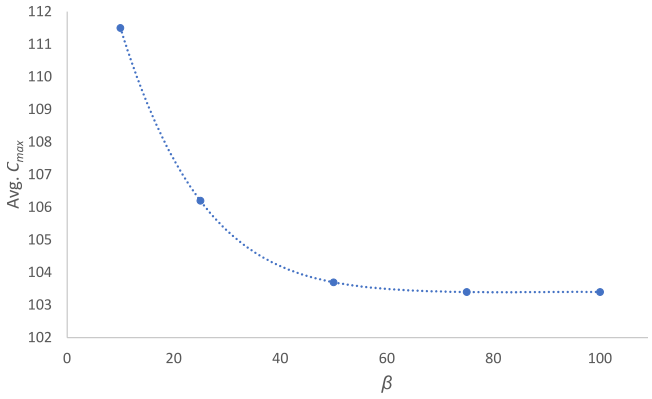


Fig. 3. Average C_{max} for different values of β.

The computational results are presented in Table 8. Each multi-column shows the results for the values of $\beta = 10\%$, $\beta = 25\%$, $\beta = 50\%$, $\beta = 75\%$ and $\beta = 100\%$ and includes four columns. The first column presents the lower bound (LB), the second column presents the makespan (C_{max}), the third column presents the gap from LB ($Gap(\%)$) and the fourth column includes the computational time required in seconds ($Time(s)$).

One can observe an increase of the makespan as the tool availability decreases. More specifically, when $\beta = 10\%$ the average makespan is 111.5, whereas when $\beta = 100\%$ the average makespan is 103.4. It is worth mentioning that, the impact on the makespan gradually decreases as β increases. In particular, when β increases from 10% to 25%, the makespan decreases by 4.75%, when β increases from 25% to 50%, the makespan decreases by 2.35% and lastly, when β increases from 50% to 75%, the makespan decreases by 0.30%. Given a limited availability of the tools, we can observe that a good enough value for β is 50%, since from that point and on-wards the improvement of the solution quality is marginal (see Fig. 3).

5.4.3. Computational experiments on problem instances with arbitrary resources and WIP buffers

In this section, we study the effect of cumulative and non-renewable resources in the form of WIP buffers and arbitrary resources on the makespan. For this purpose, ten problem instances with resource constraints were generated with $L_W = 2$ and $L_R = 10$. Three cases are

Table 7
Computational results on experiments with limited capacity buffers.

Instance	l	m	n	w/ Machine buffers				w/o Buffers				Unlimited buffers			
				LB	C_{max}	Gap (%)	Time(s)	LB	C_{max}	Gap (%)	Time (s)	LB	C_{max}	Gap (%)	Time (s)
lcb0	4	5		102	102	0.00	0.05	102	102	0.00	0.02	102	102	0.00	0.01
lcb1	4	5		95	95	0.00	0.03	95	95	0.00	0.02	88	88	0.00	0.01
lcb2	4	5		88	88	0.00	0.02	91	91	0.00	0.02	88	88	0.00	0.01
lcb3	4	5		95	95	0.00	0.02	95	95	0.00	0.02	95	95	0.00	0.01
lcb4	4	5		99	99	0.00	0.04	101	101	0.00	0.04	93	93	0.00	0.02
lcb5	4	5		98	98	0.00	0.02	105	105	0.00	0.06	95	95	0.00	0.01
lcb6	4	5		92	92	0.00	0.03	92	92	0.00	0.03	88	88	0.00	0.01
lcb7	4	5		84	84	0.00	0.02	86	86	0.00	0.02	84	84	0.00	0.01
lcb8	4	5		88	88	0.00	0.02	92	92	0.00	0.03	86	86	0.00	0.01
lcb9	4	5		97	97	0.00	0.02	100	100	0.00	1.31	94	94	0.00	0.01
Avg					93.8	0.00	0.03		95.9	0.00	0.15		91.3	0.00	0.01

Table 8
Computational results on problems with tool resource constraints.

Instance	$\beta = 10\%$				$\beta = 25\%$				$\beta = 50\%$				$\beta = 75\%$				$\beta = 100\%$			
	LB	C_{max}	Gap	Time (s)	LB	C_{max}	Gap	Time (s)	LB	C_{max}	Gap	Time (s)	LB	C_{max}	Gap	Time (s)	LB	C_{max}	Gap	Time (s)
t0	107	107	0.00	0.39	101	101	0.00	0.17	100	100	0.00	0.01	100	100	0.00	0.01	100	100	0.00	0.01
t1	103	103	0.00	0.61	101	101	0.00	0.49	97	97	0.00	0.19	97	97	0.00	0.18	97	97	0.00	0.52
t2	107	107	0.00	0.32	102	102	0.00	0.33	102	102	0.00	0.18	101	101	0.00	0.26	101	101	0.00	0.21
t3	114	114	0.00	5.08	105	105	0.00	1.40	99	99	0.00	2.66	99	99	0.00	1.76	99	99	0.00	1.34
t4	116	116	0.00	1.20	110	110	0.00	0.28	108	108	0.00	0.53	107	107	0.00	0.33	107	107	0.00	0.35
t5	108	108	0.00	1.84	101	101	0.00	0.95	99	99	0.00	0.57	99	99	0.00	0.23	99	99	0.00	0.34
t6	105	105	0.00	1.57	98	98	0.00	0.78	97	97	0.00	0.48	97	97	0.00	0.56	97	97	0.00	0.60
t7	118	118	0.00	7.22	114	114	0.00	1.89	113	113	0.00	2.69	112	112	0.00	2.11	112	112	0.00	1.60
t8	118	118	0.00	4.16	115	115	0.00	14.20	112	112	0.00	21.90	112	112	0.00	27.25	112	112	0.00	19.98
t9	119	119	0.00	0.24	115	115	0.00	0.50	110	110	0.00	0.36	110	110	0.00	0.45	110	110	0.00	0.75
Avg		111.5	0.00	2.26		106.2	0.00	2.10		103.7	0.00	2.96		103.4	0.00	3.31		103.4	0.00	2.57

Table 9
Computational results on experiments with non renewable resources and WIP buffers.

Instance	Small buffers				Large buffers				No buffers			
	LB	C_{max}	Gap (%)	Time(s)	LB	C_{max}	Gap (%)	Time (s)	LB	C_{max}	Gap (%)	Time(s)
r0	82	82	0.00	5.12	71	71	0.00	0.81	63	63	0.00	0.19
r1	69	69	0.00	0.22	65	65	0.00	0.04	37	37	0.00	0.16
r2	54	54	0.00	0.41	54	54	0.00	0.11	33	33	0.00	0.08
r3	69	69	0.00	3.47	65	65	0.00	0.92	50	50	0.00	0.95
r4	64	64	0.00	0.19	62	62	0.00	0.11	45	45	0.00	0.48
r5	72	72	0.00	0.51	70	70	0.00	0.47	53	53	0.00	0.40
r6	72	72	0.00	0.12	69	69	0.00	0.12	42	42	0.00	0.15
r7	99	99	0.00	1.01	85	85	0.00	0.13	57	57	0.00	0.19
r8	79	79	0.00	13.73	71	71	0.00	3.02	65	65	0.00	0.05
r9	74	74	0.00	0.30	63	63	0.00	0.04	44	44	0.00	0.49
Avg		73.4	0.00	2.51		67.5	0.00	0.58		48.9	0.00	0.31

considered: (a) low capacity WIP buffers, (b) high capacity WIP buffers and (c) unlimited capacity WIP buffers.

The computational results are presented in Table 9 that shares the same structure with Tables 7 and 8. In particular, the table has three multi-columns and each multi-column shows the results for low capacity WIP buffers, high capacity WIP buffers and unlimited WIP buffers. One can observe that the makespan decreases as the size of the WIP buffer increases. More specifically, we notice an average makespan of 73.40 for low capacity WIP buffers and an average makespan of 67.50 for high capacity WIP buffers, which corresponds to a decrease of almost 8%. When unlimited WIP buffers are considered, we notice an even more significant decrease of the makespan to 48.90, i.e. a 27.6% decrease from the case of high capacity buffers. We can observe that the WIP buffers significantly affect the solution quality. Lastly, in terms of the computational time, we notice that the average time needed to obtain an optimal solution for small buffers is significantly larger than the other two cases.

5.4.4. Computational experiments on problem instances with combined resources

In this section, we consider problem instances that feature a combination of all resource types supported by the proposed CP model, i.e., limited capacity buffers, utility resources, tool resources, arbitrary non-renewable resources and WIP buffers. In order to generate the problem instances, the following parameter ranges were used: $10 \leq l \leq 15$, $5 \leq m \leq 10$, $5 \leq \eta_{i,k} \leq 20$, $2 \leq f \leq 4$, $1 \leq L_T \leq 5$, $1 \leq L_U \leq 3$, $L_W = 2$, $L_R = 8$, $0 \leq y_k \leq 3$, $8 \leq \bar{U}_r \leq 10$, $1 \leq \bar{T}_r \leq 5$. In total, we generated ten representative problem instances ($lcbrtu0 - lcbrtu9$) that were solved using the CP model as well as the proposed ALNS-CP algorithm.

The computational results are presented in Table 10. The table includes two multi-columns that present the results produced by the CP model and the ALNS-CP, respectively. One can observe the superiority of the ALNS-CP. The ALNS-CP has overall a better average gap of 17.43% compared to 19.11% of the CP model, while this average gap

Table 10
Computational results on generated large-scale problem instances.

Instance	CP				ALNS-CP		
	LB	C_{max}	Gap (%)	Time (s)	C_{max}	Gap (%)	Time(s)
lcbrtu0	293	352	20.14	1800	**345**	18.43	623.45
lcbrtu1	299	**312**	4.35	1800	**312**	4.35	179.86
lcbrtu2	258	280	8.53	1800	**278**	7.36	476.44
lcbrtu3	297	411	38.38	1800	**405**	29.97	495.88
lcbrtu4	288	322	11.81	1800	**314**	8.33	604.69
lcbrtu5	276	359	30.07	1800	**356**	27.90	643.44
lcbrtu6	283	299	5.65	1800	**297**	4.59	231.97
lcbrtu7	264	290	9.85	1800	**288**	8.33	283.92
lcbrtu8	256	406	58.59	1800	**390**	47.27	800.39
lcbrtu9	294	**305**	3.74	1800	**305**	3.74	206.70
Avg		333.6	19.11	1800	329	17.43	454.67

is achieved in much shorter computational times. These results indicate the applicability of the proposed ALNS-CP for solving a wide variety of flexible job shop scheduling problems with resource constraints.

6. Conclusions and future research

This paper studied the FJSSP with renewable, non-renewable as well as cumulative resources. A novel unified solution framework was proposed for solving the FJSSP with and without resource constraints. This framework consists of an adaptive large neighbourhood search that uses a CP solver (ALNS-CP) for generating new solutions and exploring the solution space. In particular, the proposed ALNS-CP uses long-term memory structures that hold information about single operation-to-machine assignments and operation pair-to-machine assignments that have been encountered in high quality and diverse solutions through the search history. This information is translated into constraint expressions that are imposed to the CP solver. The goal was to guide the CP solver towards promising regions of the solution space.

To evaluate the performance of the proposed CP model and the ALNS-CP algorithm, we used well-known benchmark data sets from the FJSSP, the UPMR and the BJSSP literature. The computational results show that the ALNS-CP is highly competitive compared to the state-of-the-art algorithms of the literature for solving flexible job shop scheduling problems with resource constraints, while it outperforms the CP solver. Moreover, the ALNS-CP managed to produce 15 new best solutions for the UPMR problem instances, ten new best solutions for the BJSSP problem instances and 14 new best solutions for the FJSSP problem instances.

Three different sets of experiments were conducted using new small-scale problem instances to study the effect of various resource types on the makespan. More specifically, we studied the effect of limited capacity buffers, renewable discrete resources and non-renewable resources combined with WIP buffers. In the first set of experiments, the focus was on testing various capacities for the machine buffers. The results showed that the zero capacity buffers caused a 5.03% increase of the makespan compared to the case of unlimited capacity buffers, whereas, when random capacity buffers are considered, the makespan increased by 2.73%. In the second set of experiments, the focus was on testing various settings for the availability of tools. The results showed that high quality solutions could be obtained by considering a 50% tool availability, since higher availability values seemed not to have any significant effect on the reduction of the makespan. In the third set of experiments, we studied the effect of small and large WIP buffers on the makespan. The results showed that small WIP buffers can have a significant impact on the makespan (e.g., 50.10% increase compared to the case of unlimited WIP buffers). Lastly, we conducted computational experiments on new problem instances combining all resource types. The results verified the superiority of the ALNS-CP framework compared to solving the problem by using an off-the-shelf CP solver.

In terms of future research, it would be worth considering multiple resource-related objectives, such as the maximization of the utilization of buffers, the minimization of resource consumption rates and the minimization of energy costs or carbon footprint. Regarding the solution framework, it would be worth exploring the use of machine learning to predict a useful set of constraints and enable the CP solver to propagate constraints more effectively.

CRediT authorship contribution statement

Gregory A. Kasapidis: Writing – review & editing, Writing – original draft, Software, Methodology, Investigation, Conceptualization. **Dimitris C. Paraskevopoulos:** Writing – review & editing, Writing – original draft, Supervision, Methodology, Conceptualization. **Ioannis Mourtos:** Writing – review & editing, Writing – original draft, Supervision. **Panagiotis P. Repoussis:** Writing – review & editing, Writing – original draft, Supervision, Conceptualization.

Acknowledgements

The authors would like to gratefully acknowledge support from the European Commission [FACTLOG Project, award number 869951] and the Athens University of Economics and Business Research Center.

Appendix A. Supplementary data

Supplementary material related to this article can be found online at https://doi.org/10.1016/j.ejor.2024.08.010.

References

Abreu, L. R., & Nagano, M. S. (2022). A new hybridization of adaptive large neighborhood search with constraint programming for open shop scheduling with sequence-dependent setup times. *Computers & Industrial Engineering*, *168*.

Aggoun, A., & Beldiceanu, N. (1993). Extending chip in order to solve complex scheduling and placement problems. *Math and Comp. Model.*, *17*(7), 57–73.

Agnetis, A., Murgia, G., & Sbrilli, S. (2014). A job shop scheduling problem with human operators in handicraft production. *International Journal of Production Research*, *52*(13), 3820–3831.

Andrade-Pineda, J. L., Canca, D., Gonzalez-R, P. L., & Calle, M. (2020). Scheduling a dual-resource flexible job shop with makespan and due date-related criteria. *Anal. Oper. Res.*, *291*(1–2), 5–35.

Aschauer, A., Roetzer, F., Steinboeck, A., & Kugi, A. (2017). An efficient algorithm for scheduling a flexible job shop with blocking and no-wait constraints. *IFAC-PapersOnLine*, *50*(1), 12490–12495.

Aschauer, A., Roetzer, F., Steinboeck, A., & Kugi, A. (2018). Scheduling of a flexible job shop with multiple constraints. *IFAC-PapersOnLine*, *51*(11), 1293–1298.

Azzi, A., Faccio, M., Persona, A., & Sgarbossa, F. (2012). Lot splitting scheduling procedure for makespan reduction and machine capacity increase in a hybrid flow shop with batch production. *Int. J. Adv. Manuf. Syst.*, *59*(5–8), 775–786.

Barnes, J. W., & Chambers, J. B. (1996). Tabu search for the flexible-routing job shop problem. *The University of Texas, Austin, TX, Technical Report Series ORP96-10, Graduate Program in Operations Research and Industrial Engineering*, 1–11.

van der Beek, T., Souravlias, D., van Essen, J. T., Pruyn, J., & Aardal, K. (2024). Hybrid differential evolution algorithm for the resource constrained project scheduling problem with a flexible project structure and consumption and production of resources. *European Journal of Operational Research*, *313*(1), 92–111.

Beezão, A. C., Cordeau, J. F., Laporte, G., & Yanasse, H. H. (2017). Scheduling identical parallel machines with tooling constraints. *European Journal of Operational Research*, *257*(3), 834–844.

Belaid, R., T'Kindt, V., & Esswein, C. (2012). Scheduling batches in flowshop with limited buffers in the shampoo industry. *European Journal of Operational Research*, *223*(2), 560–572.

Ben Hmida, A., Haouari, M., Huguet, M.-J., & Lopez, P. (2010). Discrepancy search for the flexible job shop scheduling problem. *Computers & Operations Research*, *37*(12), 2192–2201.

Bitar, A., Dauzère-Pérès, S., Yugma, C., & Roussel, R. (2016). A memetic algorithm to solve an unrelated parallel machine scheduling problem with auxiliary resources in semiconductor manufacturing. *Journal of Scheduling*, *19*(4), 367–376.

Błażewicz, J., Cellary, W., Słowiński, R., & Węglarz, J. (1986). *vol. 7, Scheduling Under Resource Constraints: Deterministic Models* (pp. 1–359). Basel: J.C.Baltzer AG.

Boufellouh, R., & Belkaid, F. (2020). Bi-objective optimization algorithms for joint production and maintenance scheduling under a global resource constraint: Application to the permutation flow shop problem. *Computers & Operations Research*, *122*, Article 104943.

Brandimarte, P. (1993). Routing and scheduling in a flexible job shop by tabu search. *Anal. Oper. Res.*, *41*(3), 157–183.

Brucker, P., Drexl, A., Mohring, R., Neumann, K., & Pesch, E. (1999). Resource-constrained project scheduling: Notation, classification, models, and methods. *European Journal of Operational Research*, *112*(1), 3–41.

Brucker, P., Heitmann, S., Hurink, J., & Nieberg, T. (2006). Job-shop scheduling with limited capacity buffers. *OR Spectr.*, *28*(2), 151–176.

Brucker, P., & Schlie, R. (1990). Job-shop scheduling with multi-purpose machines. *Computing*.

Campos Ciro, G., Dugardin, F., Yalaoui, F., & Kelly, R. (2016). Open shop scheduling problem with a multi-skills resource constraint: A genetic algorithm and an ant colony optimisation approach. *International Journal of Production Research*, *54*(16), 4854–4881.

Carlier, J. (1982). The one-machine sequencing problem. *European Journal of Operational Research*, *11*(1), 42–47.

Chakrabortty, R. K., Abbasi, A., & Ryan, M. J. (2020). Multi-mode resource-constrained project scheduling using modified variable neighborhood search heuristic. *International Journal of Production Research*, *27*(1), 138–167.

Chan, F. T., Wong, T. C., & Chan, L. Y. (2006). Flexible job-shop scheduling problem under resource constraints. *International Journal of Production Research*, *44*(11), 2071–2089.

Coelho, J., & Vanhoucke, M. (2011). Multi-mode resource-constrained project scheduling using RCPSP and SAT solvers. *European Journal of Operational Research*, *213*(1), 73–82.

Dabah, A., Bendjoudi, A., & Aitzai, A. (2016). Efficient parallel B&B method for the blocking job shop scheduling problem. *2016 International Conference on High Performance Computing and Simulation*, 784–791.

Dabah, A., Bendjoudi, A., AitZai, A., & Taboudjemat, N. N. (2019). Efficient parallel tabu search for the blocking job shop scheduling problem. *Soft Computing*, *23*(24), 13283–13295.

Dauzère-Pérès, S., & Paulli, J. (1997). An integrated approach for modeling and solving the general multiprocessor job-shop scheduling problem using tabu search. *Anal. Oper. Res.*, *70*, 281–306.

Debels, D., & Vanhoucke, M. (2007). A decomposition-based genetic algorithm for the resource-constrained project-scheduling problem. *Operations Research*, *55*(3), 457–469.

Dongarra, J. J. (1992). Performance of various computers using standard linear equations software. *ACM SIGARCH Computer Architecture News*.

Elloumi, S., Fortemps, P., & cir Loukil, T. (2017). Multi-objective algorithms to multi-mode resource-constrained projects under mode change disruption. *Computers & Industrial Engineering*, *106*, 161–173.

Elmaraghy, H., Patel, V., & Abdallah, I. B. (2000). Scheduling of manufacturing systems under dual-resource constraints using genetic algorithms. *Journal of Manufacturing Systems*, *19*(3), 186–201.

Fan, J., Zhang, C., Liu, Q., Shen, W., & Gao, L. (2022). An improved genetic algorithm for flexible job shop scheduling problem considering reconfigurable machine tools with limited auxiliary modules. *Journal of Manufacturing Systems*, *62*(November 2021), 650–667.

Fanjul-Peyro, L., Perea, F., & Ruiz, R. (2017). Models and matheuristics for the unrelated parallel machine scheduling problem with additional resources. *European Journal of Operational Research*, *260*(2), 482–493.

Figielska, E. (2014). A heuristic for scheduling in a two-stage hybrid flowshop with renewable resources shared among the stages. *European Journal of Operational Research*, *236*(2), 433–444.

Figielska, E. (2018). Scheduling in a two-stage flowshop with parallel unrelated machines at each stage and shared resources. *Computers & Industrial Engineering*, *126*, 435–450.

Fleszar, K., & Hindi, K. S. (2018). Algorithms for the unrelated parallel machine scheduling problem with a resource constraint. *European Journal of Operational Research*, *271*(3), 839–848.

Gao, J., Sun, L., & Gen, M. (2008). A hybrid genetic and variable neighborhood descent algorithm for flexible job shop scheduling problems. *Computers & Operations Research*, *35*(9), 2892–2907.

Gehring, M., Volk, R., & Schultmann, F. (2022). On the integration of diverging material flows into resource-constrained project scheduling. *European Journal of Operational Research*, *303*, 1071–1087.

Glover, F., & Laguna, M. (1997). *Tabu Search*. Norwell, MA, USA: Kluwer Academic Publishers.

Gómez Sánchez, M., Lalla-Ruiz, E., Fernández Gil, A., Castro, C., & Voß, S. (2023). Resource-constrained multi-project scheduling problem: A survey. *European Journal of Operational Research*, *309*(3), 958–976.

González, M. A., Vela, C. R., & Varela, R. (2015). Scatter search with path relinking for the flexible job shop scheduling problem. *European Journal of Operational Research*, *245*(1), 35–45.

Grigoriev, A., Holthuijsen, M., & van de Klundert, J. (2005). Basic scheduling problems with raw material constraints. *Naval Research Logistics*, *52*(6), 527–535.

Gröflin, H., Pham, D. N., & Bürgy, R. (2011). The flexible blocking job shop with transfer and set-up times. *Journal of Combinatorial Optimization*, *22*(2), 121–144.

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

Györgyi, P., & Kis, T. (2018). Minimizing the maximum lateness on a single machine with raw material constraints by branch-and-cut. *Computers & Industrial Engineering*, *115*, 220–225.

Györgyi, P., & Kis, T. (2019). Minimizing total weighted completion time on a single machine subject to non-renewable resource constraints. *Journal of Scheduling*, *22*(6), 623–634.

Hanzálek, Z., & Šůcha, P. (2017). Time symmetry of resource constrained project scheduling with general temporal constraints and take-give resources. *Anal. Oper. Res.*, *248*(1), 209–237.

Hartmann, S., & Briskorn, D. (2022). An updated survey of variants and extensions of the resource-constrained project scheduling problem. *European Journal of Operational Research*, *297*, 1–14.

Hashimoto, S., & Mizuno, S. (2021). A tight approximation ratio of a list scheduling algorithm for a single-machine scheduling problem with a non-renewable resource. *Journal of Scheduling*, *24*(3), 259–267.

Hellsten, E. O., Sacramento, D., & Pisinger, D. (2020). An adaptive large neighbourhood search heuristic for routing and scheduling feeder vessels in multi-terminal ports. *European Journal of Operational Research*, *287*(2), 682–698.

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

Hojabri, H., Gendreau, M., Potvin, J. Y., & Rousseau, L. M. (2018). Large neighborhood search with constraint programming for a vehicle routing problem with synchronization constraints. *Computers & Operations Research*, *92*, 87–97.

Hurink, J., Jurisch, B., & Thole, M. (1994). Tabu search for the job-shop scheduling problem with multi-purpose machines. *OR Spectr.*, *225*(1981), 223–225.

Kasapidis, G. A., Paraskevopoulos, D. C., Repoussis, P. P., & Tarantilis, C. D. (2021). Flexible job shop scheduling problems with arbitrary precedence graphs. *Prod. Oper. Manag.*, *30*(11), 4044–4068.

Kovacs, A. A., Parragh, S. N., Doerner, K. F., & Hartl, R. F. (2012). Adaptive large neighborhood search for service technician routing and scheduling problems. *Journal of Scheduling*, *15*, 579–600.

Kreter, S., Schutt, A., Stuckey, P. J., & Zimmermann, J. a. (2018). *vol. 266*, *Mixed-integer linear programming and constraint programming formulations for solving resource availability cost problems* (pp. 472–486). Elsevier B.V..

Kreter, S., Schutt, A., & Stuckey, P. J. a. (2017). Using constraint programming for solving RCPSP/max-cal. *Constraints*, *22*, 432–462.

Laborie, P., Rogerie, J., Shaw, P., & Vilím, P. (2018). IBM ILOG CP optimizer for scheduling: 20+ years of scheduling with constraints at IBM/ILOG. *Constraints*, *23*(2), 210–250.

Latorre-Núñez, G., Lüer-Villagra, A., Marianov, V., Obreque, C., Ramis, F., & Neriz, L. (2016). Scheduling operating rooms with consideration of all resources, post anesthesia beds and emergency surgeries. *Computers & Industrial Engineering*, *97*, 248–257.

Lawrence, S. (1984). *Resource constrained project scheduling: an experimental investigation of heuristic scheduling techniques (Supplement)*: *Technical report*, Graduate School of Industrial Administration, Carnegie Mellon University.

Leenaerts, D. M. W., & Bokhoven, W. M. G. V. (1998). *Piecewise Linear Modeling and Analysis*. Springer New York, NY.

Lei, D., & Guo, X. (2014). Variable neighbourhood search for dual-resource constrained flexible job shop scheduling. *International Journal of Production Research*, *52*(9), 2519–2529.

Li, Y., Carabelli, S., Fadda, E., Manerba, D., Tadei, R., & Terzo, O. (2020). Machine learning and optimization for production rescheduling in industry 4.0. *Int. J. Adv. Manuf. Syst.*, *110*(9–10), 2445–2463.

Li, K., Chen, J., Fu, H., Jia, Z., & Fu, W. (2019). Uniform parallel machine scheduling with fuzzy processing times under resource consumption constraint. *Applied Soft Computing*, *82*, Article 105585.

Li, X., & Gao, L. (2020). *Effective Methods for Integrated Process Planning and Scheduling* (p. 462). Heidelberg: Springer Berlin.

Li, J., Huang, Y., & Niu, X. (2016). A branch population genetic algorithm for dual-resource constrained job shop scheduling problem. *Computers & Industrial Engineering*, *102*, 113–131.

Liu, S. Q., Kozan, E., Masoud, M., Zhang, Y., & Chan, F. T. (2018). Job shop scheduling with a combination of four buffering constraints. *International Journal of Production Research*, *56*(9), 3274–3293.

Liu, Q., Zhan, M., Chekem, F. O., Shao, X., Ying, B., & Sutherland, J. W. (2017). A hybrid fruit fly algorithm for solving flexible job-shop scheduling to reduce manufacturing carbon footprint. *Journal of Cleaner Production*, *168*, 668–678.

Mascis, A., & Pacciarelli, D. (2002). Job-shop scheduling with blocking and no-wait constraints. *European Journal of Operational Research*, *143*(3), 498–517.

Mastrolilli, M., & Gambardella, L. M. (2000). Effective neighbourhood functions for the flexible job shop problem. *Journal of Scheduling*, *3*(1), 3–20.

Meloni, C., Pranzo, M., & Samà, M. (2022). Evaluation of VaR and CVaR for the makespan in interval valued blocking job shops. *International Journal of Production Economics*, *247*.

Mogali, J. K., Barbulescu, L., & Smith, S. F. (2021). Efficient primal heuristic updates for the blocking job shop problem. *European Journal of Operational Research*, *295*(1), 82–101.

Mogali, J. K., Kinable, J., Smith, S. F., & Rubinstein, Z. B. (2021). Scheduling for multi-robot routing with blocking and enabling constraints. *Journal of Scheduling*, *24*, 291–318.

Mokhtari, H., & Dadgar, M. (2015). Scheduling optimization of a stochastic flexible job-shop system with time-varying machine failure rate. *Computers & Operations Research*, *61*, 31–45.

Muritiba, A. E. F., Rodrigues, C. D., & da Costa, F. A. (2018). A path-relinking algorithm for the multi-mode resource-constrained project scheduling problem. *Computers & Operations Research*, *92*, 145–154.

Nguyen, S., Thiruvady, D., Ernst, A. T., & Alahakoon, D. (2019). A hybrid differential evolution algorithm with column generation for resource constrained job scheduling. *Computers & Operations Research*, *109*, 273–287.

Novas, J. M. (2019). Production scheduling and lot streaming at flexible job-shops environments using constraint programming. *Computers & Industrial Engineering*, *136*(October 2018), 252–264.

Novas, J. M., & Henning, G. P. (2014). Integrated scheduling of resource-constrained flexible manufacturing systems using constraint programming. *Expert Systems with Applications*, *41*(5), 2286–2299.

Oddi, A., Rasconi, R., Cesta, A., & Smith, S. F. (2012). Iterative improvement algorithms for the blocking job shop. *ICAPS 2012 - Proceedings of the 22nd International Conference on Automated Planning and Scheduling*, 199–206.

Palpant, M., Artigues, C., & Michelon, P. (2004). LSSPER: Solving the resource-constrained project scheduling problem with large neighbourhood search. *Anal. Oper. Res.*, *131*(1), 237–257.

Patterson, J., Talbot, F., Słowiński, R., & Węglarz, J. (1990). Computational experience with a backtracking algorithm for solving a general class of precedence and resource-constrained scheduling problems. *European Journal of Operational Research*, *49*, 68–79.

Pisinger, D., & Ropke, S. (2007). A general heuristic for vehicle routing problems. *Computers & Operations Research*, *34*(8), 2403–2435.

Pranzo, M., & Pacciarelli, D. (2016). An iterated greedy metaheuristic for the blocking job shop scheduling problem. *J. Heuristics*, *22*(4), 587–611.

Prata, B. A., Abreu, L. R., & Nagano, M. S. (2024). Applications of constraint programming in production scheduling problems: A descriptive bibliometric analysis. *Results in Control and Optimization*, *14*, Article 100350.

Rifai, A. P., Nguyen, H. T., & Dawal, S. Z. M. (2016). Multi-objective adaptive large neighborhood search for distributed reentrant permutation flow shop scheduling. *Applied Soft Computing*, *40*, 42–57.

Ropke, S., & Pisinger, D. (2006). An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. *Transportation Science*, *40*(4), 455–472.

Sacramento, D., Solnon, C., & Pisinger, D. (2020). Constraint programming and local search heuristic: A matheuristic approach for routing and scheduling feeder vessels in multi-terminal ports. *Oper. Res. Forum*, *1*.

Schrimpf, G., Schneider, J., Stamm-Wilbrandt, H., & Dueck, G. (2000). Record breaking optimization results using the ruin and recreate principle. *Journal of Computational Physics*, *159*(2), 139–171.

Sha, Y., Zhang, J., & Cao, H. (2021). Multistage stochastic programming approach for joint optimization of job scheduling and material ordering under endogenous uncertainties. *European Journal of Operational Research*, *290*(3), 886–900.

Shaw, P. (1998). Using constraint programming and local search methods to solve vehicle routing problems. In M. Maher, & J.-F. Puget (Eds.), *Principles and practice of constraint programming — CP98* (pp. 417–431). Berlin, Heidelberg: Springer Berlin Heidelberg.

Słowiński, R. (1980). Two approaches to problems of resource allocation among project activities – a comparative study. *Journal of the Operational Research Society*, *31*(8), 711–723.

Słowiński, R. (1981). Multiobjective network scheduling with efficient use of renewable and nonrenewable resources. *European Journal of Operational Research*, *7*, 265–273.

Soares, L. C. R., & Carvalho, M. A. M. (2020). Biased random-key genetic algorithm for scheduling identical parallel machines with tooling constraints. *European Journal of Operational Research*, *285*(3), 955–964.

Song, R., Lau, H. C., Luo, X., & Zhao, L. (2022). Coordinated delivery to shopping malls with limited docking capacity. *Transportation Science*, *56*(2), 501–527.

Trabelsi, W., Sauvey, C., & Sauer, N. (2012). Heuristics and metaheuristics for mixed blocking constraints flowshop scheduling problems. *Computers & Operations Research*, *39*(11), 2520–2527.

Trojet, M., H'Mida, F., & Lopez, P. (2011). Project scheduling under resource constraints: Application of the cumulative global constraint in a decision support framework. *Computers & Industrial Engineering*, *61*(2), 357–363.

Van Peteghem, V., & Vanhoucke, M. (2014). An experimental investigation of metaheuristics for the multi-mode resource-constrained project scheduling problem on new dataset instances. *European Journal of Operational Research*, *235*(1), 62–72.

Waldherr, S., & Knust, S. (2017). Decomposition algorithms for synchronous flow shop problems with additional resources and setup times. *European Journal of Operational Research*, *259*(3), 847–863.

Węglarz, J. (1981). Project scheduling with continuously-divisible, doubly constrained resources. *Management Sci.*, *27*(9), 1040–1053.

Wong, T. C., Chan, F. T., & Chan, L. Y. (2009). A resource-constrained assembly job shop scheduling problem with lot streaming technique. *Computers & Industrial Engineering*, *57*(3), 983–995.

Yaurima, V., Burtseva, L., & Tchernykh, A. (2009). Hybrid flowshop with unrelated machines, sequence-dependent setup time, availability constraints and limited buffers. *Computers & Industrial Engineering*, *56*(4), 1452–1463.

Yuan, Y., Xu, H., & Yang, J. (2013). A hybrid harmony search algorithm for the flexible job shop scheduling problem. *Applied Soft Computing*, *13*(7), 3259–3272.

Yunusoglu, P., & Yildiz, S. T. (2023). Solving the flexible job shop scheduling and lot streaming problem with setup and transport resource constraints. *International Journal of Systems Science*, *10*.

Zeballos, L. J., Quiroga, O. D., & Henning, G. P. (2010). A constraint programming model for the scheduling of flexible manufacturing systems with machine and tool limitations. *Engineering Applications of Artificial Intelligence*, *23*(2), 229–248.

Zhang, S., Du, H., Borucki, S., Jin, S., Hou, T., & Li, Z. (2021). Dual resource constrained flexible job shop scheduling based on improved quantum genetic algorithm. *Machines*, *9*(6).