INTERNATIONAL TRANSACTIONS IN OPERATIONAL RESEARCH

WILEY

INTERNATIONAL TRANSACTIONS IN OPERATIONAL RESEARCH

Intl. Trans. in Op. Res. 0 (2026) 1–27
DOI: 10.1111/itor.70177

Exact and metaheuristic approaches to minimizing makespan in parallel machine scheduling with conflicting jobs

Roberto Maria Rosati[a,b,*], Dinh Quy Ta[c], Minh Hoàng Hà[c] and Andrea Schaerf[d]

[a] *Research Institute for Supply Chain Management, WU Vienna University of Economics and Business, Welthandelsplatz 1, Vienna 1020, Austria*

[b] *Department of Economics, Analytics and Operations Research, University of Klagenfurt, UniversitätsstraBe 65-67, Klagenfurt am Wörthersee 9020, Austria*

[c] *SLSCM and CADA, Faculty of Data Science and Artificial Intelligence, College of Technology, National Economics University, Hanoi, Vietnam*

[d] *DPIA, University of Udine, via delle Scienze 206, Udine I-33100, Italy*

E-mail: robertomaria.rosati@wu.ac.at[Rosati]; quydt@neu.edu.vn[Ta]; hoanghm@neu.edu.vn[Hà]; andrea.schaerf@uniud.it[Schaerf]

Received 6 May 2025; received in revised form 4 February 2026; accepted 5 February 2026

Abstract

We address the scheduling conflicting jobs on parallel identical machines problem with makespan minimization, a classical and computationally challenging variant of parallel machine scheduling. We develop and evaluate three distinct solution methodologies: a novel constraint programming (CP) formulation, and two metaheuristics: a multi-neighborhood simulated annealing that relies on an implicit solution representation and a greedy decoder, and a CP-based large neighborhood search that employs operators specifically tailored for the problem. The methods are tuned using statistically rigorous procedures and compared to highlight the strengths and weaknesses of each approach. For this purpose, we introduce and make publicly available a novel, challenging dataset, appropriately divided into training and validation instances. This dataset supports our experiments and provides a foundation for future benchmarking. Computational results show that the proposed CP formulation significantly outperforms an existing CP method, proving optimality on several instances within short computing times. Meanwhile, the MNSA metaheuristic consistently delivers high-quality solutions, especially on instances where the exact method has difficulty converging, with more consistent gaps in the presence of high conflict rates.

Keywords: scheduling; conflicting jobs; parallel machines; makespan minimization; constraint programming; metaheuristics

*Corresponding author.

1. Introduction

Machine scheduling refers to problems where the goal is to allocate jobs to machines in a way that satisfies certain constraints and objectives. These problems are significant in manufacturing and logistics, where efficient scheduling can lead to improved productivity, reduced costs, and better resource utilization. Common objectives include minimizing makespan, minimizing total completion time, and balancing the load across machines.

Among machine scheduling problems, we specifically consider the problem known as scheduling conflicting jobs on parallel identical machines (SCJPIM), which has recently gained attention in the scientific literature (Zinder et al., 2021; Ta et al., 2024). The SCJPIM is at the basis of many real-world problems when an external resource cannot be shared among jobs. Applications of the problem include traffic signal control (Irani and Leung, 1996), job scheduling on parallel CPUs, where certain jobs are memory-critical and cannot be scheduled together, called mutual exclusion scheduling (Baker and Coffman, 1996), and train dispatching, in which trains share resources (railway tracks and crew), that cannot be used in the same time slot by two locomotives (Lamorgese et al., 2018).

In this study, we focus on the makespan minimization (MM), thus addressing the specific problem called scheduling conflicting jobs on parallel identical machines problem with makespan minimization (MM-SCJPIM). According to the classification of scheduling problems in the form $\alpha|\beta|\gamma$ proposed by Graham et al. (1979) and Blazewicz et al. (1983), we classify the problem as $P|conflict, G|C_{\max}$. The MM-SCJPIM is different from the previously studied parallel machines with conflicts (PMC) problem, which considers conflicts as the impossibility of executing two jobs on the same machine, regardless of their execution time (Kowalczyk and Leus, 2017; Bianchessi and Tresoldi, 2021). Conversely, the MM-SCJPIM considers conflicts as the impossibility of executing jobs in parallel, regardless of the machine assignment. Due to this substantial difference, we consider it important to use a different notation than the one used in the past, to avoid ambiguity and to make clear that the two problems are not the same.

A review of existing work reveals that solution approaches for MM-SCJPIM have typically centered on analyzing specific graph structures to determine complexity, finding approximation schemes, or devising exact polynomial algorithms for those limited scenarios. This focus on special cases leaves a significant gap in the development of exact or heuristic algorithms for general conflict graphs, motivating further research efforts in this domain.

To address this gap in solution methods for the general MM-SCJPIM problem, we propose three distinct solution approaches. First, we present a new constraint programming (CP) model, providing an improved exact method over that of Ta et al. (2024). Second, we design a multi-neighborhood simulated annealing (MNSA). This metaheuristic operates on a global job ordering representation and utilizes Move, Swap, and 2-Opt neighborhoods for effective exploration. Finally, we propose a CP-based large neighborhood search (LNS), a hybrid approach that leverages the efficiency of our CP model for the insertion phase.

In order to set up the ground for a comprehensive comparison, we developed a randomized instance generator that aims to cover evenly the feature space of the problem instances. Using our generator, we created a dataset composed of 30 validation instances and 300 training instances. The

size of the instances spans from 45 to 300 jobs, with varying conflict rates, number of machines, and distributions of job processing times. We compare the three methods with each other and with an existing CP formulation from the literature on the new dataset. The new CP formulation improves both in solution quality and times and finds several optimal solutions on instances with small conflict rates, while the MNSA is the best performing method on instances with medium and high conflict rates.

This paper makes several key contributions to the MM-SCJPIM problem, summarized hereafter:

- We introduce a novel CP model that outperforms the model proposed in Ta et al. (2024).
- We present an MNSA approach, employing an implicit solution representation, strategically defined neighborhoods, and a greedy solution decoder.
- We propose an LNS method for the MM-SCJPIM that uses a list of job start times as the solution representation, with CP-based insertion techniques for solution reconstruction.
- To rigorously evaluate these methods, we create a new parametric instance generator and we share a diverse and challenging dataset of training and validation instances.
- We perform a comparative analysis across the three methods, aimed at identifying the strengths and weaknesses of each based on the instance features.

The remaining part of this paper is organized as follows. Section 2 surveys related research. The MM-SCJPIM problem, along with the benchmark instances used, is described in Section 3. The proposed solution methods are described in Section 4. Experimental results and analyses are presented in Section 5. Finally, Section 6 concludes our research. A preliminary and abridged version of this work appeared in Rosati et al. (2024), which considers only MNSA and a preliminary version of CP, both tested on a less challenging dataset.

2. Related work

We briefly review the literature on scheduling problems with conflict constraints, with the main works summarized in Table 1.

Several works address different kinds of conflicts in scheduling. Kowalczyk and Leus (2017) addressed the PMC problem, where the goal is to schedule a set of independent jobs on multiple identical machines, minimizing the makespan while ensuring that conflicting jobs are assigned to different machines; the paper introduces an indirect method that uses binary search on the makespan and solves related bin packing problem with conflicts feasibility instances via branch-and-price (B&P), supported by various heuristics and feasibility checks. Subsequently, Bianchessi and Tresoldi (2021) developed the first stand-alone B&P algorithm that directly optimizes the PMC problem, eliminating the need for the binary search structure and complex auxiliary components used in the prior work. This direct approach achieves competitive performance primarily through a novel hierarchical branching scheme designed to maintain subproblem structure efficiently throughout the search tree.

Mallek et al. (2019) explore scheduling conflicting identical jobs on uniform machines (SCU), a problem sharing a conflict model with PMC but featuring identical jobs and uniform machines.

Table 1
Summary of literature on scheduling under conflict constraints

Citation	Objective function	How conflict operates	Machine environment	Method(s)
Kowalczyk and Leus (2017)	Makespan	Conflicting jobs cannot run on same machine	Identical machines	Indirect B&P + binary search; heuristics
Bianchessi and Tresoldi (2021)	Makespan	Conflicting jobs cannot run on same machine	Identical machines	Direct B&P
Mallek et al. (2019)	Makespan	Conflicting jobs cannot run on same machine	Uniform machines	MILP; Heuristics
Mallek and Boudhar (2024)	Makespan	Conflicting jobs cannot run on same machine	Uniform machines	Hybrid MILP and bound heuristic; complexity results
Buchem et al. (2022)	Makespan	Conflicting machines cannot run overlapping blocking times	Identical machines	Approximation algorithm
Tellache and Boudhar (2017)	Makespan	Conflicting jobs cannot process concurrently	Open shop	Heuristic; Polynomial-time solvable cases
Tellache and Boudhar (2018)	Makespan	Conflicting jobs cannot process concurrently	Flow shop	Heuristic; Polynomial-time solvable cases
Baker and Coffman (1996)	Makespan	Conflicting unit jobs cannot process concurrently	Identical machines	Polynomial-time cases; Approximation algorithm
Even et al. (2009)	Makespan	Conflicting jobs cannot process concurrently	Identical machines	Approximation algorithm; Online algorithm
Bendraouche and Boudhar (2012)	Makespan	Agreeing jobs can process concurrently	Identical machines	Heuristics; Polynomial-time cases
Hà et al. (2019)	Max weighted on-time completion	Conflicting jobs cannot process concurrently	Identical machines	MILPs; Polynomial-time cases
Zinder et al. (2021)	Max weighted on-time completion	Conflicting jobs cannot process concurrently	Identical machines	MILPs; metaheuristic
Tresoldi (2021)	Max weighted on-time completion	Conflicting jobs cannot process concurrently	Identical machines	Flow-based MILP; Heuristic
Ta et al. (2024)	Makespan; Min weighted sum completion time; Max weighted on-time completion	Conflicting jobs cannot process concurrently	Identical machines	Three MILPs; CP model; Binary search
Our work	Makespan	Conflicting jobs cannot process concurrently	Identical machines	CP model; Two metaheuristics

The paper establishes SCU's NP-hardness under specific graph structures and offers mixed integer linear programming (MILP) models and heuristics tailored to this problem setting. More recently, Mallek and Boudhar (2024) expanded on this SCU research by providing a more detailed analysis of its computational complexity, particularly focusing on various conflict graph structures (split graphs, interval graphs, forests, etc.), and proposing a hybrid MILP approach. Buchem et al. (2022)

examine the problem of scheduling with machine conflicts (SMC), where jobs have pre- and post-processing blocking times, and the goal is to minimize the makespan while ensuring that jobs on conflicting machines do not have overlapping blocking times; the paper demonstrates that SMC is hard to approximate, but provides approximation algorithms based on independent sets and polynomial-time solutions for specific graph structures.

The challenges of job concurrency and conflict have been explored in various scheduling contexts. For instance, Tellache and Boudhar (2017) address the open shop scheduling problem with conflict graphs (OSC), where conflicting jobs cannot be processed simultaneously on different machines, focusing on minimizing the makespan. The paper establishes the NP-hardness of several OSC variants and identifies polynomial-time solvable cases. It also proposes heuristic algorithms, based on matchings in bipartite graphs and insertion techniques combined with beam search, along with lower bounds to evaluate their performance. Following this, Tellache and Boudhar (2018) tackle the flow shop scheduling problem with conflict graphs (FSC), where conflicting jobs cannot be processed simultaneously on different machines. The research establishes NP-hardness for specific FSC variants and proposes heuristic algorithms and lower bounds for the general problem. Caimi et al. (2011) tackle the complex problem of generating conflict-free train schedules on a microscopic railway infrastructure model. To overcome the weak relaxations of traditional conflict graph models, the authors introduce a resource-constrained multicommodity flow model using a resource tree conflict graph. By explicitly considering track sections and blocking times, the model efficiently identifies conflict cliques. The objective is to maximize the number of scheduled trains while respecting conflict constraints and optimizing schedule quality, demonstrating significant computational improvements using real-world data.

The challenge of conflict constraints, a key element of the SCJPIM problem, arises in several real-world scheduling scenarios. For instance, Grimes (1970) addressed this in the context of conference scheduling, where presentations that the same attendees want to hear cannot be scheduled concurrently. The paper's approach used a conflict matrix derived from attendee preferences to identify sets of presentations that must be scheduled at different times, aiming to reduce meeting length and improve attendee satisfaction. A more recent application is the train dispatching problem, as exemplified by the DISPLIB2025 competition. While not explicitly a SCJPIM, effective management of dense railway traffic inherently requires resolving conflicts arising from trains competing for shared resources like track sections, and the competition challenges the development of algorithms that can handle such conflicts in real-time. Similarly, the ROADEF 2020 challenge focused on outage maintenance planning for electricity transmission networks, a problem where some maintenance interventions are mutually exclusive due to operational risks, necessitating careful scheduling to avoid grid failures.

Since SCJPIM includes the classic makespan problem as a special case (when $E = \emptyset$), it is also NP-hard. Research has often focused on understanding the complexity and approximability based on job characteristics (like unit processing times) and the structure of the conflict graph. For the specific case where all jobs have unit processing times ($p_j = 1$ for all j), the problem is often termed *mutual exclusion scheduling*. Baker and Coffman (1996) showed that it remains strongly NP-hard if the number of machines $M \geq 3$. However, for two machines ($M = 2$), they provided a polynomial-time algorithm based on finding a maximum matching in the complement of the conflict graph. This positive result for $M = 2$ was later extended by Even et al. (2009) to include

jobs with lengths 1 or 2. If job lengths can be {1, 2, 3}, the problem becomes NP-hard again, even for $M = 2$ (Bendraouche and Boudhar, 2012). Baker and Coffman (1996) also established an important equivalence between SCJPIM with unit jobs and the M-bounded coloring problem. Leveraging this connection, polynomial-time algorithms have been developed for unit-job SCJPIM when the conflict graph belongs to specific classes, including trees/forests (Baker and Coffman, 1996), split graphs and complements of interval graphs (Lonc, 1992), cographs and bipartite graphs (Bodlaender and Jansen, 1993), graphs with constant treewidth (Kaller et al., 1995), and line graphs (Alon, 1983). Conversely, the unit-job SCJPIM problem is known to be NP-hard even for specific structured graphs like permutation graphs (Jansen, 2003), complements of comparability graphs (Lonc, 1992), and interval graphs (Bodlaender and Jansen, 1993). Regarding approximability for SCJPIM, the unit job case can be related to the M-set cover problem. When M is constant, an $(\mathcal{H}_M - 1/2)$-approximation exists (Duh and Fürer, 1997) (where $\mathcal{H}_M$ is the Mth harmonic number). However, if M is part of the input, the problem is hard to approximate within a factor of $n^{1-\epsilon}$ for any $\epsilon > 0$, under standard complexity assumptions (Zuckerman, 2006). For the case where job lengths are restricted to {1, 2, 3} and $M = 2$, a 4/3-approximation algorithm is known (Even et al., 2009).

Several studies explore exact and heuristic methods for the SCJPIM under the maximum weighted job completion constraint. Hà et al. (2019) show that the problem is polynomially solvable when there are two machines and the conflict graph is bipartite. In terms of exact methods for the problem with the general graph, they proposed two MILP models. Zinder et al. (2021) also tackle the problem, focusing on maximizing the total weight of on-time jobs on parallel machines subject to a conflict graph and a common deadline. Their contributions include a polynomial-time algorithm for unit execution time jobs on two machines, an analysis of approximation algorithm performance for scenarios with more than two machines, and the development of new MILP formulations and an iterated variable neighborhood search heuristic for the case of arbitrary job processing times. In addition to these approaches, Tresoldi (2021) proposes a flow-based MILP formulation and a simple iterated local search (ILS) for the maximum weighted job completion SCJPIM problem, evaluating their methods on a generated dataset with up to 64 jobs.

More recently, Ta et al. (2024) propose three MILP models and one CP model for the SCJPIM, considering three objective functions: makespan minimization, weighted completion time minimization, and maximum weighted job completion. To improve the speed of exact methods based on the MILP formulations, they further propose a binary search scheme for the makespan minimization problem. Computational results on randomly generated instances with up to 90 jobs demonstrate that the CP model achieves the best solution quality among their proposed approaches for these instance sizes.

3. The scheduling conflicting jobs on parallel identical machines problem

In the following section, we begin by formally defining the key elements of the SCJPIM problem, which extends the classical identical parallel machine scheduling framework. We then provide a graphical example to illustrate the problem's structure, followed by a detailed discussion of the instance generation process.

3.1. Problem description

Let $\mathcal{J} = \{1, \ldots, J\}$ be a set of J jobs, and $\mathcal{M} = \{1, \ldots, M\}$ be a set of M parallel identical machines. The problem requires executing all the jobs in $\mathcal{J}$ on the machines in $\mathcal{M}$. We also say that M is the capacity, because it is the maximum number of jobs that can be executed in parallel. Each job $j \in \mathcal{J}$ requires p_j units of processing time, is executed on only one of the machines, and must be processed by this machine for exactly p_j units continuously until its completion (no preemption is allowed). The problem involves deciding the starting time s_j of job j satisfying the constraints mentioned above. As the schedule is non-preemptive, we can compute the end time as $e_j = s_j + p_j$.

A conflict graph $\mathcal{G}(\mathcal{J}, \mathcal{C})$ is used to represent the conflict relationships between jobs. The set of nodes is the set of jobs, and each (undirected) edge $(j, k) \in \mathcal{C}$ corresponds to a pair of jobs in conflict with each other. In particular, if $(j, k) \in \mathcal{C}$ then the processing time of j and k must not overlap in the schedule, that is, either $s_j \geq e_k$ or $s_k \geq e_j$. Moreover, at most one job can be assigned to each machine at a time. We say that the conflict rate c is the ratio between the number of conflicts $|\mathcal{C}|$ and the number of edges $|\mathcal{E}|$ in the complete graph $\mathcal{G}(\mathcal{J}, \mathcal{E})$.

As already mentioned, we consider the MM-SCJPIM problem, in which the objective is to assign all jobs on given machines in a way that the *makespan*, which is the maximum e_j over all jobs $j \in \mathcal{J}$, is minimized.

In the following, Equation (1) formalizes the objective, and Equations (2)–(4) formalize, respectively, the preemption, conflicts, and capacity constraints. We add the binary matrix a such that $a_j = m$ if job j is executed on machine m.

$$\text{obj:} \quad \min \quad \max_{j \in \mathcal{J}} e_j \tag{1}$$

$$\text{s.t.:} \quad e_j = s_j + p_j \qquad \forall j \in \mathcal{J}, \tag{2}$$

$$s_j \geq e_k \vee s_k \geq e_j \qquad \forall (j, k) \in \mathcal{C}, \tag{3}$$

$$\neg(a_{jm} = 1 \wedge a_{km} = 1) \vee (s_j \geq e_k \vee s_k \geq e_j) \qquad \forall j, k \in \mathcal{J}, m \in \mathcal{M}, \tag{4}$$

$a_j = m$ if job j is executed on machine m. Then the formula would be "$a_j \neq a_k$ or $s_j \geq e_k$ or $s_k \geq e_j$".)

We point out that, given the fact that machines are identical, Constraint (4) can be more conveniently treated in terms of capacity, stating that at any point in time at most M jobs are executed.

Figure 1 shows a toy instance of MM-SCJPIM with $J = 6$ and $M = 3$, along with its solution. The makespan is given by the end time of job 3 on machine M1. Note that jobs 2 and 3 cannot start earlier because of their conflicts. The solution shown in the figure is also optimal for this toy instance, although it is not the only optimal solution.

3.2. Generator and datasets

We designed a parametric instance generator that is used to create a novel, more diverse, and challenging dataset. The new generator overcomes the limitations of a previous dataset from Rosati et al. (2024), in which instances had a more regular generation pattern that resulted in low coverage of the feature space. Five parameters guide the instance generation: number of jobs J, number of

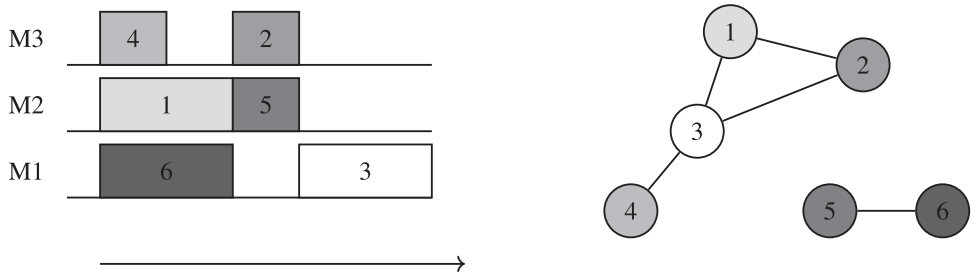


Fig. 1. Example of an MM-SCJPIM solution. Left: Gantt chart, right: conflict graph.

machines M, expected conflict rate $\hat{c}$, lower bound for the job duration p_{min}, and range for the job duration r. The actual duration of each job is drawn uniformly during the generation of the instance in the range $[p_{min}, p_{min} + r]$. Parameter $\hat{c}$ defines the probability of two jobs being in conflict. Given the stochastic nature of the generation procedure, the actual conflict rate c of the instance might slightly differ from the expected conflict rate $\hat{c}$. The resulting conflict rate c relates to the number of conflicts $|\mathcal{C}|$ through the following formula: $c = |\mathcal{C}|/(J(J-1)/2)$.

In order to create instances that are well-distributed along the feature space, the generation procedure employs the Hammersley point set (Hammersley and Handscomb, 1964) to choose pseudo-randomly the points in the feature space. The advantage of using the Hammersley point set for instance generation is that it guarantees that all areas of the selected regions of the feature space considered in our experiments are evenly represented (Viana, 2016). To guarantee that varying conflict rates are equally represented, we do this separately for low, medium, and high conflict rates. In summary, the regions of the feature space that we aim to cover in the experimental results are bounded by the following values used for instance generation:

- $J \in [30, 300]$,
- $M \in [2, 12]$,
- $\hat{c} \in \{[0.05, 0.25], [0.40, 0.60], [0.70, 0.90]\}$, and
- $p_j \in [\alpha, \alpha + 10^{\beta}]$, with $\alpha \in [1, 100]$, $\beta \in [1, 3]$.

As customary, in order to avoid overtuning of parameters on the same instances used for validation, we generated a dataset composed of 300 training and 30 validation instances, with the respective tasks of being used for algorithm tuning and for comparison. Table 2 shows the features of the validation instances. The whole dataset is available online at https://github.com/iolab-uniud/SCJPIM.

4. Solution methods

For the solution of the MM-SCJPIM, we propose three distinct search methods: a CP formulation, which is capable of finding the optimal solution, if enough time is given, an MNSA, and an LNS, which are both metaheuristic approaches based on local search, but that exploit different strategies

Table 2
Features of the validation instances

#	J	M	c	α	β
01	173	6	0.07	21	1.28
02	109	9	0.08	41	1.57
03	236	4	0.10	60	1.86
04	77	7	0.11	80	2.14
05	204	10	0.13	5	2.43
06	141	5	0.14	25	2.71
07	268	8	0.16	45	1.04
08	188	3	0.19	84	1.61
09	125	6	0.20	9	1.90
10	252	9	0.22	29	2.18
11	276	11	0.40	76	1.69
12	69	6	0.41	96	1.98
13	260	3	0.46	41	2.84
14	101	6	0.47	61	1.18
15	228	9	0.49	81	1.45
#	J	M	c	α	β
16	165	4	0.50	6	1.73
17	292	7	0.52	26	2.02
18	113	8	0.56	85	2.88
19	240	11	0.58	10	1.20
20	81	3	0.59	30	1.49
21	256	8	0.70	73	1.53
22	97	11	0.71	93	1.82
23	224	4	0.73	18	2.10
24	161	7	0.74	37	2.39
25	57	5	0.77	77	2.96
26	89	9	0.83	42	1.86
27	216	12	0.85	62	1.15
28	153	3	0.86	82	2.43
29	280	6	0.88	7	2.72
30	73	9	0.89	26	1.04

for the exploration of the search space. Moreover, the neighborhoods designed for both methods are also slightly different.

4.1. Constraint programming

In this section, we first present the CP model proposed by Ta et al. (2024). Both the model from Ta et al. (2024) and the new model proposed in this study are formulated using IBM ILOG CP Optimizer. This solver offers specialized features specifically designed for scheduling problems, such as interval variables and dedicated scheduling constraints. The use of interval variables enables an effective representation of the periods during which tasks are active, characterized by start, end, and

duration attributes that directly correspond to the decision variables of the problem. Both models use the interval variable Itv_i that represents the time interval during which job $i \in \mathcal{J}$ is processed, with a fixed duration p_i. In this problem, all jobs must be processed, and thus the variables Itv_i ($\forall i \in \mathcal{J}$) are therefore not optional.

The formulation of the CP model introduced by Ta et al. (2024) is presented as follows:

$$\text{(CP1)} \quad \text{Minimize} \quad \max_{i \in \mathcal{J}} \text{endOf}(Itv_i), \tag{5}$$

$$\text{s.t.} \quad \text{alternative}(Itv_i, ItvAlt_{im} : m \in \mathcal{M}) \qquad \forall i \in \mathcal{J}, \tag{6}$$

$$Seq_m = \{ItvAlt_{im} \mid i \in \mathcal{J}\} \quad \forall m \in \mathcal{M}, \tag{7}$$

$$\text{noOverlap}(Seq_m) \qquad \forall m \in \mathcal{M}, \tag{8}$$

$$\text{endBeforeStart}(Itv_i, Itv_j) \quad \text{OR} \qquad \forall (i, j) \in \mathcal{C},$$
$$\text{endBeforeStart}(Itv_j, Itv_i). \tag{9}$$

The objective function (5) seeks to minimize the makespan, defined as the maximum completion time among all jobs $i \in \mathcal{J}$. Here, the function endOf() is used to define the completion time of the interval variables. Constraints (6) use the alternative constraint provided by CP optimizer. It ensures that for each job i, the interval variable Itv_i is present if and only if exactly one of the optional interval variables $ItvAlt_{im}$ (representing job i running on machine m) is selected from the set of available machines $\mathcal{M}$. This effectively assigns each job to exactly one machine. Constraints (7) define the sequence variable Seq_m (implicitly defined for each machine $m \in \mathcal{M}$), which consists of job interval variables assigned to the machine. Constraints (8) enforce that jobs assigned to the same machine do not overlap in time. Finally, constraints (9) handle the job conflicts. For each pair of conflicting jobs $(i, j) \in \mathcal{C}$, the constraints enforce mutual exclusion by specifying that either the execution interval of job i must precede that of job j (endBeforeStart(Itv_i, Itv_j)), or conversely (endBeforeStart(Itv_j, Itv_i)). The function endBeforeStart() in the constraint expresses the relationship between its two arguments: the first interval variable must terminate before the second interval variable starts.

We now introduce our proposed CP model formulation:

$$\text{(CP2)} \quad \text{Minimize} \quad \max_{i \in \mathcal{J}} \text{endOf}(Itv_i), \tag{10}$$

$$\text{s.t.} \quad \sum_{i \in \mathcal{J}} \text{pulse}(Itv_i, 1) \leq M, \tag{11}$$

$$\text{endBeforeStart}(Itv_i, Itv_j) \quad \text{OR} \qquad \forall (i, j) \in \mathcal{C},$$
$$\text{endBeforeStart}(Itv_j, Itv_i). \tag{12}$$

The objective function is represented by expression (10). Next, constraint (11) guarantees that no more than M jobs are executing concurrently. It utilizes the pulse() method on a cumulative function, which signifies that one machine is consumed whenever a job's interval variable is active. Finally, constraints (12) ensure that conflicting jobs cannot be processed concurrently.

The main difference between CP1 and our CP2 model lies in handling machine capacity. CP1 explicitly assigns jobs to machines using alternative and enforces noOverlap on each machine

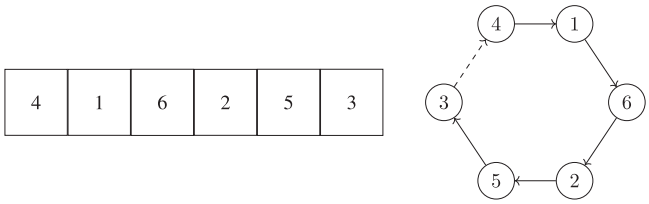


Fig. 2. Solution representation as a vector and as a route.

separately. In contrast, CP2 uses a single, global cumulative constraint (pulse) to ensure the total number of concurrently running jobs never exceeds the machine limit M. The latter approach avoids the symmetry present in CP1's explicit job-to-machine assignments. Since the machines are identical, solutions in CP1 that differ merely by which specific machine executes which job are treated as distinct, whereas they correspond to the same state within CP2's cumulative resource model.

The CP model described above guarantees optimality on smaller instances but does not scale well to larger ones. Therefore, in what follows, we introduce two metaheuristics designed to address this limitation.

4.2. Multi-neighborhood simulated annealing

Compared to single-neighborhood local search, the composition of multiple local search neighborhoods, such as in our MNSA, increases the connectivity in the search space and makes being trapped in bad local minima less likely. This approach, used within a stochastic framework, has proven successful on a wide range of challenging combinatorial problems (see, e.g., Li and Hao, 2022; Lü et al., 2023; Rosati and Schaerf, 2024).

Hereafter, we describe the components of our MNSA method: the *solution representation*, the *search space*, the *initial solution strategy*, the *neighborhood relations* and their composition, the *exploration criterion*, and the simulated annealing (SA) *guiding scheme* that dictates the *move acceptance criterion*. In following a component-based view, we consider a metaheuristic not as a monolithic procedure but as a composition of algorithmic components that may have multiple alternative instantiations. This is in line with previous work on SA (Franzin and Stützle, 2019), LNS (Turkeš et al., 2021), and Tabu Search (Da Ros and Di Gaspero, 2024).

4.2.1. Solution representation and search space

We represent the solution as the vector of the global ordering of the jobs. The search space is composed of all possible permutations of this vector. Our solution representation can be interpreted as a route connecting all jobs in sequence, similarly to the traveling salesman problem (TSP), a well-explored parallelism since early studies (Picard and Queyranne, 1978).

Figure 2 shows the solution of the example in Fig. 1 in the form of a vector with the global order of jobs and as a single route connecting all jobs. The edge connecting the last job (3) with the first one (4) is dashed because the route is not closed. However, considering it as a closed loop can be exploited to enlarge the search space for neighborhood exploration, as the next section describes.

4.2.2. Solution decoder

To reconstruct the actual solution (i.e., the starting times of the jobs), we employ a forward greedy procedure as a decoder. The procedure takes in input the global ordering and determines the starting times by inserting every job at the earliest feasible time (with respect to conflicts and capacity) that is greater or equal to the start time of the previous job in the global ordering. Thus, only feasible solutions are explored in the search space. As previously mentioned, due to the machines being identical, we do not keep track of which job is assigned to which machine, but we just ensure that the total capacity constraint is satisfied at any moment in the schedule. This has the effect of simplifying the search space and making the method more efficient.

An advantage of an implicit solution representation as a permutation of jobs is that we can employ problem-agnostic neighborhoods, which could be potentially adapted to other scheduling problems. For instance, a similar approach has demonstrated effectiveness on the home healthcare routing and scheduling problem (Ceschia et al., 2026). While this approach requires decoding the solution after every local search move to derive the actual solution, the complexity of the decoding procedure is linear to the number of jobs. Moreover, the solution needs to be reconstructed only from the first (in the global ordering) of the jobs involved in the move, which implies that a part of the solution does not have to be recomputed. Finally, we make use of differential cost evaluation to avoid copying and recomputing the solution at every iteration, but just when moves are accepted. For all these reasons, this approach is scalable and readily portable to other problem formulations.

4.2.3. Neighborhood relations

We define three neighborhood relations: Move, Swap, and 2-Opt. The third one is adapted from the TSP literature. We denote the position of job j as $\phi(j)$. The three neighborhoods are defined as follows:

Move$\langle j, \rho\rangle$:moves job j from its current position $\phi(j)$ to a new position ρ. If $\rho > \phi(j)$ then the jobs in $\{\phi(j)+1, \ldots, \rho\}$ are shifted backwards by one position, forward otherwise.
Precondition: $\phi(j) \neq \rho$.

Swap$\langle j, k\rangle$:swaps the positions of j and k. Job j takes position $\phi(k)$ and job k takes position $\phi(j)$.
Precondition: $|\phi(j) - \phi(k)| \geq 2$.

2-Opt$\langle \rho_1, \rho_2\rangle$:reverts the order of jobs in positions $\{\rho_1+1, \ldots, \rho_2\}$. If $\rho_1 > \rho_2$ the move includes the edge between the last and the first job (the dashed edge in Fig. 2, as previously discussed).
Precondition: $\rho_1 \neq \rho_2$. If $\rho_2 > \rho_1$, then $\rho_2 - \rho_1 \geq 4$, otherwise $J + \rho_2 - \rho_1 \geq 4$.

In general, a *multi-neighborhood* $\mathcal{N}_\cup$ is defined as the union of a collection of K individual neighborhoods, which is written as $\mathcal{N}_\cup = \bigcup_{k=1}^{K} \mathcal{N}_k$. In our case, with the three concrete neighborhoods designed for the problem, we define it as $\mathcal{N}_\cup = \text{Move} \cup \text{Swap} \cup \text{2-Opt}$.

Two desirable properties for a well-designed multi-neighborhood are the absence of null moves and that the neighborhoods are disjoint. Neighborhood disjointedness prevents the occurrence of repeated moves across different neighborhoods, which would otherwise lead to higher computation costs if the neighborhood is explored systematically, or to bias imbalance if the neighborhood is explored stochastically. To cope with such requirements, in the Swap neighborhood, we impose that the distance between two jobs is at least 2, which avoids repetitions with the Move neighborhood. Indeed, swapping jobs at adjacent positions can already be obtained with a Move operation

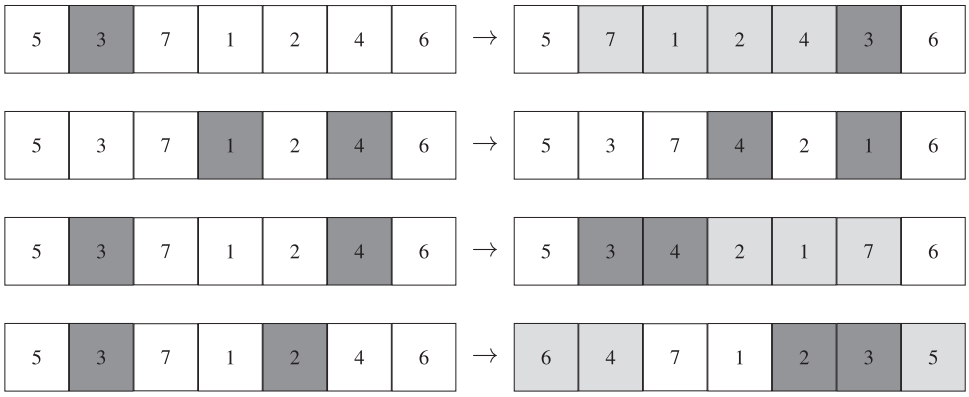


Fig. 3. Illustration of three neighborhood moves used in MNSA. From top to bottom: Move⟨3, 6⟩, Swap⟨1, 4⟩, 2-Opt⟨2, 6⟩, 2-Opt⟨5, 2⟩.

that moves a job to an adjacent position. Therefore, we prevent this operation from arising in both neighborhoods. Analogously, in the 2-Opt neighborhood, we impose that the distance between the two positions is at least 4, in order to avoid repetitions with both the Move and the Swap neighborhoods.

Figure 3 shows examples of moves and their effects. In particular, for the 2-Opt it shows both cases $\rho_2 > \rho_1$ and $\rho_1 > \rho_2$.

4.2.4. Exploration criterion

The exploration is *stochastic*. The choice of stochastic neighborhood selection is motivated by the success shown by this approach in recent studies, that has demonstrated to outperform several methods based on both systematic exploration or large neighborhoods, on competitive benchmarks such as the integrated healthcare timetabling problem from the IHTC2024 competition (Zanazzo et al., 2025), the sports timetabling problem from the ITC2021 competition (Van Bulck et al., 2024), and the capacitated examination timetabling problem from the ITC2007 competition (Van Bulck et al., 2025).

In *stochastic exploration*, every neighborhood $\mathcal{N}_k \in \mathcal{N}_\cup$ is assigned a probability $\sigma_k \in [0, 1]$ such that $\sum_{k=1}^{K} \sigma_k = 1$. In our case, the parameters are $\sigma_{\mathsf{Move}}, \sigma_{\mathsf{Swap}}, \sigma_{\mathsf{2-Opt}}$.

The move selection is performed with a *two-step biased random move selection*. First, we choose a random neighborhood, according to the probabilities σ_k, with a biased random selection. Second, a move inside the chosen neighborhood is selected stochastically, according to the uniform distribution. The two-step biased random move selection is resumed as follows:

1. Neighborhood selection: $\mathcal{N}_k \leftarrow \mathsf{RandomNH}(\mathcal{N}_\cup, \{\sigma_1 \dots \sigma_K\})$
2. Move selection: $m \leftarrow \mathsf{RandomMove}(S, \mathcal{N}_k)$,

where RandomNH indicates the neighborhood selection and RandomMove indicates the move selection inside the chosen neighborhood. At every iteration of the local search, the two-step procedure is repeated, so that all choices are independent.

Algorithm 1. Multi-Neighborhood Simulated Annealing

1: **procedure** MNSA T_0, T_f, α, ρ, σ_{Move}, σ_{Swap}, $\sigma_{2-\mathsf{Opt}}$
2: $T \leftarrow T_0$, $N_s \leftarrow \mathcal{I}/\log_\alpha(T_f/T_0)$, $N_a \leftarrow \rho N_s$
3: $S \leftarrow \mathsf{InitialSolution}(\Sigma)$, $S_{best} \leftarrow S$
4: **while** $T \geq T_f$ **do**
5: $n_s \leftarrow 0$, $n_a \leftarrow 0$
6: **while** $n_s < N_s \wedge n_a < N_a$ **do**
7: $\mathcal{N}_k \leftarrow \mathsf{RandomNH}(\mathcal{N}_\cup, \sigma_{\mathsf{Swap}}, \sigma_{2-\mathsf{Opt}})$
8: $m \leftarrow \mathsf{RandomMove}(S, \mathcal{N}_k)$
9: $\Delta F \leftarrow F(S \oplus m) - F(S)$
10: **if** $\Delta F \leq 0 \vee \mathsf{RandomReal}(0, 1) < e^{-\Delta F/T}$ **then**
11: $S \leftarrow S \oplus m$
12: $n_a \leftarrow n_a + 1$
13: **if** $F(S) < F(S_{best})$ **then**
14: $S_{best} \leftarrow S$
15: **end if**
16: **end if**
17: $n_s \leftarrow n_s + 1$
18: **end while**
19: $T \leftarrow T \cdot \alpha$
20: $N_s \leftarrow N_s + (N_s - n_s)/\log_\alpha(T_f/T)$
21: **end while**
22: **return** S_{best}
23: **end procedure**

4.2.5. Acceptance criterion

An SA procedure based on the classic version proposed by Kirkpatrick et al. (1983), with the addition of the *cut-off* mechanism (Johnson et al., 1989), is used to guide the search. It defines, particularly, the acceptance rule. Improving moves are always accepted, while worsening moves are accepted according to the Metropolis criterion, relying on a *temperature* parameter T that defines the acceptance probability as $e^{-\Delta F/T}$. ΔF is the cost difference between the evaluated and the current solution.

4.2.6. Overall algorithmic scheme

Algorithm 1 describes the whole algorithmic procedure. It relies on parameters T_0, T_f, α, ρ for the SA, and σ_{Move}, σ_{Swap}, $\sigma_{2-\mathsf{Opt}}$ for what regards the neighborhood rates. The stop criterion is defined as the total iterations $\mathcal{I}$. At line 2 with the temperature is initialized to T_0 and the values for N_s and N_a, which denote the maximum number of neighbors sampled and accepted, respectively, at each temperature level, are computed. At line 3, the initial solution is generated as a random permutation of jobs. The main loop from lines 4 to 21 is executed until the temperature reaches the final value T_f. Line 4 initializes the counters n_s and n_a of sampled and accepted neighbors in the current temperature. Lines from 5 to 14 contain the inner loop that is repeated to sample up to N_s neighbors, or up to N_a accepted moves, at every temperature level. The multi-neighborhood exploration occurs at lines 7 and 8. The acceptance criterion is applied at line 10. In order to accept worsening moves according to probability $e^{-\Delta F/T}$, we rely on a procedure $\mathsf{RandomReal}(0, 1)$ to draw a random

number between 0 and 1. Counter n_a is incremented for accepted moves (line 12), and the best-so-far is potentially updated (lines 13 and 14). The counter of the number of sampled neighbors is updated at line 17. After either N_s solutions have been sampled or N_a have been accepted, temperature T is updated according to the geometric cooling scheme (line 19), depending on parameter α. The potentially unused $N_s - n_s$ solution evaluations are redistributed equally among the remaining temperature levels (line 20). Finally, the best solution found S_{best} is returned (line 22).

4.3. Large neighborhood search

In this section, we propose a new heuristic that combines the strengths of CP with an LNS framework.

4.3.1. Solution representation

In our LNS approach, we utilize two distinct solution representations tailored to the different types of recreate operators employed:

Machine-job lists: Each machine has a list detailing the sequence of jobs assigned to it. This representation is naturally suited for heuristic recreate operators.

Job start times vector: A vector $S = (s_1, \ldots, s_J)$ stores the start time s_j for every job j. This format is used for CP-based recreate operators because it directly corresponds to the interval variables used in our CP2 formulation, simplifying integration with the CP solver.

It is important to note that these representations are interconvertible. A solution represented by the vector of start times can be converted back to the corresponding machine-job list representation by sorting jobs by their start times and then applying the forward greedy insertion procedure described in Section 4.2.1.

4.3.2. Framework

The LNS framework operates by iteratively applying multiple ruin operators ($\mathcal{R}^-$) to partially relax the current solution by removing a subset of jobs, followed by reconstruction using recreate operators ($\mathcal{R}^+$) which reinserts the removed jobs using CP. This process allows for exploring a larger solution space while potentially improving solution quality.

The pseudo-code of the proposed LNS method is presented in Algorithm 2. First, an initial solution is constructed using a greedy heuristic (Line 2). The algorithm then enters a multi-start loop (Line 3), inside which a solution S_0 is constructed using the same greedy heuristic (Line 4), and the reference solution S^* is initialized to S_0 (Line 5). The core of the algorithm operates within an inner iterative loop executed for a predefined number of iterations $Iter_{LNS}$ (Line 6). At each iteration, a `Ruin` operator is applied to S^*, creating a partial solution S_{ruin} (Line 7). Subsequently, S_{ruin} is repaired through a `Recreate` operator, yielding a new complete candidate solution $S_{recreate}$ (Line 8). An `accept` function evaluates whether the new solution $S_{recreate}$ meets certain criteria relative to the reference solution S^* (Line 9). If accepted, the reference solution S^* is updated to $S_{recreate}$ (Line 10),

Algorithm 2. LNS metaheuristic

1: **procedure** LNS $Iter_{LNS}$, $Iter_{mutistart}$, $Iter_{pert}$, p_{CP}, r_{min}, r_{max}
2: $S \leftarrow$ greedy(), $S_{best} \leftarrow S$
3: **for** $iter = 0 : Iter_{mutistart}$ **do**
4: $S_0 \leftarrow$ greedy()
5: $S^* \leftarrow S_0$
6: **for** $i = 0 : Iter_{LNS}$ **do**
7: $S_{ruin} \leftarrow \mathsf{Ruin}(S^*)$
8: $S_{recreate} \leftarrow \mathsf{Recreate}(S_{ruin})$
9: **if** $\mathsf{accept}(S_{recreate}, S^*)$ **then**
10: $S^* \leftarrow S_{recreate}$
11: **if** $F(S_{recreate}) < F(S_{best})$ **then**
12: $S_{best} \leftarrow S^*$
13: **end if**
14: **end if**
15: **if** $Iter_{pert}$ iterations without improving **then**
16: $\mathsf{Perturbate}(S^*)$
17: **end if**
18: **end for**
19: **end for**
20: **return** S_{best}
21: **end procedure**

and, possibly the overall best S_{best} is updated (Line 12). If the reference solution has not been improved for $Iter_{pert}$ consecutive iterations (Line 15), a `Perturbate` function is activated that allows the acceptance of a worsening solution as the reference solution S^* to diversify the search (Line 16). After all iterations of the outer loop are completed, the algorithm returns the overall best solution found, S_{best} (Line 20).

4.3.3. Removal and recreate operators

Our approach employs several removal heuristics to identify the subset of jobs to remove:

Time-based removal: This operator selects multiple random time frames and identifies all jobs scheduled overlap with them for removal. To implement this efficiently, the jobs on each machine are first sorted by their start time. A binary search is then used to find all jobs that overlap with a given time frame. The overall time complexity is hence $O(N\log N)$. The removal process is terminated once the target number of jobs has been removed.

Machine-based removal: In this operator, we randomly select one or several machines. For each selected machine, all jobs assigned to it are then identified and then removed with a probability λ. This operator first shuffles the order of the machines to introduce randomness. It then iterates through each machine in this random sequence and selects jobs. The removal process is also terminated once the target number of jobs has been removed. Since this process

involves a single pass over all machines and their jobs, the time complexity is $O(N)$.

Priority-based removal: We identify a set of jobs based on criteria such as longest processing time, highest number of conflicts, or random selection. Each identified job is then removed with a probability λ, until the target number of removals is reached. This operator first sorts all jobs according to a chosen priority rule. It then iterates through the sorted list to select a predefined number of jobs for removal. The overall time complexity is $O(N\log N)$, dominated by the sorting step.

In our experiments, the removal probability λ is set to a high value of 0.9. When the Ruin procedure is invoked, one of the defined removal heuristics is selected with equal probability, and the chosen operator is then executed. The number of jobs to be removed is adjusted dynamically during the search. It starts at r_{min} and increases linearly if the solution fails to improve for $Iter_{pert}$ consecutive iterations, but is capped at r_{max}.

Once a subset of jobs is removed, we use two modified versions of the CP2 model to reinsert them and potentially reschedule others:

Disjunctive insertion: In this operator, we modify the CP2 model solved during recreation. For any pair of conflicting jobs $(i, j) \in \mathcal{C}$, where *neither* i nor j was removed, we determine their relative order in the current solution before removal (e.g., i precedes j). We enforce this specific precedence (e.g., using endBeforeStart(Itv_i, Itv_j)) directly in the CP model, replacing the original disjunction (12) for that specific pair. If either job i or j in a conflicting pair $(i, j) \in \mathcal{C}$ was removed, the original disjunctive constraint (12) is used for that pair. The standard machine capacity constraint (11) and objective (10) are applied to all jobs (original non-removed and removed to be reinserted).

Machine insertion: This operator modifies the CP2 model to preserve aspects of the original machine schedules. For any pair of jobs (i, j) that were originally assigned to the same machine and were *not* removed, we enforce their original relative order (e.g., using endBeforeStart(Itv_i, Itv_j)). For conflicting pairs $(i, j) \in \mathcal{C}$ involving at least one removed job, we use the original disjunctive constraint (12), similar to the disjunctive insertion operator. The standard machine capacity constraint (11) and objective (10) are applied.

Random insertion: This operator inserts removed jobs to random slots. To implement the insertion, we first transform the solution into a permutation-based representation as described in Section 4.2. This intermediate format allows us to easily modify the solution by simply inserting a job at a random position within the permutation. The modified sequence is then decoded back into a valid schedule using the linear-time algorithm from Section 4.2

Priority-based insertion: Removed jobs are first ordered using a priority rule, such as shortest processing time, longest processing time, or the number of conflicts associated with the job. Subsequently, the jobs are inserted one by one, according to

this order, into the earliest available position that maintains schedule feasibility. The worst-case time complexity is $O(N^3)$, as the algorithm iterates through the removed jobs, scans up to $O(N)$ potential insertion spots for each, and validates every spot against $O(N)$ other jobs to ensure feasibility.

These insertion operators fall into two categories: CP-based (disjunctive insertion, machine insertion) and heuristic (random insertion, priority-based insertion). The algorithm selects a CP-based operator with probability p_{CP}, or otherwise chooses a heuristic operator (with probability $1 - p_{CP}$). We set a time limit of t_{CP} seconds for each CP-based insertion run. This is necessary because CP, despite finding good solutions quickly, requires significant time to prove their optimality. In the final setting, t_{CP} is set to three seconds. The specific operator within the selected category is then typically chosen uniformly at random.

4.3.4. Initial solution strategy

To create an initial solution for the algorithm and for each restart, we implement a greedy procedure. This function first runs a random insertion heuristic on an empty schedule to create a feasible solution. Then, after relaxing the disjunctive constraints between jobs, this solution is used as a seed to run a disjunctive insertion heuristic.

5. Experimental results

In this section, we detail the experimental results obtained to evaluate and compare the performance of the proposed methods.

We begin by comparing our CP models to the existing CP model described in Ta et al. (2024). Since the CP model in that paper was shown to outperform three of its MILP formulations, we will focus our comparison only on their CP model. Following this, we present a second experiment designed to evaluate and compare the two metaheuristics. We conclude by highlighting the key findings and insights derived from the experimental results.

All experiments were implemented in C++ and executed single-threaded on machines with Intel(R) Xeon(R) Platinum 8358 CPU @ 2.60 GHz. The CP models were solved using IBM CP Optimizer 22.1.1, with a one-hour time limit per instance.

5.1. Parameter tuning

We tuned our solvers using the IRACE package (López-Ibáñez et al., 2016), which implements the iterated race procedure, as a black-box automatic algorithm configuration tool. In order to avoid overfitting the parameters on the same instances used for validation, we perform the tuning procedure on the training dataset. This procedure follows the best practices prescribed by experts (Birattari and Kacprzyk, 2009; López-Ibáñez et al., 2016).

Table 3
MNSA tuning

Category	Parameter	Range	Value
SA	T_0	[1.0,100]	79.3
SA	T_f	[0.01,1.0]	0.036
SA	α	[0.980,0.999]	0.991
SA	ρ	[0.02, 0.30]	0.046
NH	σ_{Swap}	[0.0,0.3]	0.029
NH	$\sigma_{\mathsf{2-Opt}}$	[0.0,0.7]	0.290

Table 4
LNS tuning

Category	Parameter	Range	Value
Search control	$Iter_{LNS}$	[50000, 1000000]	727443
Search control	$Iter_{mutistart}$	[1, 10]	4
Search control	$Iter_{pert}$	[1000, 10000]	9881
Search control	p_{CP}	[0, 1]	0.0725
LNS destruction	r_{min}	[0.01, 0.2]	0.0243
LNS destruction	r_{max}	[0.2, 0.4]	0.3112

The tuning of MNSA is performed in a single stage, with the SA and the neighborhoods (NH) parameters. The budget is set to 10,000 experiments. The best resulting configurations are shown in Table 3, together with the ranges that were allowed (the ranges are fixed after preliminary testing). σ_{Move} is not tuned, as it is calculated as $1 - \sigma_{\mathsf{Swap}} - \sigma_{\mathsf{2-Opt}}$, thus reducing the size of the parameter hyperspace.

Similar to MNSA, the tuning budget for LNS was set to 10,000 experiments. The parameters tuned control various aspects of the LNS search: the main iteration counts for ILS and GRASP ($iter_{LNS}$, $iter_{GRASP}$), the probability parameter for CP insertions (*cp_prob*), and the boundaries for the destruction size (p_{min}, p_{max}). The best configuration identified by `irace` after exhausting the budget is detailed in Table 4.

5.2. Computational results for the CP models

This section presents a computational comparison between the proposed CP model (CP2) and the previous model presented by Ta et al. (2024) (CP1). Each model was executed on every instance within our dataset using a single thread, with a time limit imposed of one hour per instance. The results obtained from these experiments are detailed in Table 5. For each model, we report the best objective value found (obj), the best lower bound achieved (bound), the execution time in seconds (time), the optimality gap (gap(%)), and the termination status (O: optimality proven, F: feasible solution found, optimality not proven).

Table 5
Performance comparison of the two CP models on the benchmark instances

Instance	CP2					CP1				
#	obj	bound	time	gap(%)	status	obj	bound	time	gap(%)	status
01	899	899	1	0.0	O	899	78	3600	91.3	F
02	712	712	1	0.0	O	712	152	3600	78.7	F
03	5552	5552	7	0.0	O	5552	259	3600	95.3	F
04	1566	1566	4	0.0	O	1566	424	3600	72.9	F
05	2953	2953	36	0.0	O	2954	536	3600	81.9	F
06	7881	7881	6	0.0	O	7881	1071	3600	86.4	F
07	1671	1671	66	0.0	O	1671	110	3600	93.4	F
08	6641	6641	9	0.0	O	6641	248	3600	96.3	F
09	889	889	38	0.0	O	889	169	3600	81.0	F
10	2930	2928	3600	0.1	F	2930	361	3600	87.7	F
11	3153	2517	3600	20.2	F	3340	248	3600	92.6	F
12	1775	1707	3600	3.8	F	1780	380	3600	78.7	F
13	34183	34183	573	0.0	O	34183	1444	3600	95.8	F
14	1170	1138	3600	2.7	F	1174	150	3600	87.2	F
15	3083	2382	3600	22.7	F	3203	216	3600	93.3	F
16	1329	1329	1945	0.0	O	1329	118	3600	91.1	F
17	3694	3241	3600	12.3	F	3712	260	3600	93.0	F
18	11431	6950	3600	39.2	F	11562	1671	3600	85.5	F
19	767	375	3600	51.1	F	759	50	3600	93.4	F
20	1186	1186	32	0.0	O	1186	120	3600	89.9	F
21	5034	2868	3600	43.0	F	5257	212	3600	96.0	F
22	3113	1079	3600	65.3	F	3082	315	3600	89.8	F
23	4906	4584	3600	6.6	F	5111	286	3600	94.4	F
24	6765	3555	3600	47.5	F	6872	560	3600	91.9	F
25	10548	6197	3600	41.2	F	10737	1934	3600	82.0	F
26	2427	765	3600	68.5	F	2455	226	3600	90.8	F
27	4466	1241	3600	72.2	F	4452	150	3600	96.6	F
28	12337	11374	3600	7.8	F	12319	705	3600	94.3	F
29	27407	13135	3600	52.1	F	27313	1060	3600	96.1	F
30	937	251	3600	73.2	F	939	72	3600	92.3	F
Avg.	5713.5	4391.6	2250.6	21.0	O: 40%	5748.7	452.8	3600	89.6	O: 0%

Notably, our model successfully proves optimality for 12 of the 30 instances, often accomplishing this significantly faster than the 3600-second time limit. In contrast, the reference model fails to confirm optimality for any instance within the time frame. For instances where the time limit was reached, the proposed model consistently generates substantially tighter lower bounds, as indicated by the bold values, suggesting a more effective pruning process or a stronger relaxation. While the reference model obtains slightly better feasible objective values in a few cases (19, 22, 27–29), our model generally finds solutions of equal or superior quality, achieving distinctly better objective values on several instances. These results demonstrate advantages for our formulation regarding computational speed, success in proving optimality, and the quality of the resulting lower bounds.

Table 6
Comparison of metaheuristic methods and CP on benchmark instances

Instance	MNSA					LNS					CP2		
#	max	min	avg	gap(%)	std	max	min	avg	gap(%)	std	obj	gap(%)	time
01	899	899	899.0	0.000	0.0	899	899	899.0	0.000	0.0	899	0.000	1
02	712	712	712.0	0.000	0.0	712	712	712.0	0.000	0.0	712	0.000	2
03	5552	5552	5552.0	0.000	0.0	5552	5552	5552.0	0.000	0.0	5552	0.000	6
04	1566	1566	1566.0	0.000	0.0	1566	1566	1566.0	0.000	0.0	1566	0.000	3
05	2954	2953	2953.1	0.003	0.3	2957	2953	2953.9	0.030	1.4	**2953**	0.000	33
06	7881	7881	**7881.0**	0.000	0.0	7882	7881	7881.1	0.001	0.3	**7881**	0.000	5
07	1672	1671	1671.7	0.042	0.5	1676	1673	1674.6	0.215	0.8	**1671**	0.000	74
08	6641	6641	6641.0	0.000	0.0	6641	6641	6641.0	0.000	0.0	6641	0.000	6
09	889	889	889.0	0.000	0.0	889	889	889.0	0.000	0.0	889	0.000	53
10	2930	2929	**2929.1**	0.000	0.3	2953	2931	2940.8	0.399	6.8	2930	0.031	900
11	3080	2998	**3032.7**	0.000	28.4	3487	3315	3408.0	12.375	45.3	3153	3.967	900
12	1777	1737	**1750.3**	0.000	13.3	1826	1753	1780.0	1.697	23.2	1775	1.411	900
13	34183	34183	**34183.0**	0.000	0.0	34186	34183	34183.7	0.002	0.9	**34183**	0.000	537
14	1168	1161	**1164.6**	0.000	2.1	1235	1178	1206.4	3.589	21.8	1170	0.464	900
15	2930	2839	**2872.9**	0.000	29.1	3329	3173	3255.5	13.318	48.4	3110	8.253	900
16	1330	1329	**1329.3**	0.000	0.5	1339	1329	1332.5	0.241	3.5	1330	0.053	900
17	3652	3556	**3582.4**	0.000	30.1	3757	3637	3704.3	3.403	38.7	3718	3.785	900
18	11008	10659	**10755.2**	0.000	107.7	11885	11203	11545.5	7.348	196.1	11533	7.232	900
19	713	683	**697.0**	0.000	9.7	772	739	759.6	8.981	9.1	770	10.473	900
20	1186	1186	1186.0	0.000	0.0	1186	1186	1186.0	0.000	0.0	1186	0.000	42
21	4655	4450	**4538.4**	0.000	61.0	5314	5120	5195.2	14.472	59.5	5103	12.441	900
22	2843	2788	**2808.4**	0.000	18.0	3213	2954	3091.9	10.095	81.7	3118	11.024	900
23	4814	4747	**4778.0**	0.000	22.5	5024	4861	4935.3	3.292	55.6	4959	3.788	900
24	6319	6076	**6195.1**	0.000	75.4	6815	6504	6682.1	7.861	106.1	6765	9.199	900
25	10196	10001	**10065.4**	0.000	68.0	10826	10196	10556.2	4.876	175.0	10652	5.828	900
26	2316	2296	**2304.4**	0.000	6.4	2499	2373	2448.9	6.271	40.3	2438	5.798	900
27	4131	4027	**4078.9**	0.000	31.5	4810	4546	4669.9	14.489	74.0	4530	11.059	900
28	11871	11725	**11798.0**	0.000	44.4	12638	12288	12440.3	5.444	113.2	12412	5.204	900
29	25686	24992	**25405.8**	0.000	208.8	28199	27395	27843.9	9.597	250.6	27975	10.113	900
30	901	896	**898.4**	0.000	1.4	941	913	929.4	3.451	10.7	937	4.297	900
Avg.	5548.5	5467.4	**5503.9**	0.002	25.3	5833.6	5684.8	5762.1	4.382	45.4	5750.4	3.814	595.5

5.3. Computational results for the metaheuristics

Table 6 presents the results for the two metaheuristics, and their comparison with the CP model's results from Table 5 on the validation dataset. For MNSA and LNS, which are stochastic, the maximum, minimum, and average costs and average running time, out of 10 independent runs per instance, are reported. For all algorithms, the execution time cutoff has been set to 900s. As the CP is deterministic, only one value per instance is reported (column "obj"). Values in boldface indicate the best results obtained. If, for a given instance, all solvers consistently found the same cost, no result is highlighted.

Absolute values of the makespan vary across different orders of magnitude in different instances. To ease comparability of methods, three columns "gap(%)" indicate, for the average results of each

method, the gap from the best performing method. To track also tiny differences, we express the gap with three decimal digits. The minimum gap recording is obviously 0.000%, indicating that the method is the best performing on the given instance, while the maximum is 14.489%, recorded by the LNS on instance 27. The maximum gap of MNS is only 0.042%, with an average gap from the best method of 0.002%. The gaps are more consistent for LNS and CP2, which record, respectively, average values of 4.382% and 3.814%.

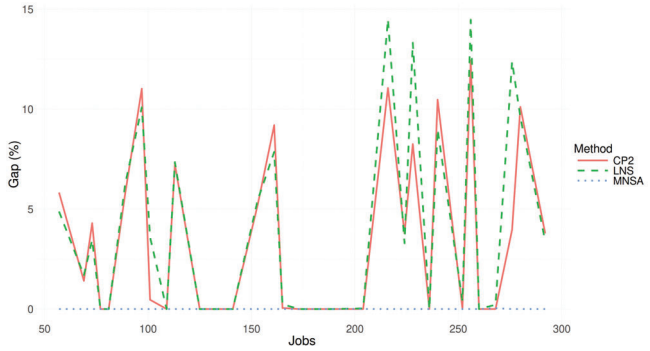
We observe that in many small or low-conflict instances, all methods find consistently the optimal solution, while the challenging instances are those with higher conflict rates, where the exact method struggles. Indeed, the presence of a high number of conflicts makes finding good schedules much harder. Therefore, when the conflict rate is low, choosing the CP solver would lead to the optimal solution quickly, even on large instances. We point out that also the metaheuristics in most of the instances with low conflict rates find the optimal solution consistently.

As the conflict rate increases, the CP model does not provide good solutions, regardless of the number of jobs in the instance. In these cases, the metaheuristics represent a most suitable choice, with the MNSA offering a clear domination over the LNS, with the exception of instance 20 (in which, nevertheless, also CP finds the optimal solution).

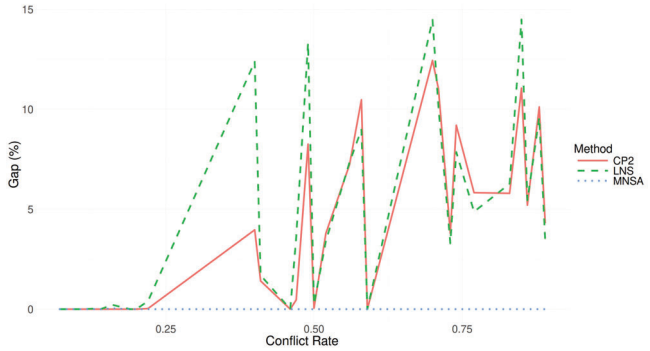
A Wilcoxon paired statistical test confirms that the MNSA is superior to both LNS (with p-value 2.889×10^{-5}) and to CP2 (with p-value 1.063×10^{-4}). Conversely, no statistical significance emerges when LNS and CP2 are compared (p-value is 0.4029).

Figure 4 shows the results of the three methods according to two features of the instances: number of jobs J (Fig. 4a) and conflict rate c (Fig. 4b). In both figures, the x-axis indicates the given instance feature (J or c) and the y-axis shows the gap in percentage from the best method, as reported in Table 6. From the first plot, we observe that gaps between methods are found both for small and large instances; however, gaps above 10% are found almost exclusively in instances with more than 200 jobs. Conversely, no clear pattern emerges, as cases where all three methods get to the same result occur regardless of the number of jobs. From the second plot, we observe a clearer effect of the conflict rate: the three methods behave rather similarly when conflict rates are low, while the advantage of MNSA is more evident in instances with higher conflict rates. We also observe that LNS and CP2 show a rather similar behavior. This is no surprise, considering that LNS uses CP-based repair operators. Instances with low-to-moderate conflict rates, however, seem to make an exception, as CP2 performs better than LNS on them.

The performances of the algorithms depending on combinations of instance features are shown in Fig. 5. For each pair of features, the best-performing algorithm on a given instance is represented (cross: MNSA, plus: LNS, triangle: CP, as shown in the legend). In case of a tie, more symbols are superimposed. Two observations that emerge clearly are the superiority of MNSA and the relationship of the performance with the conflict rate. When the conflict rate (c) is small, no clear winner emerges, while when the conflict rate is high, MNSA is the best-performing method. This relation is evident when the conflict rate c is put in a relationship with any other feature. Conversely, the other features do not seem to influence the algorithm performances as strongly as the conflict rate. The graphical analysis of the results suggests that the gap between the MNSA and the other methods is more evident when the conflict graph is denser. Finally, the plots suggest that the size of the instance (in terms of number of jobs J) is not an issue for CP2, which is the sole best performer on two instances with a high number of jobs, as long as they have a low conflict rate: they are instances 5 and 7, with 204 and 268 jobs, respectively, which are easily solved to the optimum in a



(a) Instances sorted by number of jobs J.



(b) Instances sorted by conflict rate c.

Fig. 4. Percentage gaps of the compared methods on the test instances.

few seconds. MNSA also achieves a good performance on these instances but does not consistently find the optimum within the time limit.

The varying performances of the algorithms on instances having different features tell us that the problem of choosing the right algorithm for the problem is still open and deserves further investigation.

6. Conclusions and future work

In this work, we addressed the scheduling conflicting jobs on parallel identical machines problem with makespan minimization (MM-SCJPIM), a computationally challenging extension of classical parallel machine scheduling. We proposed and evaluated three independent solution methods: a CP formulation serving as an exact approach and two metaheuristics: an MNSA and a CP-based LNS.

To support reproducibility and benchmarking, we also introduced a novel dataset comprising challenging and diverse instances with up to 300 jobs and varying conflict structures. Our experiments demonstrate that the proposed CP model significantly outperforms a recent formulation from the literature (Ta et al., 2024), achieving optimal solutions within seconds on low-conflict instances. Among the two metaheuristics, MNSA consistently outperforms LNS in both solution

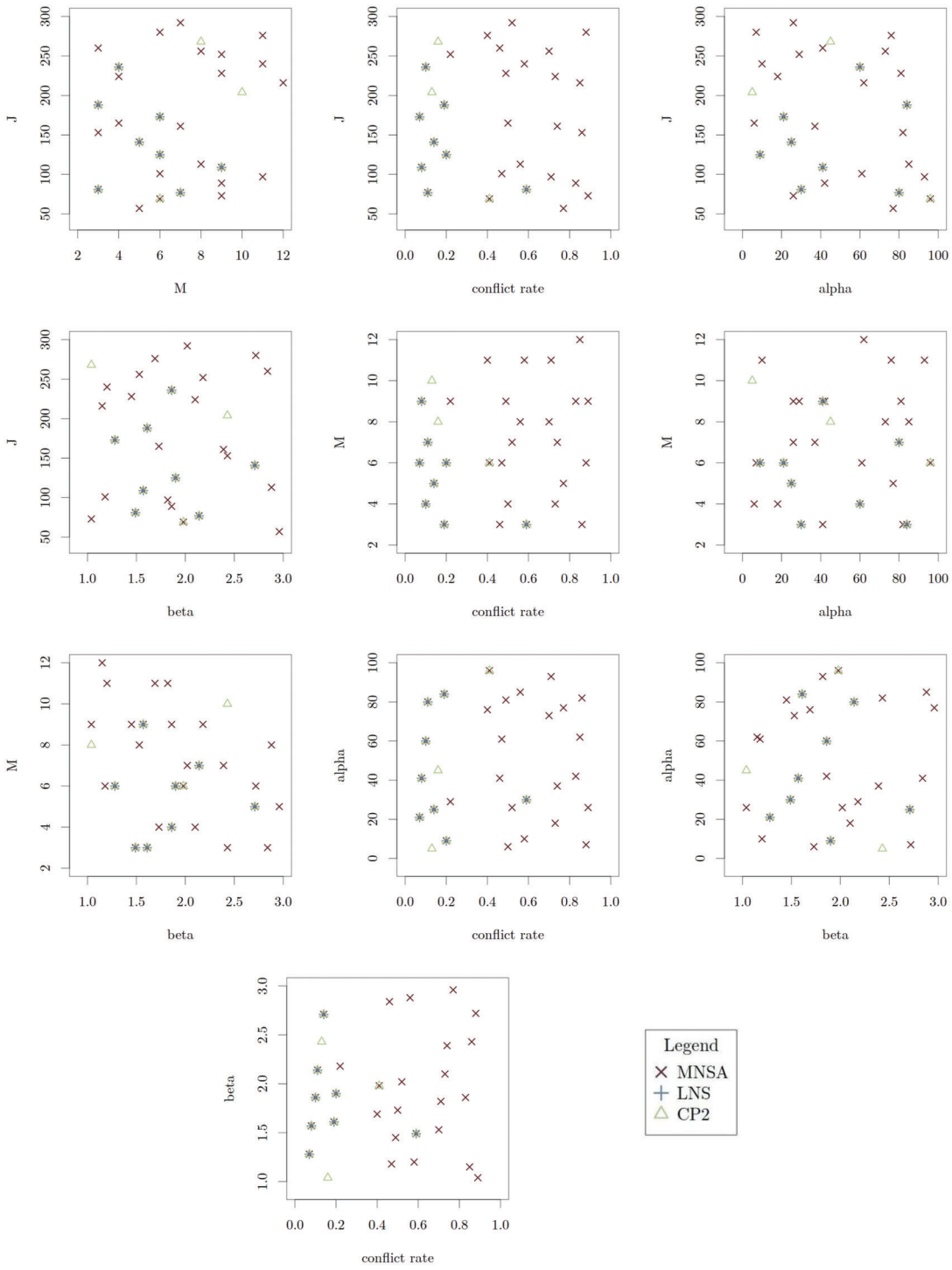


Fig. 5. Best results obtained by the method on the validation instance, plotted by instance features

quality and computational time. Notably, LNS shows promising behavior on highly conflicting instances, where the reduced search space enables faster local optimization.

Looking ahead, we intend to adapt our solution methods to similar scheduling problems involving parallel identical machines, with different notions of conflicts. For instance, we plan to work on the problem proposed by Bianchessi and Tresoldi (2021), who treat the conflicts in terms of the impossibility of executing two jobs on the same machine. Another problem of interest is the automated guided vehicles (AGV) scheduling problem with battery constraints proposed by Boccia et al. (2023), in which AGVs can be treated as parallel identical machines. Moreover, we aim to enhance our metaheuristics by incorporating advanced neighborhoods from vehicle routing literature, such as 3-Opt and Multi-Opt operators, to further improve solution quality and flexibility. Finally, other promising directions are algorithm selection, following approaches such as those proposed by Van Bulck et al. (2024), and component-wise analysis, taking inspiration from Franzin and Stützle (2019).

Acknowledgments

Open Access funding provided by Wirtschaftsuniversitat Wien/KEMÖ.

References

Alon, N., 1983. A note on the decomposition of graphs into isomorphic matchings. *Acta Mathematica Hungarica* 42, 221–223.

Baker, B.S., Coffman, E.G., Jr., 1996. Mutual exclusion scheduling. *Theoretical Computer Science* 162, 225–243.

Bendraouche, M., Boudhar, M., 2012. Scheduling jobs on identical machines with agreement graph. *Computers & Operations Research* 39, 382–390.

Bianchessi, N., Tresoldi, E., 2021. A stand-alone branch-and-price algorithm for identical parallel machine scheduling with conflicts. *Computers & Operations Research* 136, 105464.

Birattari, M., Kacprzyk, J., 2009. *Tuning Metaheuristics: A Machine Learning Perspective*. Studies in Computational Intelligence, volume 197. Springer.

Blazewicz, J., Lenstra, J.K., Kan, A.R., 1983. Scheduling subject to resource constraints: classification and complexity. *Discrete Applied Mathematics* 5, 11–24.

Boccia, M., Masone, A., Sterle, C., Murino, T., 2023. The parallel AGV scheduling problem with battery constraints: a new formulation and a matheuristic approach. *European Journal of Operational Research* 307, 590–603.

Bodlaender, H.L., Jansen, K., 1993. On the complexity of scheduling incompatible jobs with unit-times. In *Proceedings of the 18th International Symposium on Mathematical Foundations of Computer Science 1993*, (MFCS'93), Gdansk, Poland, August 30 - September 3, 1993. Springer, Berlin, pp. 291–300.

Buchem, M., Kleist, L., Schmidt genannt Waldschmidt, D., 2022. Scheduling with machine conflicts. In the *International Workshop on Approximation and Online Algorithms*. Springer, Berlin, pp. 36–60.

Caimi, G., Chudak, F., Fuchsberger, M., Laumanns, M., Zenklusen, R., 2011. A new resource-constrained multicommodity flow model for conflict-free train routing and scheduling. *Transportation Science* 45, 212–227.

Ceschia, S., Di Gaspero, L., Rosati, R.M., Schaerf, A., 2026. Multi-neighborhood simulated annealing for the home healthcare routing and scheduling problem. *International Transactions in Operational Research* 33, 38–67.

Da Ros, F., Di Gaspero, L., 2024. An empirical analysis of tabu lists. In Sevaux, M., Olteanu, A.L., Pardo, E.G., Sifaleras, A., Makboul, S. (eds) *Metaheuristics*. Springer Nature Switzerland, Cham. pp. 50–64.

Duh, R., Fürer, M., 1997. Approximation of *k*-set cover by semi-local optimization. In Leighton, F.T., Shor, P.W. (eds) *Proceedings of the Twenty-Ninth Annual ACM Symposium on the Theory of Computing*, El Paso, TX, May 4-6, 1997. ACM, New York, pp. 256–264.

Even, G., Halldórsson, M.M., Kaplan, L., Ron, D., 2009. Scheduling with conflicts: online and offline algorithms. *Journal of Scheduling* 12, 199–224.

Franzin, A., Stützle, T., 2019. Revisiting simulated annealing: a component-based analysis. *Computers & Operations Research* 104, 191–206.

Graham, R.L., Lawler, E.L., Lenstra, J.K., Kan, A.R., 1979. Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of Discrete Mathematics* 5, 287–326.

Grimes, J.E., 1970. Scheduling to reduce conflict in meetings. *Communications of the ACM* 13, 351–352.

Hammersley, J.M., Handscomb, D.C., 1964. *Monte Carlo Methods*. Chapman and Hall, London.

Hà, M.H., Vu, D.M., Zinder, Y., Nguyen, T.T., 2019. On the capacitated scheduling problem with conflict jobs. In *2019 11th International Conference on Knowledge and Systems Engineering* (KSE). IEEE, Piscataway, NJ, pp. 1–5.

Irani, S., Leung, V., 1996. ., In *Proceedings of the Seventh Annual ACM-SIAM Symposium on Discrete Algorithms*. ACM Press, New York, pp. 85–94.

Jansen, K., 2003. The mutual exclusion scheduling problem for permutation and comparability graphs. *Information and Computers* 180, 71–81.

Johnson, D.S., Aragon, C.R., McGeoch, L.A., Schevon, C., 1989. Optimization by simulated annealing: an experimental evaluation; part I, graph partitioning. *Operations Research* 37, 865–892.

Kaller, D., Gupta, A., Shermer, T., 1995. The χ_t coloring problem. In *Symposium on Theoretical Aspects of Computer Science* (STACS 95). Springer, Berlin, pp. 409–420.

Kirkpatrick, S., Gelatt, D., Vecchi, M., 1983. Optimization by simulated annealing. *Science* 220, 671–680.

Kowalczyk, D., Leus, R., 2017. An exact algorithm for parallel machine scheduling with conflicts. *Journal of Scheduling* 20, 355–372.

Lamorgese, L., Mannino, C., Pacciarelli, D., Krasemann, J.T., 2018. Train Dispatching. In *Handbook of Optimization in the Railway Industry*. Springer International Publishing, Cham, pp. 265–283.

Li, Y., Hao, J.K., 2022. Multi-neighborhood simulated annealing for personalized user project planning. *Applied Soft Computing* 119, 108566.

Lonc, Z., 1992. On complexity of some chain and antichain partition problems. In *Proceedings of the 17th International Workshop*. Springer-Verlag, London, pp. 97–104.

López-Ibáñez, M., Dubois-Lacoste, J., Cáceres, L.P., Birattari, M., Stützle, T., 2016. The Irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives* 3, 43–58.

Lü, Z., Fang, Y., Su, Z., Wang, Y., Wu, X., Glover, F., 2023. Dual-neighborhood iterated local search for routing and wavelength assignment. *Computers & Operations Research* 160, 106396.

Mallek, A., Bendraouche, M., Boudhar, M., 2019. Scheduling identical jobs on uniform machines with a conflict graph. *Computers & Operations Research* 111, 357–366.

Mallek, A., Boudhar, M., 2024. Scheduling on uniform machines with a conflict graph: complexity and resolution. *International Transactions in Operational Research* 31, 863–888.

Picard, J.C., Queyranne, M., 1978. The time-dependent traveling salesman problem and its application to the tardiness problem in one-machine scheduling. *Operations Research* 26, 86–110.

Rosati, R.M., Schaerf, A., 2024. Multi-neighborhood simulated annealing for the capacitated dispersion problem. *Expert Systems with Applications* 255, 124484.

Rosati, R.M., Ta, D.Q., Hà, M.H., Schaerf, A., 2024. Multi-neighborhood search for the makespan minimization problem on parallel identical machines with conflicting jobs. In the *Metaheuristics International Conference*. Springer, Cham, pp. 373–379.

Ta, D.Q., Vu, D.M., Nguyen, T.T., Le, A.D., Hà, M.H., 2024. Exact approaches for scheduling problems on parallel identical machines with conflict jobs. In *Handbook of Combinatorial Optimization*. Springer, Cham, pp. 1–26.

Tellache, N.E.H., Boudhar, M., 2017. Open shop scheduling problems with conflict graphs. *Discrete Applied Mathematics* 227, 103–120.

Tellache, N.E.H., Boudhar, M., 2018. Flow shop scheduling problem with conflict graphs. *Annals of Operations Research* 261, 339–363.

Tresoldi, E., 2021. Solution approaches for the capacitated scheduling problem with conflict jobs. In *Optimization and Decision Science: ODS, Virtual Conference, November 19, 2020*. Springer, Cham, pp. 129–140.

Turkeš, R., Sörensen, K., Hvattum, L.M., 2021. Meta-analysis of metaheuristics: quantifying the effect of adaptiveness in adaptive large neighborhood search. *European Journal of Operational Research* 292, 423–442.

Van Bulck, D., Goossens, D., Clarner, J.P., Dimitsas, A., Fonseca, G.H., Lamas-Fernandez, C., Lester, M.M., Pedersen, J., Phillips, A.E., Rosati, R.M., 2024. Which algorithm to select in sports timetabling? *European Journal of Operational Research* 318, 2, 575–591.

Van Bulck, D., Goossens, D., Schaerf, A., 2025. Multi-neighbourhood simulated annealing for the ITC-2007 capacitated examination timetabling problem. *Journal of Scheduling* 28, 217–232.

Viana, F.A., 2016. A tutorial on Latin hypercube design of experiments. *Quality and Reliability Engineering International* 32, 1975–1985.

Zanazzo, E., Smet, P., Ceschia, S., Rosati, R.M., Schaerf, A., Vanden Berghe, G., 2025. A multi-neighborhood simulated annealing approach to the integrated healthcare timetabling problem. SSRN 5376065. http://doi.org/10.2139/ssrn.5376065.

Zinder, Y., Berlińska, J., Peter, C., 2021. Maximising the total weight of on-time jobs on parallel machines subject to a conflict graph. In *Proceedings of the 20th International Conference on the Mathematical Optimization Theory and Operations Research* (MOTOR 2021), Irkutsk, Russia, July 5–10, 2021. Springer, Cham, pp. 280–295.

Zuckerman, D., 2006. Linear degree extractors and the inapproximability of max clique and chromatic number. In Kleinberg, J.M. (ed.), *Proceedings of the 38th Annual ACM Symposium on Theory of Computing*, Seattle, WA, May 21–23, 2006. ACM, New York, pp. 681–690.