Constraint programming for unrelated parallel machine scheduling with unrelated servers

Nikolaos Liouliakis[1][0009−0009−4234−1958], Angelo Sifaleras[1][0000−0002−5696−7021], and Rachid Benmansour[2][0000−0003−2553−4116]

[1] Department of Applied Informatics, University of Macedonia, School of Information Sciences, 156 Egnatias Str., 54636, Thessaloniki, Greece
nliouliak@uom.edu.gr, sifalera@uom.gr

[2] Research Laboratory in Information Systems, Intelligent Systems and Mathematical Modeling, National Institute of Statistics and Applied Economics, Rabat, Morocco
r.benmansour@insea.ac.ma

Abstract. Scheduling problems constitute an important topic in operations research due to their direct impact on productivity, resource utilization, and service quality in manufacturing and service systems. In this paper, we introduce a novel scheduling problem involving unrelated parallel machines and unrelated servers required for job setup, a setting motivated by practical production environments where setup operations require shared auxiliary resources. To support reproducibility and support future research, we also propose a new benchmark set spanning a broad range of problem sizes. To solve this problem, we developed an exact constraint programming formulation in MiniZinc and solved it using OR-Tools CP-SAT. The computational results demonstrate that the proposed model can prove optimality for almost all 280 benchmark instances, including all instances with up to 250 jobs in the scaling dataset, within a time limit of three hours. Secondly, we observed substantial runtime variability between instances, suggesting that structural characteristics play a significant role in determining computational difficulty. Overall, this work establishes a first exact optimization baseline for the proposed problem, together with a publicly available benchmark and a validated CP model.

Keywords: Constraint Programming · Heuristics · Optimization · Scheduling · Parallel Machines.

1 Introduction

Scheduling problems constitute an important topic in operations research due to their direct impact on productivity, resource utilization, and service quality in manufacturing and service systems [18]. In many practical environments, jobs must be assigned to several machines operating in parallel while respecting limited shared resources and technological or legal constraints. The complexity of these scheduling problems is further amplified when machines are not identical,

as processing times then depend on the specific job-machine pairing. Such environments, commonly referred to as unrelated parallel machine environments, arise naturally in modern production systems, flexible manufacturing, and logistics platforms, where equipment capabilities and performance levels differ significantly.

Furthermore, many real-world scheduling problems involve auxiliary (or external) resources, such as operators, robots, or handling devices, that are required to prepare jobs before processing. These resources are often modeled as servers responsible for setup, loading, unloading, or preparation operations. The presence of servers introduces additional synchronization constraints. A distinct variant is the problem studied in [15] where the setup is performed by a server, but the processing operation requires the simultaneous presence of the server and the machine (e.g., an anesthetist and a surgeon in operating rooms). This differs fundamentally from the problem addressed in this work (see Section 2), where the loading phase requires both a server and a machine simultaneously, but the subsequent execution phase requires only the machine, thus freeing the server for other tasks immediately after loading. When machines are assumed to be identical, this class of problems already exhibits high computational complexity. For example, the specific problem $P2, S1 \mid s_j = 1 \mid C_{\max}$, which involves two identical parallel machines and a single server with equal setup times (of one unit), is NP-hard for the makespan minimization objective, as proven by [6].

The challenge becomes even greater when the servers and the machines are heterogeneous, with setup times depending on the specific job, server, and machine combination. Such environments motivate the study of Unrelated Parallel Machine Scheduling (UPMS) with unrelated servers, a problem setting that captures a high level of realism, but remains largely unexplored in the literature.

A substantial amount of research has been devoted to parallel machine scheduling with shared servers under simplified assumptions. Classical studies on the problem of parallel machine scheduling with a single server establish its NP-hardness and propose exact and heuristic solution approaches [6,14]. Subsequent work develops mixed-integer linear programming formulations, branch-and-bound or branch-and-price algorithms, and a wide variety of metaheuristics to cope with problem complexity [7,13,20]. These approaches demonstrate that, even when machines and servers are identical, exact methods are generally limited to small instances, while heuristics are required to obtain high-quality solutions for larger problem sizes.

Several extensions have considered richer server-related structures, including multiple servers or the distinction between loading and unloading operations. Studies addressing problems with dedicated loading and unloading servers confirm the strong NP-hardness of these variants and highlight the scalability limits of MILP formulations [2,9,12]. To overcome these limitations, researchers have proposed reduction techniques and advanced metaheuristics such as general variable neighborhood search [1,3] or hybrid greedy algorithms [8,9]. Although these contributions significantly advance the state of the art, they are almost exclu-

sively restricted to identical parallel machines and do not account for machine-dependent processing times.

Another research direction focuses on integrating additional constraints into parallel machine scheduling. For example, the study presented in [4] in which the authors address a problem that involves identical parallel machines, a single server, and two distinctive features: (1) conflict constraints that prevent certain pairs of jobs from being processed concurrently, and (2) a mandatory fixed-duration maintenance period for the server that must be scheduled before a given deadline. The authors propose both a constraint programming model and a simulated annealing heuristic. Although this work incorporates complex constraints like conflicts and maintenance, it remains within the classical setting of identical machines and a single identical server, and it does not address the heterogeneity of resources central to the problem studied here.

Despite these advances, the existing literature reveals a clear research gap. To our knowledge, no previous study addresses the scheduling problem that simultaneously considers unrelated parallel machines and unrelated servers, where both processing times p_{ik} and setup times s_{irk} are fully heterogeneous. Moreover, no constraint programming model has been proposed for this fully heterogeneous setting. Existing approaches either rely on MILP formulations that face severe scalability issues or on heuristic and metaheuristic methods tailored to special cases with identical resources, thereby limiting their generality and optimality guarantees. This paper aims to fill this gap by proposing a novel constraint programming approach for the scheduling problem indicated by $P_m, S_r \mid s_{irk}, p_{ik} \mid C_{\max}$. The proposed CP model is specifically designed to capture complex interactions between unrelated machines and unrelated servers while maintaining modeling flexibility and computational efficiency. The effectiveness of the approach is evaluated through computational experiments.

The remainder of this paper is organized as follows. Section 2 describes the UPMS problem with unrelated servers, while Sections 3 and 4 present the constraint programming formulation and constructive heuristics for the solution of the proposed problem. The experimental results are shown in Section 5 and current limitations and future work are discussed in Section 6. Finally, conclusions are drawn in Section 7.

2 Problem description

We consider a scheduling problem denoted by

$$P_m, S_r \mid s_{irk}, p_{ik} \mid C_{\max},$$

where jobs must be loaded by servers and executed on unrelated parallel machines. Let $J = \{1, \ldots, n\}$ be a set of n non-preemptive jobs, all available at time zero. The system consists of:

- a set $M = \{1, \ldots, k\}$ of k unrelated parallel machines,
- a set $S = \{1, \ldots, r\}$ of r unrelated servers.

Each job $i \in J$ is made up of two successive phases:

1. Loading (setup) phase: Job i is loaded by exactly one server $r \in S$ onto a machine $k \in M$. This phase requires the simultaneous availability of the selected server and machine. The loading time depends on the server and machine used and is denoted by s_{irk}.
2. Execution (processing) phase: Immediately after completion of the loading phase, job i is executed on the same machine k. This phase requires only the availability of the machine and lasts p_{ik} time units.

Once the loading phase of a job is completed, the server becomes immediately available and may load another job, while the machine remains occupied until the execution of the job is finished. At any time, each server can load at most one job, and each machine can process at most one job. There are no precedence constraints among jobs. The objective is to determine an assignment of jobs to machines and servers, together with a feasible schedule of loading and execution operations, such that the makespan $C_{\max}$, defined as the completion time of the last job, is minimized.

2.1 Illustrative Example

Consider an instance with 10 jobs, two servers, and two machines with processing and setup times as shown in Table 1. An optimal schedule is shown in Fig. 1 and has a makespan of $C_{\max} = 147$.

Table 1. Processing and setup times for a 10-job instance.

Job i	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Processing times										
p_{i1}	29	40	15	27	30	28	39	32	31	21
p_{i2}	31	25	26	31	26	13	20	22	27	36
Setup times										
s_{i11}	10	7	6	7	9	3	3	5	8	7
s_{i12}	6	4	4	8	2	2	3	6	3	10
s_{i21}	10	9	9	9	8	8	3	3	6	5
s_{i22}	6	7	4	9	4	4	9	2	7	6

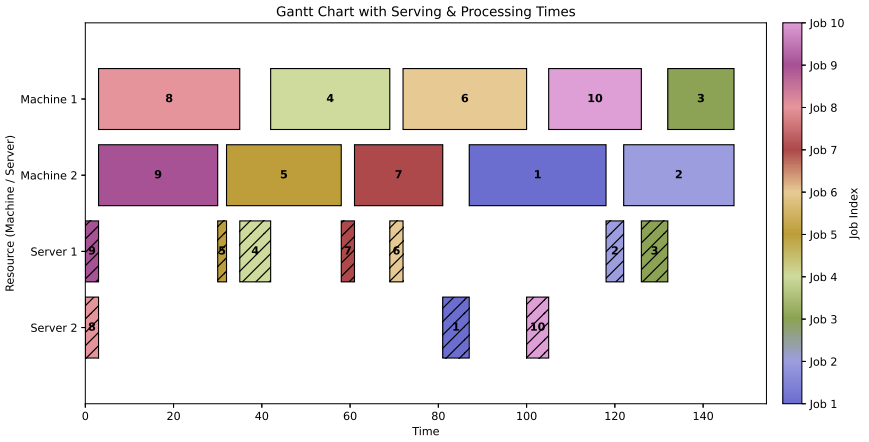

Fig. 1. Gantt chart of an optimal solution for the illustrative problem.

3 Constraint Programming Model

To define the search space, we compute an upper bound T for the makespan. This bound assumes a worst-case scenario:

$$T = \left\lceil \frac{|J|}{\min(|S|, |M|)} \right\rceil \cdot \left(\max_{i \in J, r \in S, k \in M} (s_{irk}) + \max_{i \in J, k \in M} (p_{ik}) \right) \tag{1}$$

For each job $i \in J$, we define the following decision variables:

- $st_i \in \{0, \ldots, T\}$: start time of the loading (setup) phase.
- $c_i \in \{0, \ldots, T\}$: completion time of the execution (processing) phase.
- $m_i \in M$: index of the machine assigned to job i.
- $r_i \in S$: index of the server assigned to job i.
- sd_i, pd_i: setup and processing durations, respectively, determined by the assignments (m_i, r_i).

The objective is to minimize the makespan: $\min C_{\max} \in \{0, \ldots, T\}$.

$$C_{\max} \geq c_i \qquad \forall i \in J \tag{2}$$
$$c_i = st_i + sd_i + pd_i \qquad \forall i \in J \tag{3}$$
$$sd_i = s_{i,r_i,m_i} \qquad \forall i \in J \tag{4}$$
$$pd_i = p_{i,m_i} \qquad \forall i \in J \tag{5}$$
$$\text{disjunctive}\left(\{[st_i, st_i + sd_i] \mid i \in J, r_i = r\}\right) \qquad \forall r \in S \tag{6}$$
$$\text{disjunctive}\left(\{[st_i, c_i] \mid i \in J, m_i = m\}\right) \qquad \forall m \in M \tag{7}$$

Constraint (2) defines $C_{\max}$ as the lower bound of the makespan, restricted by the completion time of every job. Constraint (3) enforces the setup to immediately

precede processing without interruption. The durations sd_i and pd_i are linked to the resource assignment variables r_i and m_i via element constraints in (4) and (5). Finally, resource capacity is managed by two sets of disjunctive constraints. Constraint (6) ensures that each server r handles at most one setup interval at a time. Crucially, the server is only occupied during the setup phase sd_i. Conversely, constraint (7) ensures that each machine m is occupied for the entire duration $[st_i, c_i]$, as the machine is reserved the moment setup begins until the job is completed.

4 Constructive heuristics for parallel machine scheduling

To assess the performance of the CP model, it is essential to employ several constructive heuristics (rule-based) for comparison. We adapted both deterministic and randomized heuristics [10,11,21] to the proposed problem. Therefore, the former only needs one pass per instance, whereas the latter could benefit from multiple passes. The same solution representation is used for all following heuristics which consists of the machine on which each job is to be processed, the server that performs the setup, and the order in which the setup of jobs starts. Note that this representation does not uniquely map the space of possible solutions; multiple representations of the same solution can exist. Given the solution representation, an optimal schedule can be determined with a greedy approach.

4.1 Random

The random approach generates a solution by selecting a random permutation of jobs for the order. For each job, the machine and server that will execute it are randomly selected from the set of machines M and servers S, respectively. Due to its purely stochastic nature, it varies greatly across runs and generally does not provide high quality solutions.

4.2 Multi-Start random 100 & 1000

The multi-start random is used to reduce the impact of chance in the previous heuristic, it generates multiple solutions using the Random heuristic, and reports the best which minimizes the makespan. For our experiments, we selected 100 and 1000 restarts and denoted those heuristics as MR (100) and MR (1000), respectively.

4.3 Shortest Processing Time (SPT)

The classical SPT sequences jobs in ascending order based on their processing times. In our problem, each job needs a machine-server pair, which affects the total time required. Therefore, we adapt SPT by defining the effective processing time of the job i as the minimum sum of the setup and processing times:

$$T_i^{min} = \min_{r,k} \{p_{ik} + s_{irk}\} \tag{8}$$

Then the jobs are ordered in ascending order of T_i^{min}, and each job is assigned to the machine and server that attains the minimum effective processing time.

4.4 Minimum Completion Time (MCT)

The classical MCT heuristic aims to balance the workload by sequentially assigning each job to the resource that produces the earliest completion time, given the current availability of resources. We extended it to handle both machine and server resources. First, the jobs are ordered exactly as in SPT. Then, in that order, each job is assigned to the machine and server that minimizes its completion time

$$C_i = \min_{r,k} \{\max(R_k, Q_r) + s_{irk} + p_{ik}\} \tag{9}$$

where R_k and Q_r are the current availability times of machine k and server r, which are then updated.

4.5 Min-Min

The classical Min–Min heuristic repeatedly selects the job and resource combination that can be completed the earliest. To adapt it, at each step, for every unscheduled job i, we compute its minimum possible completion time over all machine–server pairs Eq. 9. We then select the job with the smallest C_i min and assign it to the corresponding machine and server and update R_k and Q_r. Unlike MCT, which follows a fixed job order, Min–Min evaluates all unscheduled jobs at every step, which improves solution quality at the expense of higher time complexity.

5 Experimental results and comparison

For this study, we generated two datasets inspired by the methodology of [15]. Both share common processing times distribution that are drawn from $U(10, 40)$, and the setup times follow $U(\lfloor \alpha \cdot 10 \rfloor, \lceil \alpha \cdot 40 \rceil)$ where α is the setup ratio.

The first set, hereafter referred to as the scaling dataset, serves as our primary benchmark. It replicates a standard configuration of $m = 2$, $s = 2$, and $\alpha = 0.25$ across three job scales: small ($n \in \{10, 15, 20, 25\}$), medium ($n \in \{50, 60, 70, 80\}$), and large ($n \in \{100, 150, 200, 250\}$).

The second set, the factorial dataset, is an extension designed to isolate and test the impact of system complexity and setup intensity on large-scale problems. Therefore, we fixed the job count at $n = 100$ and employed a 4×4 factorial design, combining four machine-server configurations $(m, s) \in \{(2, 2), (2, 4), (4, 2), (4, 4)\}$ with four setup ratios $\alpha \in \{0.1, 0.25, 0.5, 0.75\}$.

By generating 10 instances for every parameter combination in both sets (280 instances in total), we created a robust environment to evaluate our algorithms under both standard conditions and complex scenarios. The dataset used in this study is publicly available at https://github.com/Nick-Liou/upms-benchmarks.

The scaling experiments were conducted on an Intel Core i9-10900K processor, while the factorial experiments utilized an Intel Core i7-14700KF. The software environment consisted of Python 3.10.12 and MiniZinc v2.9.4 [16] using the OR-Tools CP-SAT solver v9.14.6206 [17]. Solver parameters were set to optimization level 1, a three-hour time limit per instance, and the maximum available CPU threads. The execution times reported in Tables 2 and 5 include both the compilation (flattening to FlatZinc) and the solving duration. Conversely, for the comparison between the CP approach and heuristics in Fig. 3, Python wall-clock time was used to ensure a uniform baseline.

For each instance, the CP-SAT solver and heuristics were executed once, and the reported statistics were computed across instances of identical size. For all primary benchmark instances, an optimal solution was found within the specified time limit, notably the largest instances ($n = 250$) were solved in less than two hours on average. In contrast, for the second benchmark, the optimal solution was not reached within the time limit for certain instances involving 4 machines and 2 servers.

Table 2. Mean and Standard Deviation (SD) of objective value and runtime as a function of the number of jobs n for CP-SAT, computed over independent instances.

n	**Mean objective**	**SD objective**	**Mean run time (s)**	**SD run time (s)**
10	131.2	12.96	0.38	0.08
15	186.7	12.22	1.38	0.47
20	240.5	18.72	2.98	0.88
25	307.8	19.88	6.02	1.35
50	613.4	22.28	81.22	22.21
60	731.0	33.52	132.36	21.19
70	836.8	35.81	163.33	35.14
80	984.4	45.35	272.37	66.93
100	1221.6	32.14	550.35	200.93
150	1814.5	39.33	1370.43	266.69
200	2407.5	47.63	3127.56	699.65
250	3035.5	82.49	5824.90	744.31

The mean objective values of both the optimal solutions and the heuristic solutions, shown in Fig. 2, appear to follow an approximately linear trend in the range of job counts in the scaling dataset. Notably, the variability of the optimal objective value generally decreases as the size of the problem increases, which is evident from the drop of the coefficient of variation from approximately 9% for $n = 10$ to below 3% for $n = 250$, indicating that larger instances exhibit increasingly homogeneous objective values relative to their scale. This suggests

Table 3. Run times (s) of CP-SAT and Gap (%) of Min-Min for the factorial set.

(m, s)	**Metric**	α **(Setup Time Ratio)**			
		0.10	**0.25**	**0.50**	**0.75**
$(2, 2)$	Run Time	420	393	629	1035
	Gap (%)	1.16	1.27	1.22	1.21
$(2, 4)$	Run Time	1261	765	955	1497
	Gap (%)	0.87	0.96	0.59	1.01
$(4, 2)$	Run Time	6477	10000	10800	10800
	Gap (%)	3.53	3.42	5.34	3.56
$(4, 4)$	Run Time	1899	2064	3030	3784
	Gap (%)	3.51	3.83	3.74	3.85

that the structural differences of the specific instance have a diminishing impact on solution quality as the number of jobs grows.

In contrast, runtime variability for the CP-SAT solver remains substantial across all problem sizes. The coefficient of variation ranges between 12% and 35%, showcasing significant instance variability in computational effort. Therefore, as one might expect, while the problem size is the main driver of runtime growth, structural properties of individual instances play a major role in determining solver performance.

As expected, the objective values of the solutions computed from the heuristics are not optimal. Thus, to qualitatively measure their quality, we use the percentage gap from the corresponding optimal makespan, as expressed in Eq. 10, which is always non-negative. The gaps can be found in Table 4 that clearly show that the best performing heuristic for all problem sizes is Min-Min.

$$Gap(\%) = \frac{Heuristic_{makespan} - Optimal_{makespan}}{Optimal_{makespan}} \cdot 100\% \tag{10}$$

Figure 4 illustrates the improvement in solution quality over time of the CP-SAT solver for instances with $n = 200$ jobs. The solver quickly finds solutions within seconds, which in the first few seconds to minutes significantly improves and then gradually enhances them until they reach optimality. The median trajectory shows that 90% of the final solution quality is typically achieved within the first 10-15% of total runtime, demonstrating effective early exploration followed by refinement. If we significantly prioritize the time over the solution quality, then the best performing heuristic Min-Min, which consistently achieves the lowest gap compared to the others, offers an irresistible balance between minimizing the makespan and execution time.

The results for the factorial dataset ($n = 100$) are presented in Table 3, showing the relationship between resource configurations and setup intensity. It is important to note that for the machine-server configuration $(4, 2)$, several instances were not solved to optimality within the time limit. In these cases,

the percentage gap for the Min-Min heuristic was calculated relative to the best feasible solution found by the CP-SAT solver rather than the optimal makespan. The data shows that problem difficulty is highly sensitive to the machine-server ratio, and the $(4, 2)$ configuration proves the most challenging, likely due to server scarcity creating a significant bottleneck. While Min-Min remains the best performing heuristic, its gaps with multiple machines are notably higher often exceeding 3.5%. It should be emphasized that actual gap would likely be even higher if the true optimal solutions were available. Furthermore, as the setup ratio α increases, the computational effort generally rises, confirming that higher setup intensity complicates the coordination between resources and slows the solver's convergence.

Finally, we investigated the sensitivity of the solver to the upper bound T (defined in Eq. 1) by introducing a multiplier, $t_{modifier}$, ranging from 0.6 to 1.55 with a 0.05 step. This experiment was conducted across three runs for all $n = 100$ instances in the scaling dataset. Interestingly, the overall impact on mean runtime was relatively marginal, however a performance degradation was observed when the bound was very strict and approached the optimal value. In these cases, the overly restrictive bound appeared to force the solver into frequent "dead ends" within the search tree, hindering its ability to prune the search space effectively and leading to slightly higher computational overhead compared to more relaxed bounds.

Table 4. Mean gap (%) from optimal solution across problem sizes for the constructive heuristics.

n	Random	MR (100)	MR (1000)	SPT	MCT	Min-Min
10	59.38	19.86	9.69	26.51	12.29	**8.44**
15	64.23	26.14	19.24	37.79	12.69	**4.43**
20	72.80	32.86	25.52	30.22	15.00	**4.38**
25	82.22	39.03	29.79	33.19	14.59	**3.74**
50	70.22	44.26	35.50	34.42	12.17	**2.32**
60	68.92	47.25	39.10	34.43	11.86	**2.21**
70	72.48	46.82	42.05	32.33	11.30	**1.82**
80	64.74	47.02	44.34	31.71	11.51	**1.76**
100	69.52	49.58	46.10	30.29	11.59	**0.89**
150	70.31	54.87	50.32	32.84	11.24	**0.69**
200	72.01	56.93	52.05	33.16	10.77	**0.55**
250	73.48	57.77	54.80	31.49	11.40	**0.49**

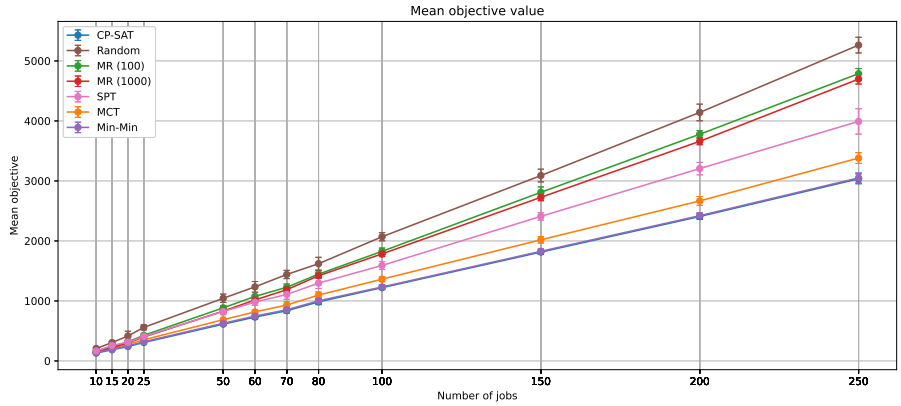

Fig. 2. Mean objective value as a function of the number of jobs n. Error bars indicate one standard deviation across instances.

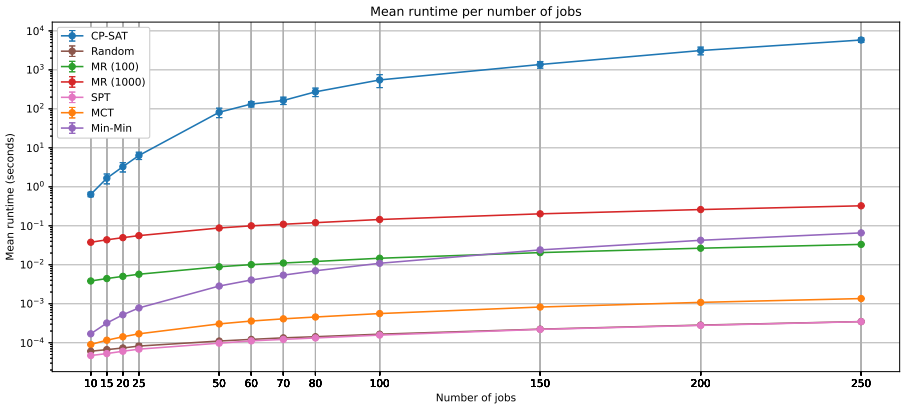

Fig. 3. Mean runtime as a function of the number of jobs n. Error bars indicate one standard deviation across instances.

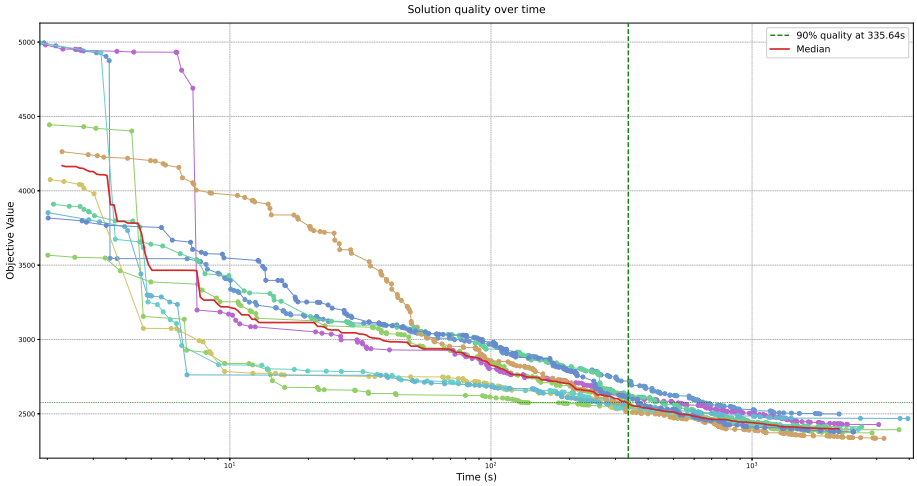

Fig. 4. Objective value over time for instances where $n = 200$ for the CP-SAT.

6 Limitations & future work

Although the proposed CP model is capable of solving almost all benchmark instances to optimality, several limitations highlight important directions for future research.

First, for the generation of the benchmark specific parameter values were used, for example the processing times distribution. Although this controlled setting is useful for isolating model behavior, it does not reflect the full range of conditions encountered in practice. Future work should therefore investigate a broader spectrum of system sizes and parameter values. In addition, experimenting with alternative processing-time distributions would help assess how robust the model is under different workload characteristics.

The second limitation concerns scalability. As shown in Section 5, runtimes increase rapidly with instance size and the number of machines. Although optimal solutions can still be obtained, solving larger instances to proven optimality quickly becomes computationally prohibitive. In real industrial environments, where schedules often need to be generated or updated in near real time, relying exclusively on exact CP methods may therefore be impractical.

To address this, the constructive heuristics offer value beyond just a point of comparison. While the CP-SAT solver's architecture does not support warm-starting with these solutions, the heuristic results serve as a vital baseline for gauging the solver's performance. Experimental testing indicated that using tighter upper bounds from these heuristics did not yield significant performance gains, on the contrary overly restrictive boundaries appeared to interfere with the solver's internal search heuristics and propagation logic. This suggests that the CP model is capable of navigating the search space effectively.

More generally, the computational effort required for large instances motivates the exploration of better approaches that trade optimality guarantees for faster solution times. Metaheuristics such as Variable Neighborhood Search (VNS) [5] or Simulated Annealing [19] could produce even higher-quality schedules within strict time limits compared to heuristics, making them attractive for large-scale or time-critical applications.

Finally, several characteristics of real-world scheduling environments have not yet been captured by the current model. Extending the formulation to incorporate additional practical constraints would improve its applicability. Examples include the introduction of job release dates and deadlines, more complex resource structures with shared and dedicated resources, and dynamic job arrivals that require online or rolling-horizon scheduling. It would also be interesting to move beyond a single-objective perspective and consider multi-objective formulations, e.g., jointly optimizing makespan and energy consumption.

7 Conclusions

In this paper, we introduced a novel scheduling problem that involves unrelated parallel machines and unrelated servers required for job setup. This setting is motivated by practical production environments in which setup operations depend on multiple factors. To encourage reproducibility and support future research, we also proposed a new set of benchmarks that span a wide range of instance sizes.

To solve this problem, we developed an exact constraint programming formulation in MiniZinc and solved it using OR-Tools CP-SAT. The computational results demonstrate that the proposed model can prove optimality for almost all 280 benchmark instances, including cases with up to 250 jobs, within a time limit of three hours. These results show that CP is a viable exact solution approach for this new problem class at moderate scales and that modern CP solvers are capable of handling large number of constraints.

In addition to the exact approach, we adapted several common heuristics as baseline methods. Although used primarily for comparison, these heuristics also provided useful insight into the structure of the problem and the quality of solutions that can be obtained with very limited computational effort. Their performance highlights the potential of using fast constructive or improvement heuristics not only as standalone methods but also possibly as part of hybrid solution strategies.

Our experiments also revealed several notable empirical patterns. Firstly, high-quality solutions are typically found early in the CP search, with solutions within 90% of optimality often obtained in a small fraction of the total runtime. Later search stages are mainly devoted to closing the remaining optimality gap. Secondly, we observed substantial runtime variability between instances of identical size, suggesting that structural characteristics play a significant role in determining computational difficulty. Thirdly, the upper bound T does not

play a significant role in the solver performance unless it is very restrictive and hinders its ability to effectively explore the solution space.

Overall, this work establishes the first exact optimization framework for the proposed scheduling problem, together with a publicly available benchmark set and a validated CP model. These contributions provide a solid foundation for future research. Promising directions include the development of hybrid CP–heuristic approaches, the design of faster approximation or metaheuristic methods for large-scale and time-critical settings, and a deeper investigation of structural instance properties to further improve real-world applications.

Acknowledgments. This paper is a result of research conducted within the "MSc in Artificial Intelligence and Data Analytics" of the Department of Applied Informatics of University of Macedonia. The presentation of the paper is funded by the University of Macedonia Research Committee.

References

1. Benbrik, O., Benmansour, R., Elidrissi, A., Sifaleras, A.: Advanced algorithms for the reclaimer scheduling problem with sequence-dependent setup times and availability constraints. In: Metaheuristics International Conference. MIC 2024. Lecture Notes in Computer Science. vol. 14753, pp. 291–308. Springer, Cham (2024)
2. Benmansour, R., Sifaleras, A.: Scheduling in parallel machines with two servers: the restrictive case. In: Variable Neighborhood Search. ICVNS 2021. Lecture Notes in Computer Science. vol. 12559, pp. 71–82. Springer (2021)
3. Benmansour, R., Sifaleras, A., Mladenović, N. (eds.): Variable Neighborhood Search. 7th International Conference, ICVNS 2019, Rabat, Morocco, October 3-5, 2019, Revised Selected Papers, LNCS, vol. 12010. Springer, Cham (2020)
4. Benmansour, R., Sifaleras, A., Todosijevic, R.: Constraint programming and simulated annealing approaches for parallel-machine scheduling with conflict constraints, server setups, and flexible maintenance. In: Proceedings of the 15th International Conference on Operations Research and Enterprise Systems (ICORES 2026). pp. 302–310. Marbella, Spain (2026)
5. Brimberg, J., Salhi, S., Todosijević, R., Urošević, D.: Variable neighborhood search: The power of change and simplicity. Computers & Operations Research **155**, 106221 (2023)
6. Brucker, P., Dhaenens-Flipo, C., Knust, S., Kravchenko, S.A., Werner, F.: Complexity results for parallel machine problems with a single server. Journal of Scheduling **5**(6), 429–457 (2002)
7. Elidrissi, A., Benbrahim, M., Benmansour, R., Duvivier, D.: Variable neighborhood search for identical parallel machine scheduling problem with a single server. In: International Conference on Variable Neighborhood Search. pp. 112–125. Springer (2019)
8. Elidrissi, A., Benmansour, R., Hasani, K., Werner, F.: Minimizing the makespan on two parallel machines with a common server in charge of loading and unloading operations. Computers & Operations Research **167**, 106638 (2024)
9. Elidrissi, A., Benmansour, R., Sifaleras, A.: A general variable neighborhood search for the parallel machine scheduling problem with two common servers. Optimization Letters **17**(9), 2201–2231 (2023)

10. Hariri, A., Potts, C.N.: Heuristics for scheduling unrelated parallel machines. Computers & Operations Research **18**(3), 323–331 (1991)
11. Ibarra, O.H., Kim, C.E.: Heuristic algorithms for scheduling independent tasks on nonidentical processors. Journal of the ACM **24**(2), 280–289 (1977)
12. Jiang, Y., Zhou, P., Wang, H., Hu, J.: Scheduling on two parallel machines with two dedicated servers. The ANZIAM Journal **58**(3-4), 314–323 (2017)
13. Kim, M.Y., Lee, Y.H.: Mip models and hybrid algorithm for minimizing the makespan of parallel machines scheduling problem with a single server. Computers & Operations Research **39**(11), 2457–2468 (2012)
14. Kravchenko, S.A., Werner, F.: Parallel machine scheduling problems with a single server. Mathematical and Computer Modelling **26**(12), 1–11 (1997)
15. Krimi, I., Benmansour, R.: Self-adaptive general variable neighborhood search algorithm for parallel machine scheduling with unrelated servers. Computers & Operations Research **163**, 106480 (2024)
16. Nethercote, N., Stuckey, P.J., Becket, R., Brand, S., Duck, G.J., Tack, G.: Minizinc: Towards a standard CP modelling language. In: Bessière, C. (ed.) Principles and Practice of Constraint Programming – CP 2007. pp. 529–543. Springer Berlin Heidelberg, Berlin, Heidelberg (2007)
17. Perron, L., Didier, F.: CP-SAT, https://developers.google.com/optimization/cp/cp_solver
18. Pinedo, M.L.: Planning and scheduling in manufacturing and services. Springer New York, NY, 2 edn. (2009)
19. Rolim, G.A., Nagano, M.S.: Designing state-of-the-art metaheuristics: What have we learned from the parallel-machine scheduling problem with setups? Computers & Operations Research p. 107110 (2025)
20. Silva, J.M.P., Teixeira, E., Subramanian, A.: Exact and metaheuristic approaches for identical parallel machine scheduling with a common server and sequence-dependent setup times. Journal of the Operational Research Society pp. 1–15 (2019)
21. Đurasević, M., Jakobović, D.: Heuristic and metaheuristic methods for the parallel unrelated machines scheduling problem: a survey. Artificial Intelligence Review **56**(4), 3181–3289 (2023)