Constraint Programming and Simulated Annealing Approaches for Parallel-Machine Scheduling with Conflict Constraints, Server Setups, and Flexible Maintenance

Rachid Benmansour[1][a], Angelo Sifaleras[2][b] and Raca Todosijevic[3][c]
[1]*Research Laboratory in Information Systems, Intelligent Systems and Mathematical Modeling, INSEA, Rabat, Morocco*
[2]*Department of Applied Informatics, School of Information Sciences, University of Macedonia, 156 Egnatia Str., Thessaloniki 54636, Greece*
[3]*Univ. Polytechnique Hauts-de-France, LAMIH, CNRS, UMR 8201, F-59313 Valenciennes, France*

Keywords: Scheduling, Parallel Machines, Conflict Constraints, Single Server, Maintenance, Manufacturing-as-a-Service.

Abstract: This paper addresses a parallel-machine scheduling problem where jobs require setups performed by a single server and must respect conflict constraints that prevent certain jobs from running simultaneously. This type of problem can find applications in logistics and transport operations, particularly when scheduling vehicle fleets that share limited resources. The server is also subject to a fixed-duration maintenance activity that must be scheduled alongside the jobs. The objective is to minimize the makespan. We develop both constraint programming and simulated annealing approaches to solve this problem. Experimental results demonstrate that the constraint programming model, executed on the Minizinc solver, successfully obtains optimal solutions for small instances with 10 jobs. For larger problems with 15 jobs, simulated annealing is a good alternative since it allows to obtain solutions at 2.3% of the optimum on average despite a fixed calculation time of 10 seconds.

1 INTRODUCTION

Machine scheduling problems are fundamentally concerned with the assignment of jobs to machines while respecting a set of operational constraints and optimizing one or more performance objectives, such as the makespan $C_{\max}$. In many practical industrial applications, these problems become particularly complex because of two key factors. First, jobs often require a setup operation that must be performed by a single shared server. Second, these jobs cannot be processed simultaneously due to technological limitations or resource conflicts. Furthermore, unlike other problems in the literature, the server is not available all the time. Its availability is interrupted by necessary maintenance activities, which significantly influences the overall scheduling performance.

This paper addresses a scheduling problem that integrates these elements: a set of identical parallel machines, a single setup server, pairwise job conflicts, and planned preventive maintenance on the server. Each real job requires a non-preemptive setup on the server followed by non-preemptive processing on a dedicated machine. Maintenance is modeled as a dummy job executed on a fictitious (or dummy) machine that occupies the server for a fixed duration within a specified time window. The objective is to minimize the makespan.

This problem integrates the classical *Parallel Machine Scheduling with a Single Server* (PSS) and *Parallel Machine Scheduling with Conflicts* (PMC), both of which are NP-hard. The PMCS problem can be expressed in the three-field notation as

$$Pm, S1|conflict, p_j, s_j, nr-a|C_{max}$$

where m is the number of identical parallel machines, $S1$ indicates the single setup server, p_j and s_j are the processing and setup times of job j, and the objective is to minimize the makespan C_{max} and *nr-a* denotes a nonresumable availability constraint. Note that a job is considered nonresumable if, when interrupted by a machine's unavailability, it must be restarted from

[a] https://orcid.org/0000-0003-2553-4116
[b] https://orcid.org/0000-0002-5696-7021
[c] https://orcid.org/0000-0002-9321-3464

Benmansour, R., Sifaleras, A. and Todosijevic, R.
Constraint Programming and Simulated Annealing Approaches for Parallel-Machine Scheduling with Conflict Constraints, Server Setups, and Flexible Maintenance.
DOI: 10.5220/0014306700004055

In *Proceedings of the 15th International Conference on Operations Research and Enterprise Systems (ICORES 2026)*, pages 302-310
ISBN: 978-989-758-799-3; ISSN: 2184-4372

the beginning rather than continued from the point of interruption. While the literature provides extensive methodologies for problems involving either a single setup server or conflict constraints separately, the integrated problem remains unexplored. We term this novel problem the Parallel Machine scheduling with a single server, Conflict constraints, and Unavailability constraints (denoted as PMCSU).

To the best of our knowledge, no prior research has investigated these three critical constraints simultaneously. We formulate a constraint programming (CP) model to capture these constraints, including server and machine capacities, job conflicts, symmetry-breaking for machine assignments (Ostrowski et al., 2010), and maintenance scheduling. Computational experiments are conducted on systematically generated instances to evaluate the effectiveness of the CP model. The remainder of the paper presents the CP model, the instance generation procedure, the experimental setup, and the results.

2 LITERATURE REVIEW

Parallel machine scheduling with a single shared server has been studied extensively, as in the classical problem $Pm, S1|p_j, s_j|C_{\max}$. The early results, by Kravchenko and Werner, provided a pseudo-polynomial algorithm for two machines with unit setup times and established unary NP-hardness in the general case (Kravchenko and Werner, 1997), while (Hall et al., 2000) and (Brucker et al., 2002) extended the complexity analysis to the analysis to $m \geq 2$. Subsequent research developed exact and heuristic methods, including branch-and-price algorithms (Gan et al., 2012), MILP formulations with hybrid metaheuristics (Kim and Lee, 2012), and arc-time-indexed MILP solved by iterated local search (Silva et al., 2019). Recent approaches such as variable-neighborhood or GVNS heuristics (Elidrissi et al., 2019; Elidrissi et al., 2024) and constraint-programming models (Heinz et al., 2022) further demonstrate the challenge of these problems, even without additional constraints. The problem of scheduling jobs on parallel machines with shared servers has several real applications, particularly in logistics. A notable example is the real-world Biomass Truck Scheduling (BTS) problem (Torjai and Kruzslicz, 2016), which models trucks as parallel machines and a central biorefinery operating a single unloader (single server). In the same vein, (Benmansour et al., 2024) addressed a system with multiple unloading servers by proposing a General Variable Neighborhood Search (GVNS) metaheuristic, which finds high-quality solutions in seconds to face daily operational planning.

Scheduling with conflict constraints addresses situations where certain jobs cannot overlap due to resource limitations or technological requirements. Kowalczyk and Leus proposed a branch-and-price method combining bin-packing and graph-coloring techniques (Kowalczyk and Leus, 2017), while Hong and Lin provided NP-hardness results and polynomial solutions for specific two-machine cases (Hong and Lin, 2018). More general MILP and CP frameworks were developed later (Hà et al., 2021) and scaled to larger instances (Dinh et al., 2024). These contributions offer essential modeling insights for handling conflict-graph restrictions in parallel-machine environments.

Finally, machine unavailability and planned maintenance have been extensively studied. Surveys such as the one provided by Kacem (Kacem, 2007) summarize some models with breakdowns and maintenance periods, while Lee (Lee, 1996) and Schmidt (Schmidt, 2000) provided exact, heuristic and approximation methods. These results help decision makers integrate maintenance activities into scheduling problems.

The PMCSU problem considered here combines these three challenging aspects: a single shared setup server, pairwise conflict constraints, and a planned maintenance period. Although separate methods exist for each component, no previous work addresses the integrated problem. Insights from the single-server, conflict, and maintenance literature provide the foundation for our CP model and will inform the development of alternative exact and heuristic approaches.
Scheduling challenges are also encountered in contemporary Manufacturing-as-a-Service (MaaS) online platforms. These distributed manufacturing environments typically break down scheduling requests and assemble supply chains through sophisticated artificial intelligence and optimization methods. Recently, Kaparis et al. (Kaparis et al., 2025) introduced a MaaS framework that integrates production planning, semantic interoperability, and service modeling to enable flexible configuration of manufacturing value chains. The research methodology presented in this paper will likewise be utilized to tackle comparable real-world optimization problems that occur on the Tec4MaaSEs platform http://tec4maases.eu.

3 PROBLEM DEFINITION

We consider a scheduling problem involving a set of identical parallel machines $M = \{1, \ldots, m\}$ that share

a single setup server. A set of jobs $J = \{1, \ldots, n\}$ are available at the beginning of the schedule and must be executed on the machine after the setup operation. More formally, each job i requires a setup operation, performed by the shared server, before it can be processed on its assigned machine. The setup is non-preemptive, lasts s_i time units, and must be immediately followed by non-preemptive processing of duration p_i. While a job is being set up, both the server and the assigned machine are busy. In addition, we consider that certain pairs of jobs cannot be processed concurrently due to technological or resource constraints. These incompatibilities are modeled by an undirected conflict graph $G = (J, \mathcal{C})$, where an edge $(i, j) \in \mathcal{C}$ indicates that the processing intervals of jobs i and j should not overlap.

The setup server is also subject to fixed duration planned maintenance $U_p = 1$, which must start no later than a given time $T = \frac{1}{m}\sum_{i \in J} p_i$. To model this, a dummy job $n+1$ is introduced with setup duration $s_{n+1} = U_p$ and processing time $p_{n+1} = 0$, assigned exclusively to a dummy machine $m+1$. This ensures that the server is unavailable for exactly U_p consecutive time units while the dummy machine handles its processing independently, so that the real machines are not affected.

A feasible schedule must assign each real job to exactly one real machine, and specify setup and processing start times such that the server performs at most one setup at a time, including the fictitious job's setup interval. In addition, each real machine processes at most one job at any time, considering both setup and processing intervals. In addition, conflicting jobs do not have overlapping processing intervals. In other words, the dummy maintenance job starts no later than T and occupies the server for U_p consecutive time units. The objective is to minimize the makespan $C_{\max} = \max_{i \in J} C_i$, where C_i denotes the completion time of the job i. The makespan is computed over the real jobs only; the maintenance job may indirectly delay them by occupying the server.

4 CONSTRAINT PROGRAMMING MODEL

We propose a constraint programming (CP) formulation for the scheduling problem described in Section 3. The model assigns each real job to exactly one real machine and determines the start times of setup and processing for all jobs, including the dummy maintenance job, with the objective of minimizing the makespan $C_{\max}$ of real jobs. CP was chosen because it allows a more natural modeling of the scheduling constraints, shared setup server, and job incompatibilities than MILP.

Let $J' = J \cup \{n+1\}$ denote the set of jobs including the dummy maintenance job, and $M' = M \cup \{m+1\}$ denote the set of machines including the dummy machine. The decision variables for all $i \in J'$ are:

- st_i: start of the setup
- t_i: processing start
- C_i: completion
- $machine_i$: assigned machine
- $C_{\max}$: $\max_{i \in J} C_i$

Objective:

$$\min C_{\max}.$$

Constraints:

$$\forall i \in J': \quad t_i = st_i + s_i \tag{1}$$

$$\forall i \in J': \quad C_i = t_i + p_i \tag{2}$$

$$\text{disjunctive}(\{[st_i, st_i + s_i]\}_{i \in J'}) \tag{3}$$

$$\forall k \in M: \tag{4}$$

$$\text{cumulative}([st_i]_{i \in J}, [s_i + p_i]_{i \in J}, [\mathbb{1}_{machine_i = k}]_{i \in J}, 1) \tag{5}$$

$$\forall (i, j) \in \mathcal{C}: \quad C_i \leq t_j \vee C_j \leq t_i \tag{6}$$

$$\forall i \in J, \forall k \in M \setminus \{1\}: \tag{7}$$

$$\text{if } machine_i = k \text{ then } \sum_{j < i} \mathbb{1}_{machine_j = k-1} \geq 1 \tag{8}$$

$$machine_{n+1} = m+1 \tag{9}$$

$$st_{n+1} \leq T \tag{10}$$

$$s_{n+1} = U_p \tag{11}$$

$$p_{n+1} = 0 \tag{12}$$

$$\forall i \in J: \quad C_{\max} \geq C_i \tag{13}$$

The objective of the model is to minimize the makespan $C_{\max}$ over all real jobs J, i.e., $\min C_{\max}$. Each job $i \in J'$ must respect the temporal relationships: the processing start time t_i immediately follows the setup start st_i (1) and the completion time C_i follows the end of processing (2). The setup server can handle at most one setup at a time, including the dummy maintenance job (3). Real machines $k \in M$ can process at most one job at a time, accounting for both setup and processing intervals (5). Any pair of conflicting jobs $(i, j) \in \mathcal{C}$ must not have overlapping processing intervals (6). To reduce symmetry

in the search space, if a job $i \in J$ is assigned to machine $k > 1$, then at least one job scheduled earlier must be assigned to machine $k-1$ (8). The dummy maintenance job $n+1$ ensures that the server is unavailable for U_p consecutive time units: it is assigned to the dummy machine $m+1$ (9), must start no later than time T (10), has setup duration $s_{n+1} = U_p$ (11) and zero processing time $p_{n+1} = 0$ (12). Finally, the makespan $C_{\max}$ is defined as the maximum completion time over all real jobs $i \in J$ (13).

4.1 Example with Maintenance

Consider an instance with $n = 10$ real jobs scheduled on $m = 2$ identical parallel machines.

Table 1: Setup and processing times for the 10 real jobs in the instance with maintenance.

Job i	1	2	3	4	5	6	7	8	9	10
s_i	3	1	1	5	2	1	2	2	2	1
p_i	3	1	2	10	4	1	2	3	10	1

While the dummy maintenance job has $s_{11} = U_p = 1$ and $p_{11} = 0$. The conflicts among real jobs are $\{(1,6),(2,9),(4,5),(3,4),(3,5)\}$, meaning that jobs 1 and 6, 2 and 9, 3 and 4, 3 and 5 as well as 4 and 5 cannot overlap in their processing intervals.

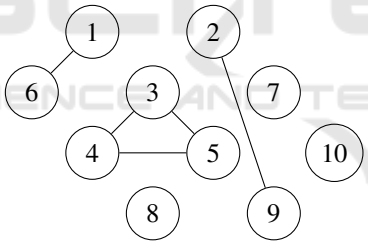

Figure 1: Conflict graph for the 10 jobs instance with maintenance.

The dummy maintenance job ensures that the server is unavailable for $U_p = 1$ time unit, and it must start no later than $T = 19$. In any feasible schedule, jobs $(1,6)$, $(2,9)$, and $(4,5)$ must not overlap. All real jobs must be assigned to real machines 1 or 2, while the maintenance job is assigned to the dummy machine 3. The single server sequences all setups, including the maintenance setup.

For this instance, an optimal schedule respecting all conflicts and the maintenance constraint achieves a makespan of $C_{\max} = 29$. Jobs must be sequenced such that conflicting jobs do not overlap, setups on the server are non-overlapping, and the maintenance setup occurs before time T.

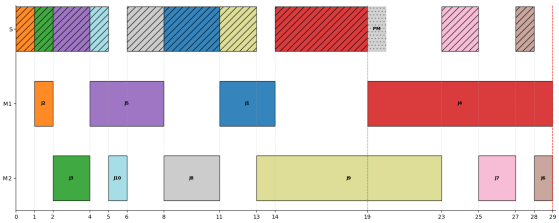

Figure 2: Gantt chart of the 10-job instance with maintenance, $C_{\max} = 29$.

5 SIMULATED ANNEALING ALGORITHM

We develop a simulated annealing algorithm to solve the problem studied. Simulated annealing is inspired by metallurgy process, where controlled heating and cooling helps metals to achieve better crystalline structures. In optimization context, this means we sometimes accept worse solutions to escape from local optima and find better schedules. The method was first introduced by (Kirkpatrick et al., 1983) for solving the traveling salesman problem in 1983.

5.1 Algorithm Implementation

In the proposed heuristic, a solution is encoded as a permutation of jobs that specifies the order in which setup operations are performed on the single server. Given such an order, a deterministic decoder constructs a complete schedule by assigning each job to the earliest available machine and start time while respecting (i) server availability, (ii) the non-preemptive nature of setup and processing, and (iii) incompatibilities in the conflict graph. Moreover, for a fixed server order, the placement of the maintenance activity reduces to choosing its best insertion position in the sequence (before the first setup, between consecutive setups, or after the last setup), subject to the deadline T.

In our implementation, the function EVALUATEWITHMAINTENANCE serves as a deterministic decoder. It takes a permutation of jobs and evaluates its quality while automatically determining the optimal maintenance placement. Given a permutation, it systematically tests all admissible insertion positions for maintenance activity up to the deadline T. At each candidate position, it constructs the complete schedule by simulating the execution of jobs and maintenance on the server and machines, respecting all constraints including server capacity, machine availability, and job conflicts. The function returns the minimum achievable makespan by selecting the best maintenance position.

The proposed simulated annealing (SA) heuristic is summarized in Algorithm 1. SA is initialized by GENERATERANDOMSOLUTION, which draws a random permutation of jobs. Once an initial solution is obtained, the main loop starts. At each iteration of the loop, the GENERATENEIGHBOR function produces a new candidate solution by applying one of three classical neighborhood operators to the current solution: swap, insert, and 2-opt. The swap operator exchanges the positions of two randomly selected jobs in the sequence (with probability p_{insert}). The insertion operator moves a single job from its current position to a different random position in the sequence (with probability p_{swap}). The 2-opt operator (with probability $p_{reverse}$) reverses a randomly selected subsequence of jobs, providing more substantial changes to the solution structure. These neighborhood moves maintain the feasibility of the solution while allowing effective exploration of the search space. Each candidate solution is evaluated using the EVALUATEWITHMAINTENANCE function.

Algorithm 1: Simulated Annealing with Integrated Maintenance.

Function SimulatedAnnealing(n, m, C, U_p, T)**:**
  $S \leftarrow$ GENERATERANDOMSOLUTION(n);
  $S_{best} \leftarrow S$;
  $f_{best} \leftarrow$ EVALUATEWITHMAINTENANCE(S_{best}, U_p, T);
  $\tau_{current} \leftarrow \tau_0$;
  $t_{start} \leftarrow$ CURRENTTIME();
  while CURRENTTIME() - $t_{start} < t_{max}$ **do**
    $S' \leftarrow$ GENERATENEIGHBOR(S);
    $f(S') \leftarrow$ EVALUATEWITHMAINTENANCE(S', U_p, T);
    $\Delta f \leftarrow f(S') - f(S)$;
    if $\Delta f < 0$ ***or*** RANDOM() $< \exp(-\Delta f / T_{current})$ **then**
      $S \leftarrow S'$, $f(S) \leftarrow f(S')$;
    if $f(S) < f_{best}$ **then**
      $S_{best} \leftarrow S$, $f_{best} \leftarrow f(S)$;
    $\tau_{current} \leftarrow \alpha \tau_{current}$;
  end
  return S_{best};

In Algorithm 1, S represents the current solution (permutation of jobs), while S_{best} stores the best solution found so far during the search. The variable f_{best} keeps track of the best makespan value encountered. $\tau_{current}$ is the current temperature that controls the acceptance probability of worse solutions, starting from the initial temperature τ_0. The variable t_{start} records the starting time of the algorithm to enforce the time limit t_{max} in seconds.

It should be noted that, in preliminary testing, we tried a two-phase approach: first, we optimized the schedule while *ignoring* maintenance, using SA to obtain a high-quality server order; second, holding that order fixed, we invoked EVALUATEWITHMAINTENANCE to insert the maintenance before the deadline T. This approach consistently produced worse results than the integrated one, often yielding noticeably larger makespans. This outcome confirms that maintenance and server order are *non-separable* decisions. Therefore, working on a relaxed (no-maintenance) objective may lead to very poor solutions.

5.2 SA Algorithm Parameters

After some preliminary computational tests, we identified effective parameter settings for the proposed SA: initial temperature $\tau_0 = 100$, cooling rate $\alpha = 0.95$, and time limit $t_{max} = 10$ seconds. These parameters strike a balance between sufficient exploration of the solution space and convergence to high-quality solutions within practical computation times.

For neighborhood selection, we use the probabilities $p_{swap} = 0.4$ for swap, $p_{insert} = 0.4$ for insertion, and $p_{reverse} = 0.2$ for 2-opt. This distribution favors the more refined local search moves (swap and insert) while still including the more disruptive 2-opt move to escape local minima. The swap and insert operators provide gradual improvements, while the 2-opt operator enables more significant structural changes to the solution.

6 INSTANCE GENERATION

Benchmark instances are generated to capture the structure of parallel machine scheduling problems with a single setup server and job conflicts. Each instance consists of n real jobs to be executed on m identical machines, with a single dummy maintenance job $n+1$. Setup times for real jobs satisfy $s_i \geq 1$, processing times $p_i \in [1, b]$, and the dummy maintenance job has $s_{n+1} = U_p = 1$ and $p_{n+1} = 0$. The server maintenance operation must start no later than $T = \lceil \sum_{i=1}^{n} p_i / m \rceil$, ensuring that it takes place during the schedule. Conflict graphs between real jobs are generated using the Erdős–Rényi model $G(n, d)$, where each of the $\frac{n(n-1)}{2}$ potential edges is included with probability d. An edge (i, j) indicates that the jobs i and j cannot overlap.

Instances are characterized by the triple parameter (n, d, b), where n is the number of machines, d is the probability used to generate a conflict graph and b are

the upper bounds of processing time. For each triple (n,d,b), the number of machines, m, is varied within the set $m \in \{2,3,5\}$. For every unique combination of these parameters (n,d,b,m), we generate 10 independent instances using different random seeds to ensure reproducibility.

The experimental design considers job-set sizes $n \in \{10,15\}$, conflict probabilities $d \in \{0.1,0.3,0.5\}$, and processing-time upper bounds $b \in \{10,50\}$. Here, n controls the number of real jobs to schedule, d determines the density of the Erdős–Rényi conflict graph, and b specifies the maximum processing time from which individual job durations p_i are uniformly sampled. These parameter ranges allow us to explore both small and medium problem sizes, low to high levels of job conflicts, and short versus long processing-time distributions, thereby providing a balanced set of benchmark scenarios for evaluating the constraint programming model. All instances, including larger ones, will be made publicly available in an extended version of this work.

7 EXPERIMENTAL ANALYSIS

We run all of our experiments on a workstation with an Intel Core i7-12700H processor and 16 GB of RAM. For the exact method, we uses MiniZinc version 2.9.3 (Van Hentenryck and Michel,) with the Chuffed solver (Chu et al., 2023) and it's default settings. Our SA algorithm was coded in C using Code::Blocks 17.12.

7.1 Constraint Programming Results

The constraint programming model demonstrates good performance for smaller instances. In Table 2, for problems with 10 jobs, the first three columns define the problem parameters: m represents the number of machines available, d indicates the percentage of jobs that conflicts with each other, and b is the upper bound for job processing times. The symbol $\overline{C_{\max}}$ represents the average optimal makespan value across 10 instances for each parameter combination, giving us an reliable measure of typical schedule length. Finally, the Avg. CPU column reports the average computation time in seconds required to find and prove optimality.

As shown in Table 2, the Constraint Programming model successfully finds proven optimal solutions for all 180 configurations within reasonable computation times. The results reveal clear patterns. The processing time bound significantly impacts the makepan, with longer jobs leading to substantially extended schedules. Conflict density has more moderate but still noticeable affects, as job incompatibilities limits parallel execution oportunities. Interestingly, having more machines generaly reduces schedule length, though the relationship is not always linear due to the complex interactions between setup times and machine assignments. All instances were solved to proven optimality within seconds, demonstrating the effectiveness of CP for smaller problem sizes.

Table 2: CP performance for problems with 10 jobs.

m	d	b	$\overline{C_{\max}}$	Avg. CPU (s)
2	10	10	41.6	2.40
2	10	50	167.0	4.17
2	30	10	44.6	1.96
2	30	50	203.0	6.13
2	50	10	43.4	1.92
2	50	50	174.0	2.60
3	10	10	35.9	1.10
3	10	50	124.0	1.45
3	30	10	37.0	1.09
3	30	50	157.0	1.93
3	50	10	35.6	1.09
3	50	50	151.0	1.56
5	10	10	38.3	1.12
5	10	50	136.0	1.50
5	30	10	37.2	1.14
5	30	50	149.0	2.07
5	50	10	34.0	1.13
5	50	50	157.0	1.93

For larger 15-job problems shown in Table 3, CP faces greater computational challenges that highlight the complexity of the problem.

Table 3: CP results for $n = 15$ jobs.

m	d	b	$\overline{C_{\max}}$	Avg. CPU (s)
2	10	10	90.1	1800.0
2	10	50	454.8	1800.0
2	30	10	98.4	1800.0
2	30	50	492.6	1800.0
2	50	10	76.2	1800.0
2	50	50	474.9	1800.0
3	10	10	58.3	185.8
3	10	50	290.9	908.9
3	30	10	72.5	546.9
3	30	50	427.8	1440.7
3	50	10	68.5	542.6
3	50	50	351.8	1090.7
5	10	10	52.5	4.1
5	10	50	229.0	192.4
5	30	10	51.7	5.9
5	30	50	268.8	549.1
5	50	10	52.8	4.4
5	50	50	353.3	1083.4

Table 3 maintains the same parameter structure, but reveals significantly different computational behavior under the extended 30-minute time limit. Although we tested 10 instances per parameter combination as in the $n = 10$ case, the solver frequently struggled to prove optimality within the allocated time. The CPU time column clearly indicates this challenge: when the value reaches 1800 seconds, it means that the solver hit the time limit without proving optimality for any of the 10 instances in that configuration.

With only 2 machines, the solver consistently hits the 30-minute timeout across all instances, though it always finds feasible solutions. The situation improves substantially with more machines - configurations with 5 machines are generally solved faster, with many instances proven optimal within seconds. This is likely because additional machines provide more parallelism and flexibility, allowing the CP solver to explore feasible schedules more efficiently and potentially find optimal solutions faster.

7.2 Simulated Annealing Performance

We evaluated Simulated Annealing against constraint programming results across 180 configurations for each problem size. Table 4 shows that SA achieves remarkable performance on 10-job instances, with an average optimality gap of only 0.5%.

Table 4: SA performance for problems with 10 jobs (10 instances per setting).

m	d	b	$\overline{Gap}$ (%)	Max Gap (%)
2	10	10	0.0	0.0
2	10	50	0.6	2.1
2	30	10	0.2	0.8
2	30	50	1.2	4.3
2	50	10	0.3	1.2
2	50	50	1.8	6.5
3	10	10	0.0	0.0
3	10	50	0.8	2.9
3	30	10	0.3	1.1
3	30	50	1.1	3.8
3	50	10	0.4	1.6
3	50	50	0.9	3.2
5	10	10	0.0	0.0
5	10	50	0.1	0.4
5	30	10	0.0	0.0
5	30	50	0.7	2.5
5	50	10	0.2	0.9
5	50	50	1.0	3.6
			0.5	**6.5**

Tables 4 and 5 retain the first three parameter columns used earlier and add two solution-quality metrics. The column ($\overline{Gap}$%) reports the average percentage gap between the SA and CP solutions in each set of tests, where the per-instance gap is calculated as

$$Gap(\%) = \frac{(SA_{\text{makespan}} - CP_{\text{makespan}})}{CP_{\text{makespan}}} \times 100\%.$$

The column *Max Gap* gives the worst (largest) gap observed for each configuration, highlighting the robustness of the algorithm. Note that gaps can be negative when SA finds better solutions than CP, particularly when CP hits time limits and the reported solution is sub-optimal.

Table 4 shows that SA performs very well on 10-job instances, with an average gap of only 0.5%. In general, SA frequently matches the optimal solutions found by CP and never exceeds a 6.5% gap in the most challenging cases.

For 15-job instances (Table 5), SA shows a more variable performance with both positive and negative gaps. Negative gaps arise mainly when the CP solver reaches its time limit and reports sub-optimal solutions. The average gap of -25.1% demonstrates the superiority of SA in these difficult cases, with the most significant improvements observed for configurations with $m = 2$ and $m = 3$ on problems too hard for the CP. The positive gaps observed for certain $m = 5$ configurations, where CP quickly proves optimality, suggest that SA's fixed 10-second time limit likely impacts its ability to match CP's performance in these easier cases.

Table 5: SA performance for problems with 15 jobs (10 instances per setting).

m	d	b	$\overline{Gap}$(%)	Max Gap (%)
2	10	10	-25.1	-47.6
2	10	50	-38.7	-57.6
2	30	10	-30.0	-50.0
2	30	50	-42.2	-60.5
2	50	10	-15.2	-39.2
2	50	50	-37.1	-56.3
3	10	10	-9.9	-55.3
3	10	50	-31.6	-57.8
3	30	10	-24.6	-53.4
3	30	50	-48.0	-60.2
3	50	10	-22.8	-53.3
3	50	50	-31.8	-58.9
5	10	10	0.2	9.3
5	10	50	-16.0	-39.8
5	30	10	0.2	12.8
5	30	50	-23.5	-52.9
5	50	10	1.5	15.0
5	50	50	-35.1	-56.6
Total			**-25.1**	**-60.5**

Based on our experimental analysis, several important conclusions can be drawn:

- The constraint programming method works very well for small problems and gives the best possible solution but becomes too slow for larger problems.
- The SA method works very well for small problems and often finds better solutions than CP for larger, more difficult problems where the exact method runs out of time.
- The 10-second time limit for SA sometimes prevents it from finding the best solution on easier problems where CP works quickly.
- The best method depends on the problem characteristics: problems with more machines are generally easier to solve, while problems with many job conflicts (higher d) and longer processing times (higher b) present the greatest challenge for both methods.

8 CONCLUSION

This paper addresses a new scheduling problem involving job assignments to machines, where setup operations and maintenance activities must be considered at the same time. We proposed and evaluated two distinct solution strategies. The exact approach (CP) gives optimal solutions for smaller instances (10 jobs) but can give only feasible solutions when the problem size increases. In contrast, the simulated annealing (SA) algorithm produces high-quality solutions within fixed time limit 10 seconds. For instances of the problem with 15 jobs, SA maintains an average deviation below 2.5% from optimal values. This makes the method particularly suitable for practical applications requiring rapid feasible solutions. For future research, it is important to focus on enhancing parameter calibration for simulated annealing, implementation of mixed-integer programming to establish solution quality benchmarks when optimality remains unproven, and finally, development of integrated approaches that leverage the complementary advantages of both exact and approximate methods.

ACKNOWLEDGMENTS

This research was funded by the European Health and Digital Executive Agency, Project: 101138517, Tec4MaaSEs, HORIZON-CL4-2023-TWIN-TRANSITION-01.

REFERENCES

Benmansour, R., Darvish, M., Todosijević, R., and Zufferey, N. (2024). Variable neighbourhood search for parallel machine scheduling with a single loading server: a truck-scheduling perspective. In *Supply Chain Forum: An International Journal*, volume 25, pages 308–320. Taylor & Francis.

Brucker, P., Dhaenens-Flipo, C., Knust, S., Kravchenko, S. A., and Werner, F. (2002). Complexity results for parallel machine problems with a single server. *Journal of Scheduling*, 5(6):429–457.

Chu, G., Stuckey, P. J., Schutt, A., Ehlers, T., Gange, G., and Francis, K. (2023). The chuffed solver. https://github.com/chuffed/chuffed. Accessed: 2025-12-14.

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

Elidrissi, A., Benbrahim, M., Benmansour, R., and Duvivier, D. (2019). Variable neighborhood search for identical parallel machine scheduling problem with a single server. In *International Conference on Variable Neighborhood Search*, pages 112–125. Springer.

Elidrissi, A., Benmansour, R., Hasani, K., and Werner, F. (2024). Minimizing the makespan on two parallel machines with a common server in charge of loading and unloading operations. *Computers & Operations Research*, 167:106638.

Gan, H.-S., Wirth, A., and Abdekhodaee, A. (2012). A branch-and-price algorithm for the general case of scheduling parallel machines with a single server. *Computers & Operations Research*, 39(9):2242–2247.

Hà, M. H., Ta, D. Q., and Nguyen, T. T. (2021). Exact algorithms for scheduling problems on parallel identical machines with conflict jobs. *arXiv preprint arXiv:2102.06043*.

Hall, N. G., Potts, C. N., and Sriskandarajah, C. (2000). Parallel machine scheduling with a common server. *Discrete Applied Mathematics*, 102(3):223–243.

Heinz, V., Novák, A., Vlk, M., and Hanzálek, Z. (2022). Constraint programming and constructive heuristics for parallel machine scheduling with sequence-dependent setups and common servers. *Computers & Industrial Engineering*, 172:108586.

Hong, H.-C. and Lin, B. M. (2018). Parallel dedicated machine scheduling with conflict graphs. *Computers & Industrial Engineering*, 124:316–321.

Kacem, I. (2007). Scheduling under unavailability constraints to minimize flow-time criteria. *Multiprocessor Scheduling*, page 47.

Kaparis, K., Georgiou, A. C., Lounis, S., Tsaples, G., Ioannis, M., Zois, G., Sifaleras, A., Watson, K., Casla, P., Turkay, M., and Ozcan, M. C. (2025). Designing a manufacturing as a service ecosystem through distributed value networks and structured volume-variety dynamics. *International Journal of Systems Science: Operations & Logistics*, 12(1):2597349.

Kim, M.-Y. and Lee, Y. H. (2012). 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.

Kirkpatrick, S., Gelatt Jr, C. D., and Vecchi, M. P. (1983). Optimization by simulated annealing. *science*, 220(4598):671–680.

Kowalczyk, D. and Leus, R. (2017). An exact algorithm for parallel machine scheduling with conflicts. *Journal of Scheduling*, 20(4):355–372.

Kravchenko, S. A. and Werner, F. (1997). Parallel machine scheduling problems with a single server. *Mathematical and Computer Modelling*, 26(12):1–11.

Lee, C.-Y. (1996). Machine scheduling with an availability constraint. *Journal of global optimization*, 9(3):395–416.

Ostrowski, J., Anjos, M. F., and Vannelli, A. (2010). *Symmetry in scheduling problems*. GERAD Montreal, QC, Canada.

Schmidt, G. (2000). Scheduling with limited machine availability. *European Journal of Operational Research*, 121(1):1–15.

Silva, J. M. P., Teixeira, E., and Subramanian, A. (2019). Exact and metaheuristic approaches for identical parallel machine scheduling with a common server and sequence-dependent setup times. *Journal of the Operational Research Society*, pages 1–15.

Torjai, L. and Kruzslicz, F. (2016). Mixed integer programming formulations for the biomass truck scheduling problem. *Central European Journal of Operations Research*, 24(3):731–745.

Van Hentenryck, P. and Michel, L. Minizinc: A free constraint modelling language. https://www.minizinc.org. Accessed: 2025-12-14.