High Performance Algorithms for the Unrelated Parallel Machines Scheduling Problem with a Common Server and Job-Sequence Dependent Setup Times

Nathalie Grangeon, Youssouf Hadhbi, Laurent Deroussi, Sylvie Norre

High Performance Algorithms for the Unrelated Parallel Machines Scheduling Problem with a Common Server and Job-Sequence Dependent Setup Times

Youssouf Hadhbi, Laurent Deroussi, Nathalie Grangeon, and Sylvie Norre

Université Clermont Auvergne, CNRS, Clermont Auvergne INP, Mines Saint-Étienne, LIMOS, 63000 Clermont-Ferrand, France
{youssouf.hadhbi, laurent.deroussi, nathalie.grangeon, sylvie.norre}@uca.fr

1 Introduction

This paper deals with a new variant of the *non-preemptive unrelated parallel machines scheduling problem with setup times* (UPMS-SDST). In this context, the setup processing is not operated automatically such that a common server is used as an extra shared resource to operate the setup processing between each two jobs assigned to the same machine. For this, we consider the so-called job-sequence dependent setup times which means that the setup times depend on only the sequence of jobs. This problem is known under the name of *unrelated parallel machines scheduling problem with a common server and job-sequence dependent setup times* (UPMS-CS-SDST) [27]. It can be formally defined as follows. We consider a set of unrelated parallel machines M. There does not exist a dependency between these machines such that each machine $i \in M$ has its proper characteristics as speed, configuration, energy consumption and quality of work [27]. Notice that the machines M are available along a set of T_{max} consecutive periods denoted by $T = \{1, 2, ..., T_{max} - 1, T_{max}\}$. Let N be a set of n jobs numbered from 1 to n such that each job k is characterized by

- a subset of qualified machines $M_k \subseteq M$ that are capable of processing the job k,
- a processing-time $p_i^k \in \mathbb{N}$ on each machine $i \in M$ such that $p_i^k = +\infty$ for each non qualified machine $i \in M \setminus M_k$.
- a priority factor $w_k \in \mathbb{R}^*$,
- a set of setup-times $s_k^j \in \mathbb{N}$ of job k after job $j \in N_0 \setminus \{k\}$ if the two jobs k and j are assigned to the same machine that is to say the job-sequence dependent setup times, where N_0 denotes the set of jobs in N with an additional dummy-job 0 (i.e., $N_0 = N \cup \{0\}$) such that the dummy-job 0 precedes each first job assigned to each machine.

As mentioned before, a common server is used to manage the setup operations between each pair of consecutive jobs (j, k) assigned to the same machine. Moreover, the common server can be unavailable at some periods in T. For this, we consider a binary parameter a_t which equals to 1 if the server is available at period t, and 0 if not. However, the jobs N are available in all periods of T. The problem consists in assigning each job k to one of its qualified machine in M_k while satisfying the following technological constraints

- each job k must be processed only one time by one of its qualified machines $i \in M_k$. Moreover, p_i^k consecutive periods are assigned to each job k if it is assigned to machine i (*processing-time*). As a consequence, each job k has one completion period of processing in T (*non-premption of processing*),
- s_k^j consecutive periods are assigned to each job k if it is processed immediately after a job j over the same machine (*setup-time*),
- the common server cannot start the setup processing for a job if it hasn't finished the setup for another job yet (*non-premption of setup*),
- each machine can handle at most one job at each period such that two jobs cannot be processed at the same period on the same machine (*non-overlapping of processing*),
- the setup operation cannot be ensured at period t if the common server is not available at period t. Moreover, the common server can ensure the setup operation of at most one job at period t (*non-overlapping of setup*),

– the common server interrupts the setup processing for a job when it becomes unavailable. However, it can resume the setup processing of this job when it becomes available (*server availability*).

The objective is to minimize the total weighted completion time of processing for the different jobs.
From a practical point of view, the UPMS-CS-SDST problem arises when planning production and scheduling for some industrial flexible manufacturing systems. It appears in some applications related to the production of some mechanical parts of automobiles, hydraulic and electrical sectors [27] in the context of Industry 4.0. In this context, optimization of these real modern systems is a real challenge such that effective scheduling is a key issue to well manage the different resources and improve productivity of manufacturing systems.
The UPMS-CS-SDST problem is NP-hard in the strong sense [27]. The main contributions of this paper is as follows.
We first formulate the problem as a mixed integer linear program (MILP), and further devise an exact algorithm based on a branch-and-cut (B&C) algorithm to solve the problem. Due to the complexity of the problem, we propose a metaheuristic based on an iterated local search (ILS) algorithm [21] for solving the problem. Using this, we provide several matheuristics based on a two stage algorithm: ILS first and MILP last. We also carry out a comparative study between these methods and show the effectiveness of our approach using small-sized instances and large-sized instances.
The rest of this paper is organized as follows. In Section 2, we present some related works that have been well studied in the literature. In Section 3, we present a MILP for the problem. We introduce the ILS in Section 4. Matheuristics are presented in Section 5. We then present an extensive computational study using different classes of instances. Finally, we summarize our results and future outlook in Section 7.

2 Related Works

In the most general statement, the classical *unrelated parallel machines scheduling problem* (UPMS) has been well studied in the literature. The setup times are not considered in this case. It has been shown to be NP-hard [19]. Several exact algorithms have been proposed to solve the problem [1][8][23][22][33]. Despite the NP-hardness of the problem, heuristics and metaheuristics have also been required to solve the problem [14][33]. On the other hand, some approximate algorithms with good performance guarantee have been developed for solving the problem [16][19][24][32].
Notice that for some works, the setup-time has been considered as a part of processing-time. Here, we focus on the works that considered the processing and the setup operation separately. This is related to the UPMS-SDST problem. For this, we consider sequence-dependent setup times such that a setup operation is needed after each completion of processing of a job and before each starting of processing of another job assigned to the same machine. In this context, each machine is regulated to process the next job. This needs a setup time which depends only on the pair of consecutive jobs who share the same machine. This has first been studied by Allahverdi et al. [2]. Several exact algorithms have been proposed to solve the UPMS-SDST problem. They are based on branch-and-bound algorithm [29], branch-and-check algorithm [13], branch-and-price algorithm [25] and Bender decomposition [30]. However, these approaches have been shown to be less efficient when using large-sized instances of the same problem.
For this, heuristics [5][20][26] and metaheuristics have been used to solve the UPMS-SDST problem. Rabadi et al. [26] presented a greedy randomized adaptive search procedure to solve the problem. A simulated annealing has been used by Radhakrishnan and Ventura to solve the same problem [28]. Helal et al. [17] presented also a tabu search algorithm for the problem. On the other hand, some population-based metaheuristics have been introduced to solve the problem. Vallada and Ruiz [31] proposed a genetic algorithm for the problem. An ant colony optimization algorithm has been developed by Arnaout et al. [3] for solving the problem. Recently, Arnaout et al. [4] developed a worm optimization algorithm and compared it with some known metaheuristics for the same problem.
Hybrid methods have also been developed to solve the UPMS-SDST problem. Fang et al. [12] developed an hybridization of adaptive large neighborhood search algorithm with a tabu search

algorithm. Zeidi and Hosseini [34] proposed a two-stage algorithm which combines a genetic algorithm with a simulated annealing algorithm. Behnamian et al. [6] presented an hybridization of ant colony optimization, simulated annealing and variable neighborhood search.
Notice that the setup-processing is operated automatically in these previous works. This means that they did not consider the existence of a single or multiple servers to manage the setup process. Moreover, this resource is shared by all jobs N and machines M.
There exist a few works that have taken into account these additional resources (servers) and setup constraints such that the UPMS-CS-SDST problem is more less studied than the UPMS-SDST problem. A new integer linear programming formulation has been proposed by Bektur et al. [7] for solving the UPMS-CS-SDST problem without taking into account the unavailability of the common server in some periods of T. Moreover, the same authors developed some metaheuristics based on a simulated annealing and a tabu search algorithm for solving the problem. Elidrissi et al. [11] introduced a mixed integer programming formulation for a similar problem with two commun servers. They also developed two greedy heuristics and a general variable neighborhood search algorithm. The results showed that this latter outperformed the two other approaches.
To the best of our knowledge, the work done by Raboudi et al. [27] represents the initial study of the UPMS-CS-SDST problem taking into account the technological constraints and problem characteristics that have been considered in our study. Their mixed integer linear programming formulation has shown some limits such that it has been shown to be not able to solve small instances to optimality with a number of jobs up to 7 and 6 machines.

3 Mixed Integer Linear Programming Formulation

In what follows, we present a mixed integer linear programming formulation [15] for solving the UPMS-CS-SDST problem based on the following variables

- for each job $k \in N$ and machine $i \in M$, let u_i^k be a binary variables which equals to 1 if job k is assigned to machine i, and 0 if not,
- for each machine $i \in M$, job $j \in N_0$ and job $k \in N$, we denote by $x_{j,k}^i$ a binary variables which is related to the sequence-dependent setup times such that it takes 1 if jobs j is processed immediately before job k on machine i, and 0 if not,
- for each job $j \in N$ and period $t \in T$, variable y_t^k equals to 1 if the setup operation of job k is performed at period t, and 0 if not,
- for each two distinct jobs $j, k \in N$, we consider the variables q_k^j which takes 1 if the job k is processed after job j even if they are not assigned to the same machine, and 0 if not. This means that the starting period of the setup operation for job k is performed after the ending period of the setup operation of job j,
- we denote by $b^k \in \mathbb{R}^+$ (resp. $e^k \in \mathbb{R}^+$) the starting period (resp. the ending period) of the setup operation for job k,
- the completion period of processing for each job $k \in N$ is denoted by $c_k \in \mathbb{R}^+$.

The UPMS-CS-SDST problem is then equivalent to the following MILP

$$\min \sum_{k \in N} w_k c_k, \tag{1}$$

subject to

$$\sum_{i \in M_k} u_i^k = 1, \forall k \in N, \tag{2}$$

$$\sum_{i \in M \setminus M_k} u_i^k = 0, \forall k \in N, \tag{3}$$

$$\sum_{j \in N_0 \setminus \{k\}} x_{j,k}^i = u_i^k, \forall k \in N \text{ and } i \in M \tag{4}$$

$$\sum_{j \in N \setminus \{k\}} x_{k,j}^i \leq u_i^k, \forall k \in N \text{ and } i \in M, \tag{5}$$

$$\sum_{k \in N} x_{0,k}^i \leq 1, \forall i \in M, \tag{6}$$

$$\sum_{k \in N} y_t^k \leq a_t, \forall t \in T, \tag{7}$$

$$\sum_{t \in T} y_t^k = \sum_{j \in N_0 \setminus \{k\}} \sum_{i \in M} s_k^j x_{j,k}^i, \forall k \in N, \tag{8}$$

$$b^k \leq t y_t^k + B(1 - y_t^k), \forall k \in N \text{ and } t \in T, \tag{9}$$

$$t y_t^k - B \sum_{t'=1}^{t-1} y_{t'}^k \leq b_k, \forall k \in N \text{ and } t \in T, \tag{10}$$

$$\sum_{j \in N \setminus \{k\}} \sum_{i \in M} p_j^i x_{j,k}^i \leq b^k, \forall k \in N, \tag{11}$$

$$(t+1) y_t^k \leq e^k, \forall k \in N \text{ and } t \in \{1, ..., T_{max} - 1\}, \tag{12}$$

$$c_j + B(\sum_{i \in M} x_{j,k}^i - 1) \leq b^k, \forall k \in N \text{ and } j \in N \setminus \{k\}, \tag{13}$$

$$e^k - b^k \geq \sum_{j \in N_0 \setminus \{k\}} \sum_{i \in M} s_k^j x_{j,k}^i, \forall k \in N, \tag{14}$$

$$e^k \leq b^j + B q_k^j, \forall k \in N \text{ and } j \in N \setminus \{k\}, \tag{15}$$

$$e^j \leq b^k + B(1 - q_k^j), \forall k \in N \text{ and } j \in N \setminus \{k\}, \tag{16}$$

$$c_k = e^k + \sum_{i \in M_k} p_k^i u_i^k, \forall k \in N, \tag{17}$$

$$c_k \leq T_{max}, \forall k \in N, \tag{18}$$

$$q_k^j \leq 1, \forall j \in K \text{ and } k \in N \setminus \{j\}, \tag{19}$$

$$u_i^k, x_{j,k}^i, y_t^k, q_k^j, b^k, e^k, c_k \geq 0, \tag{20}$$

$$u_i^k, x_{j,k}^i, y_t^k, q_k^j \in \{0, 1\}, \tag{21}$$

where $T' = \{1, ..., T_{max} - 1\}$, and B a large integer number (eg., $B = T_{max}$ is feasible).
The objective function (1) consists in minimizing the total weighted completion time of processing for the different jobs in N. Equations (2) express the fact that each job $k \in N$ is handled by only one machine of its qualified machine $i \in M_k$. Equations (3) show that each job $k \in N$ cannot be assigned to a non qualified machine $i \in M \setminus M_k$. Equations (4) ensure that a job k is preceded by one job $j \in N_0 \setminus \{k\}$ on a machine $i \in M$ if and only if job k is assigned to machine i. Moreover, constraints (5) shows that a job k can be the predecessor of at most one job $j \in N \setminus \{k\}$ on machine i if and only if it is assigned to machine i. The dummy-job can precede at most one job on each machine $i \in M$ as shown by constraints (6). The common server can handle at most one job at each period $t \in T$ if and only it is available at period t as noticed in constraints (7). However, variables y_t^k are forced to be equal to 0 for each $k \in N$ when the common server is not available at period t. Equations (8) show that the number of periods in which the setup is performed for job k must be equal to its setup-time. Constraints (9) and (10) ensure that a period t can be a starting period of the setup operation of jobs k if the setup of job k is performed at period t and there does not exist a period $t' \in \{1, ..., t-1\}$ in which the setup of job k is performed. Constraints (11) ensure that the starting period of setup for a job k is forced to be greater than the processing time of a job j which is processed immediately before job k on a shared qualified machine in $M_k \cap M_j$. In a similar way, we ensure in constraints (12) that the setup operation of each job $k \in N$ is accomplished after the last period t in which the setup is performed for job k. Constraints (13) ensure the non-overlapping of the processing operation with the setup operation for two distinct jobs that are processed one after the other and immediately. The gap between the ending period and the starting period of setup for job j is greater than its setup time as shown in constraints (14). Constraints (15) and (16) ensure the non-preemption of setup constraints. The completion period for each job $k \in K$ is computed as shown in constraints (17). Constraints (18) impose that the completion period should be smaller than T_{max}. Inequalities (19) and (20) are the trivial inequalities, and constraints (21) are the integrality constraints.
Using this formulation, we devise a branch-and-cut algorithm to solve the UPMS-CS-SDST problem [15] by combining a *branch-and-bound* algorithm with a *cutting plane* algorithm.

4 Iterated Local Search

In this section, we give a detailed description of the iterated local search algorithm [21] used to solve the UPMS-CS-SDST problem. We discuss the importance of its different procedures. This algorithm aims at building iteratively a sequence of solutions generated by an improvement heuristic based on the so-called local search (LS) algorithm. The ILS is based on the following procedures

- *Representation of a solution:* in this work, a solution S of the UPMS-CS-SDST problem is considered as a sequence of jobs for the setup-processing. It can be represented as vector $\{[1], [2], ..., [n]\}$ where $[k]$ denotes the job that is placed in position k in S.
- *Evaluation procedure:* for this, we first consider a solution S of the problem. This solution is then evaluated by using a greedy algorithm as follows. At each iteration, we select the first job k from the solution S that is not yet assigned to a machine $i \in M_k$. After this, for each qualified machine $i \in M_k$ of job k, we compute the starting period of setup, the ending period of setup, and the completion time of the processing for job k while satisfying all constraints of the problem with the set of jobs that are already proceeded (i.e., the set of jobs that precede k in the solution S). Then, the selected job k is assigned to the qualified machine $i \in M_k$ that offers the minimum completion time C_k. The algorithm stops when all jobs are assigned, or when the completion time C_k of the current job k exceeds T_{max} which means that solution S is infeasible. The output of this algorithm is given by a quadruple $(f(S), G, C, R)$ where
  - $f(S)$ denotes the total weighted completion time of processing for the different jobs in S,
  - G is a matrix of $m * T_{max}$ dimension such that each element G_t^i denotes the index of the job assigned at period t of the machine i, and equals to 0 if no job is assigned to machine i at period t. This can be used to draw a Gantt diagram.
  - C is a vector of size n which presents the completion time of processing for the set of jobs such that each C_k represents the completion time of job k as mentioned before.
  - R is a vector of size T_{max} such that each element R_t stores the index of the job for which the setup processing is done at period t by the common server, and 0 if no setup is done at period t
- *Initial solution:* an initial solution for the UPMS-CS-SDST problem can be seen as a starting point in a search area of the solutions. For this, one can randomly generate an initial solution S_0 for the problem. We then use an evaluation procedure described above to evaluate this solution and further show if it is feasible or not for the problem. Generally, this starting solution S_0 does not give a good quality solution. This step must not be neglected such that starting with a good solution improves the quality of the algorithm and allows achieving high quality solutions as fast as possible and especially the computation time is very short.
- *Neighborhood procedure:* this aims at exploring the neighborhood area of a solution S in particular and the search area of all solutions of the problem in general. This procedure generates a new solution S' for the problem, called neighbor of S (denoted by $S' = Neighbor(S)$) such that some positions of certain jobs of S are randomly changed in S'. This new solution is then evaluated to be compared with solution S. This procedure should be executed many times to explore the neighborhood space of a solution and extend the search space. For this, we distinguish several neighborhood strategies that can be used to explore the solution space
  - *Exchange:* we need to select randomly two jobs j and k and change their positions in the sequence J.
  - *Insertion:* it consists in moving randomly a job from a position a and inserting it in another position b in sequence J.
  - *Inversion:* we first select randomly two positions a and b. Then, we reverse the sub-sequence of jobs situated between a and b.
- *Acceptance Criterion:* choosing the ideal acceptance criterion is very important such that it aims at determining the rules for the acceptance of updating the current best solution and replacing it by an iteration solution. For this, we use the so-called "Better" criterion proposed by Lourenço et al. [21] such that a solution S' can replace a solution S at each iteration of the algorithm if and only if the quality of the new solution S' is better or equal to the quality of solution S.
- *Stopping criterion:* in our study, the algorithm terminates when we exceed a limited number of iterations or a maximum CPU time.

– *Improvement procedure:* this is based on a local search algorithm. Consider a solution $S^{'}$, the local search algorithm aims at finding a nearby solution $S^{'*}$ with better quality than $S^{'}$. This algorithm needs as input the maximum number of iterations without improvement, neighborhood method, a maximum CPU time, and an initial solution $S^{'}$. We then explore the neighborhood space of solution $S^{'}$ given at each iteration of the ILS which is considered as the initial best solution of the LS (denoted by $S^{*\prime}$). At each iteration of the local search algorithm, we generate a new solution $S^{"}$ that can be seen as a neighbor of the current best solution $S^{*\prime}$ and then evaluate it using the greedy-algorithm. This will then be compared with the current best solution $S^{*\prime}$. We update the current best solution if it satisfies the acceptance criterion. The algorithm stops when the stopping criterion is verified. Algorithm 1 summarizes the different steps of the local search algorithm.

Algorithm 1 Local Search Algorithm

Require: a maximum number of iterations without improvement denoted by $Max_{it} \geq 0$, a maximum CPU time denoted by $Max_{Cpu} \geq 0$, perturbation method denoted by $Neighbor$, evaluation procedure, initial solution denoted by S'.
$i \leftarrow 0$ and $S^{*\prime} \leftarrow S'$
while $i \leq Max_{it}$ and Max_{Cpu} is not exceeded **do**
  $S" \leftarrow Neighbor(S^{*\prime})$ and $i \leftarrow i + 1$
  if $f(S") \leq f(S^{*\prime})$ **then**
    if $f(S") < f(S^{*\prime})$ **then**
      $i \leftarrow 0$
    end if
    $S^{*\prime} \leftarrow S"$
  end if
end while
return $S^{*\prime}$

To summarize, the ILS can be seen as an iterative improvement technique. Here, the goal is to find the best sequencing of jobs which gives a high quality solution for the problem. For this, we first use a construction method to generate an initial feasible solution S_0 for the problem that will be improved by the LS to provide an initial best solution S^* for the problem. After this, and at each iteration, we apply multiple perturbations on the current solution S^* using a neighborhood method. As a result, a new solution S' is found for the problem. We then use the local search algorithm to explore the neighborhood space of solution S' and return a local optimum $S^{*\prime}$ of the LS. The resulting solution will then be compared with the current best solution of the ILS and become the new best solution if it satisfies the acceptance criterion. The algorithm is stopped if one of the stopping criteria is verified.
All these steps are summarized in Algorithm 2.

Algorithm 2 Iterated Local Search Algorithm

Require: a maximum number of iterations for the ILS denoted by $Max_{it}^{ILS} \geq 0$, a maximum CPU time for the ILS denoted by $Max_{Cpu}^{ILS} \geq 0$, number of perturbations for the ILS denoted by b, perturbation method for the ILS denoted by $Neighbor_{ILS}$, LS's parameters, evaluation procedure, initial solution denoted by S_0.
$S^* \leftarrow LS(Max_{it}^{LS}, Max_{Cpu}^{LS}, Neighbor_{LS}, f, S_0)$ and $i \leftarrow 0$
while $i \leq Max_{it}^{ILS}$ and Max_{Cpu}^{ILS} is not exceeded **do**
  $S' \leftarrow S^*$, $a = 0$ and $i \leftarrow i + 1$
  while $a \leq b$ **do**
    $S' \leftarrow Neighbor(S')$ and $a \leftarrow a + 1$ *//Perturbation(S')*
  end while
  $S^{*\prime} \leftarrow LS(Max_{it}^{LS}, Max_{Cpu}^{LS}, Neighbor_{LS}, f, S')$
  if $f(S^{*\prime}) \leq f(S^*)$ **then** *//AcceptanceCriterion($S^*, S^{*\prime}$)*
    $S^* \leftarrow S^{*\prime}$
  end if
end while
return S^*

5 Matheuristics

In what follows, we introduce three matheuristics for solving the UPMS-CS-SDST problem. They can be considered as a two stage algorithm combined an iterated local search algorithm with a modified version of our MILP formulation [15]. Notice that these approaches provide approximate solutions for the problem without guarantee of optimality.
Throughout the following sections, our MILP formulation already presented in Section 3, will be considered as the basic MILP formulation for the problem.

5.1 Matheuristic I: Machine-Job-Sequencing Fixing

For this, we first use an ILS to solve the problem. Consider the resulting solution S. We then use an updated formulation of the basic MILP with some additional constraints. This matheuristic aims at producing an optimal solution for the resulting sequencing of jobs provided by S while respecting some additional precedence constraints required by S. This means that each job $j \in N$ should be preceded by each job k having a smallest index in S compared with the index of j in S if and only if the two jobs j and k are assigned to the same machine. Otherwise, the precedence constraint between these two jobs is not considered. For this, we consider a matrix P of $n * n$ dimension such that $P_{j,k} = 1$ if job j is preceded by job k in S, and 0 if not. After this, we use a MILP formulation to solve the problem such that we keep the same variables and constraints used in the basic MILP. Moreover, we add the following precedence constraints

$$c_k + B(u_i^j + u_i^k - 2) \leq b_j, \forall k \in N, \forall j \in N \setminus \{k\}, \forall i \in M \text{ with } P_{j,k} = 1. \tag{22}$$

Inequalities (22) ensure that if job j and k are assigned to the same machine, and $P_{j,k} = 1$ then the starting period of setup for job j must be greater than the completion period of job k.
Notice that this modified MILP can also be used as an exact method to solve a new variant of the UPMS-CS-SDST problem that can be called *unrelated parallel machines scheduling problem with a common server, job-sequence dependent setup times and precedence constraints*.

5.2 Matheuristic II: Machine-Job-Assignment Fixing

In this case, we aim at identifying the optimal solution of the problem with pre-assignment of machines. For this, we make the set of qualified machines $M_k = \{i\}$ if job k is assigned to machine i in solution S of ILS. Based on this, we introduce a new MILP to solve the problem taking into account the pre-assignment of machines as additional constraints. It's based on the same variable of the basic MILP without taking into account the variables u_i^k given that the jobs are already assigned to machines. We also modify the definition of variables $x_{j,k}^i$ such that we consider a new variable $x_{j,k}$ which equals to 1 if job k is processed immediately after job j and they should be already assigned to the same machine, and 0 if not. As a consequence, the number of variables is decreased and becomes less compared with the basic MILP. Moreover, the constraints that are related to the machine assignment should be deleted, and constraints (4), (5), (8), (11), (13), (14) and (17) should be modified as follows

$$\sum_{j \in N_k \cup \{0\}} x_{j,k} = 1, \forall k \in N, \tag{23}$$

$$\sum_{j \in N_k} x_{k,j} \leq 1, \forall k \in N, \tag{24}$$

$$\sum_{t \in T} y_t^k = \sum_{j \in N_k \cup \{0\}} s_k^j x_{j,k}, \forall k \in N, \tag{25}$$

$$\sum_{j \in N_k} p_j^i x_{j,k} \leq b^k, \forall k \in N \text{ with } \{i\} = M_k, \tag{26}$$

$$c_j + B(x_{j,k} - 1) \leq b^k, \forall k \in N, \forall j \in N_k, \tag{27}$$

$$e^k - b^k \geq \sum_{j \in N_k \cup \{0\}} s_k^j x_{j,k}, \forall k \in N, \tag{28}$$

$$c_k = e^k + p_k^i, \forall k \in N \text{ with } \{i\} = M_k, \tag{29}$$

where N_k denotes the set of jobs that are assigned to the same machine with job k in solution S. As can be noticed, the number of variables and constraints is largely decreased due to these modifications compared with the basic MILP.

5.3 Matheuristic III: Setup-Job-Sequencing Fixing

The third matheuristic consists in determining the optimal solution for the problem with some additional constraints such that the predecessor of each job is known in advance using the solution S of the ILS. As a consequence, the setup time of each job is known in advance. For this, we denote by j_k the index of job that has been processed immediately before job k and assigned to the same machine in solution S. Using this, we introduce another MILP based on the original MILP with some modifications. We keep the same variables of the original MILP without considering the setup variables $x^i_{j,k}$. Some constraints should be removed from the model (e.g., constraints (4), (5), (6)), and other ones as (8), (11), (13), (14), should be modified as follows

$$\sum_{t\in T} y^k_t = s^{j_k}_k, \forall k \in N, \tag{30}$$

$$\sum_{i\in M} p^i_{j_k} \leq b^k, \forall k \in N, \tag{31}$$

$$c_{j_k} \leq b^k, \forall k \in N, \tag{32}$$

$$e^k - b^k \geq s^{j_k}_k, \forall k \in N. \tag{33}$$

Moreover, each job k and its predecessor j_k should be assigned to the same machine. For this, we add the following constraints to the new MILP formulation

$$u^{j_k}_i = u^k_i, \forall k \in N, \forall i \in M. \tag{34}$$

6 Computational Study

The different approaches described above have been implemented in Java and run on high performance computing servers with 64 GB as a memory limit. For each MILP, we devise a Branch-and-Cut algorithm to solve the problem. We have also implemented the MILP of Raboudi et al. [27]. For this, we use CPLEX [10] to solve the MILP formulations provided in previous sections. We also use its proper cuts to obtain tighter bounds for the linear relaxation and boost the performance of the Branch-and-Cut algorithm. We consider 900 sec as a CPU time limit for the ILS and 3600 sec for the B&C algorithm. The maximum number of iterations for the ILS is limited to 200000 such that the number of perturbations of the ILS equals to 4 at each iteration. The common server is available for 8 contiguous periods and then unavailable for 8 contiguous periods in T. All the approaches have been tested using three families of instances described as follows.

Instances / Characteristics	$\|M\|$	$\|N\|$	w_k	p^i_k	s^j_k	$\|M_k\|$	T_{max}
Instances I [27]	{2,3,4}	{4,5,6,7, 8, 10, 20, 24}	[1; 5]	[10; 350]	[1; 10]	≥ 1	[200; 1000]
Instances II	{2,3,4}	{4,5,6,7, 8, 10, 20, 24}	[1; 5]	[10; 350]	[1; 10]	≥ 2	[200; 1000]
Instances III	{2,5,10, 15, 20}	{5,10, 20, 30, 40}	[1; 6]	[1; 16]	[1; 6]	≥ 1	[300; 4000]

Table 1. Instances characteristics.

Concerning the ILS, for each instance and each neighborhood procedure, we compute the average value of the objective function (denoted by *avg*) and the objective function for the best solution (denoted by *min*) of 10 independent ILS solutions starting from the same initial solution that has been generated randomly.
On the other hand, for the different matheuristics, and for each instance, we use the best solution (*min*) found by 10 independent ILS solutions that will provide some additional constraints for the second stage of matheuristic as already described in Section 5.
The main objective of this study is to show the effectiveness of our approaches using different instances (30 instances of type I, 32 instances of type II and 131 instances of type III). For this, we address a comparison study between the different approaches in tables 2, 3, 4 and 5.
We consider the following indicators:

- % opt (avg): percentage of instances that are solved to optimality at each replication by the chosen algorithm (10 replications for the ILS and one replication when using a Branch-and-Cut algorithm for each instance).
- % opt (min): percentage of instances that are solved to optimality in at least one replication by the chosen algorithm.
- % best sol (avg): percentage of instances for which the chosen algorithm provides the best *avg*.
- % best sol (min): percentage of instances for which the chosen algorithm provides the best *min*.
- % low standard deviation (SD): percentage of instances for which the chosen algorithm has the best SD.
- % best sol the ILS is improved or equal: percentage of instances for which the chosen algorithm found a solution value which is equal or strictly better than the ILS's solution value.
- % best sol the ILS is improved: percentage of instances for which the chosen algorithm found a solution value which is strictly better than the ILS's solution value.

Notice that the results of our B&C algorithm (using our MILP) will be taken as a benchmark in our computational study given that it has been shown to be the best model in our previous study [15] compared with the MILP of Raboudi et al. [27], and other ones that we already proposed in previous studies.
First, results show that for the two first classes of instances I and II respectively, the B&C algorithm is able to solve to optimality 70% and 53,13% of instances respectively. Moreover, the B&C is shown to be able to beat the existing one MILP of Raboudi et al. [27] which suffers more from a tractability point of view even for small-sized instances. The latter solves to optimality 10%, 3,13% and 2,29% of instances of type I, II and III respectively. However, the B&C algorithm becomes less performant for the large-sized instances (Instances III) such that 6,11% of these instances are solved to optimality. As a consequence, the B&C becomes more less efficient when using large-sized instances and even for the medium ones. For this reason, and as mentioned before, we use the ILS and some matheuristics to solve the UPMS-CS-SDST problem.
We first show in Table 2 the effectiveness of the ILS using different neighborhood procedures denoted respectively by ILS_Exc , ILS_Inv and ILS_Ins when using respectively exchange, inversion and insertion as neighborhood procedure while considering the instances that are solved to optimality by the B&C. For this, we consider the two indicators % opt (avg) and % opt (min) to compare between these algorithms using the instances that are solved to optimality by the B&C.

		% opt (avg)	% opt (min)
Instances_I	**Raboudi et al.**	14,29	14,29
	ILS_Exc	90,48	90,48
	ILS_Inv	90,48	90,48
	ILS_Ins	90,48	90,48
	Math_MJSF	**95,24**	**95,24**
	Math_MJAF	85,71	85,71
	Math_SJSF	76,19	76,19
Instances_II	**Raboudi et al.**	5,88	5,88
	ILS_Exc	70,59	70,59
	ILS_Inv	70,59	70,59
	ILS_Ins	70,59	70,59
	Math_MJSF	**88,24**	**88,24**
	Math_MJAF	82,35	82,35
	Math_SJSF	70,59	70,59
Instances_III	**Raboudi et al.**	37,50	37,50
	ILS_Exc	75,00	75,00
	ILS_Inv	75,00	75,00
	ILS_Ins	75,00	75,00
	Math_MJSF	62,50	62,50
	Math_MJAF	**87,50**	**87,50**
	Math_SJSF	62,50	62,50

Table 2. Comparison between different algorithms based on the instances that are solved to optimality by the B&C.

The ILS has been shown to be very performant for the different instances such that it is able to solve several instances to optimality and even for the instances that are not solved to optimality by the MILP of Raboudi et al. [27]. The ILS also allows solving several instances that are not solved

by the B&C but without guarantee of optimality.
Moreover, we show in Table 3 the efficiency of each algorithm based on two indicators: % best sol (avg) and % best sol (min) while using all instances.

		% best sol (avg)	% best sol (min)
Instances_I	**Raboudi et al.**	46,67	46,67
	B&C	76,67	76,67
	ILS_Exc	**83,33**	**86,67**
	ILS_Inv	80,00	80,00
	ILS_Ins	**83,33**	**86,67**
	Math_MJSF	76,67	76,67
	Math_MJAF	73,33	73,33
	Math_SJSF	70,00	70,00
Instances_II	**Raboudi et al.**	40,63	40,63
	B&C	75,00	75,00
	ILS_Exc	**84,38**	**78,13**
	ILS_Inv	71,88	68,75
	ILS_Ins	75,00	**78,13**
	Math_MJSF	78,13	78,13
	Math_MJAF	75,00	75,00
	Math_SJSF	71,88	71,88
Instances_III	**Raboudi et al.**	7,63	7,63
	B&C	9,16	9,16
	ILS_Exc	**61,07**	64,12
	ILS_Inv	18,32	32,82
	ILS_Ins	59,54	**81,00**
	Math_MJSF	16,03	16,03
	Math_MJAF	9,92	9,92
	Math_SJSF	12,98	12,98

Table 3. Comparison between different algorithms based on all instances.

We notice in Table 3 that the ILS improves the quality of certain solutions proposed by the B&C and is able to find better solutions (based on *average* and *min*) than those found by the B&C or the MILP of Raboudi et al. [27]. These results are still stable for the different neighborhood procedures such that the exchange neighborhood procedure is shown to be advantageous in some instances compared with the other ones and also having the low SD (see Table 4) for the different instances of type I, II and III. Moreover, and using large-sized instances of type III, the ILS with its different neighborhood procedures, is able to solve some instances to optimality. However, for the other ones, the ILS cannot guarantee the optimality. For these same instances, the ILS gives several high quality solutions compared with the B&C.

	% low SD by ILS_Exc	% low SD by ILS_Inv	% low SD by ILS_Ins
Instances_I	**86,67**	83,33	**86,67**
Instances_II	**100,00**	87,5	90,63
Instances_III	**61,83**	27,48	59,54

Table 4. Comparison between different neighborhood procedures using standard deviation indicator.

Next, we combine the ILS with a B&C algorithm to devise a matheuristic for solving the UPMS-CS-SDST problem as already described in Section 5. For this, we assess the performance of three matheuristics denoted respectively by Math_MJSF, Math_MJAF and Math_SJSF. Table 2 shows that that they are also able to solve several instances to optimality and showed to be better than the ILS while using Math_MJSF for instances of type I and II. This latter solves some instances to optimality that are not solved to optimality by the ILS and the MILP of Raboudi et al. [27]. Table 3 also shows that the Math_MJSF is advantageous in some instances compared with the other approaches. However, when the optimality is not guaranteed or when using large-sized instances, the ILS proposes better solutions than the different matheuristics due to the usage of the MILP and the B&C in the second stage such that each MILP becomes more less tractable when using large-sized instances.
On the other hand, we aim to evaluate the impact of using the best solution of the ILS to add some additional constraints to the MILP in order to devise the different matheuristic, and also evaluate the impact of using this solution as a warm-start for these matheuristics. For this, Table 5 shows initially that the different matheuristics are able to produce several solutions with quality better or equal than those provided by the ILS (see column 1 in Table 5), and strictly better than those

of ILS for other ones (see column 2 in Table 5). Moreover, the warm-start technique is shown to be very efficient when using large-sized instances such that this technique is capable of boosting these matheuristics, and further allows obtaining strictly better solutions for more instances compared with when it's not used.

		Without Warm-Start		With Warm-Start
		% best sol the ILS is improved or equal	% best sol of the ILS is improved	% best sol of the ILS is improved
Instances_I	**Math_MJSF**	86,67	10,00	10,00
	Math_MJAF	83,33	10,00	10,00
	Math_SJSF	83,33	13,33	13,33
Instances_II	**Math_MJSF**	75,00	18,75	18,75
	Math_MJAF	78,13	31,25	31,25
	Math_SJSF	78,13	31,25	34,38
Instances_III	**Math_MJSF**	17,56	6,11	28,24
	Math_MJAF	12,21	4,58	29,01
	Math_SJSF	16,03	4,58	32,82

Table 5. Effectiveness of Matheuristics.

As a consequence, all these previous results prove the high quality performance of our approaches for solving the UPMS-CS-SDST problem.

7 Conclusion

In this paper, we have addressed the non-preemptive unrelated parallel machines scheduling problem with a common server and job-sequence dependent setup times. First, we have presented some related works considering the setup processing. We have proposed a mixed integer program to formulate the problem, and have devised an exact algorithm based on a branch-and-cut algorithm for solving the problem. Despite the NP-hardness of the problem, we have developed a metaheuristic based on an iterated local search algorithm to solve the problem. Using these results, we have presented different matheuristics that can be seen as post-optimization algorithms. The results have shown the effectiveness of these approaches and the advantages of certain ones.
Finally, it would be interesting to further develop some adaptive and hybrid metaheuristics for solving the problem. We also plan to study new realistic or real variants of the problem while considering some additional technological constraints (machine availability, delay for jobs) and ressources (multiple servers).

Acknowledgment

This work was supported by the French Government, France-Relance Project in collaboration with InoProd (https://www.inoprod.com/).

References

1. E. Åblad, A. Strömberg, and D. Spensieri, "Exact makespan minimization of unrelated parallel machines". Open Journal of Mathematical Optimization, 2, 2021, pp. 1-15.
2. A. Allahverdi, J. N. D Gupta, and T. Aldowaisan, "A review of scheduling research involving setup considerations". OMEGA International Journal of Management Science, 27, 1999, pp. 219–239.
3. J.P. Arnaout, G. Rabadi, and R. Musa, "A two-stage Ant Colony optimization algorithm to minimize the makespan on unrelated parallel machines—part II: enhancements and experimentations". Journal of Intelligent Manufacturing, 25, 2014, pp. 43–53.
4. J.P. Arnaout, "A worm optimization algorithm to minimize the makespan on unrelated parallel machines with sequence-dependent setup times". Annals of Operations Research, 285, 2020, pp. 273-293.
5. A. Al-Salem, "Scheduling to minimize makespan on unrelated parallel machines with sequence dependent setup times". Engineering Journal of University of Qatar, 17, 2004, pp. 177-187.
6. J. Behnamian,M. Zandieh, and S. M. T. Fatemi Ghomi, "Parallel-machine Scheduling Problems with Sequence-dependent Setup times Using an ACO, SA and VNS Hybrid Algorithm". Expert Systems with Applications, 2009, 36, pp. 9637–9644.
7. G. Bektur, and T. Saraç, "A mathematical model and heuristic algorithms for an unrelated parallel machine scheduling problem with sequence-dependent setup times, machine eligibility restrictions and a common server". Journal of Computers and Operations Research, 103, 2019, pp. 46-63.

8. Z. Chen, and W. B. Powell, "Solving parallel machine scheduling problems by column generation". INFORMS Journal on Computing, 11, 1999, pp. 78–94.
9. L. P. Cota, F. G. Guimarães, F. B. de Oliveira, and M. J. Freitas Souza, "An Adaptive Large Neighborhood Search with Learning Automata for the Unrelated Parallel Machine Scheduling Problem". IEEE Congress on Evolutionary Computation (CEC), 2017, pp. 185-192.
10. I.I. Cplex: V12. 9, "User's Manual for Cplex". IBM, 46(53), pp. 157.
11. A. Elidrissi, R. Benmansour, and A. Sifaleras, "General variable neighborhood search for the parallel machine scheduling problem with two common servers". Optimization Letters, 2022, pp. 1-31.
12. W. Fang, H. Zhu, and Y. Mei, "Hybrid meta-heuristics for the unrelated parallel machine scheduling problem with setup times". Knowledge-Based Systems Journal, 241, 2022, 108193 p.
13. L. Fanjul-Peyro, R. Ruiz, and F. Perea, "Reformulations and an exact algorithm for unrelated parallel machine scheduling problems with setup times". Computers and Operations Research Journal, 101, 2019, pp. 173-182.
14. C.A. Glass, C.N. Potts, and P. Shade, "Unrelated parallel machine scheduling using local search". Mathematical and Computer Modelling Journal, 20, 1994, pp. 41–52.
15. Y. Hadhbi, L. Deroussi, N. Grangeon, and S. Norre, "Improved Formulations and Branch-and-Cut Algorithm for the Unrelated Parallel Machines Scheduling Problem with a Common Server and Job-Sequence Dependent Setup Times". 9th International Conference on Control, Decision and Information Technologies (CoDIT), 2023, Rome, Italy, pp. 1-6.
16. L.A. Hall, "Approximation algorithms for scheduling". Book of Approximation Algorithms for NP-Hard Problems, D.S. Hochbaum, editor, PWS Publishing, Boston, MA, 1997, pp. 1–45.
17. M. Helal, G. Rabadi, A. Al-Salem, "A tabu search algorithm to minimize the makespan for the unrelated parallel machines scheduling problem with setup times". IJOR, 3, 2006, 182-192.
18. A.A. Juan, H. R. Lourenço, M. Mateo, R. Luo, Q. and Castella, "Using iterated local search for solving the flow-shop problem: Parallelization, parametrization, and randomization issues". International Transactions in Operational Research, 21, 2014, pp. 103-126.
19. J.K. Lenstra, B.S. David, and E. Tardos, "Approximation algorithms for scheduling unrelated parallel machines". Mathematical Programming, 46, 1990, pp. 259–271.
20. S.W. Lin, C.C. Lu, and K.C. Ying, "Minimization of total tardiness on unrelated parallel machines with sequence- and machine-dependent setup times under due date constraints". International Journal of Advanced Manufacturing Technology, 53, 2011, pp. 353–361.
21. H.R. Lourenço, O.C. Martin, T. Stützle, "Iterated Local Search". Glover, F., Kochenberger, G.A. (eds) Handbook of Metaheuristics, International Series. Operations Research & Management Science, vol 57. Springer, Boston, MA, 2003.
22. E. Mokotoff, and P. Chrétienne, "A cutting plane algorithm for the unrelated parallel machine scheduling problem". European Journal of Operational Research, 141, 2002, pp. 515–525.
23. S. Martello, F. Soumis, and P. Toth, "Exact and approximation algorithms for makespan minimization on unrelated parallel machines". Discrete Applied Mathematics Journal, 75, 1997, pp. 169-188.
24. Z. Pei, M. Wan, and Z. Wang, "A new approximation algorithm for unrelated parallel machine scheduling with release dates". Annals of Operations Research, 285, 2020, pp. 397–425.
25. M.J. Pereira Lopes, and J.M.V. De-Carvalho, "A branch-and-price algorithm for scheduling parallel machines with sequence dependent setup times". EJOR, 176, 2007, pp. 1508–1527.
26. G. Rabadi, R. Moraga, and A. Al-Salem, "Heuristics for the unrelated parallel machine scheduling problem with setup times". Journal of Intelligent Manufacturing, 17, 2006, pp. 85-97.
27. H. Raboudi, G. Alpan, F. Mangione, G. Tissot, and F. Noels, "Scheduling unrelated parallel machines with a common server and sequence dependent setup times". 10th IFAC Conference on Manufacturing Modelling, Management and Control MIM 2022, 55, 2022, pp. 2179-2184.
28. S. Radhakrishnan, and J. A. Ventura, "Simulated annealing for parallel machine scheduling with earliness-tardiness penalties and sequence-dependent setup times". International Journal of Production Research, 38, 2000, pp. 2233–2252.
29. P. Rocha, M. Ravetti, G. Mateus, and P. Pardalos, "Exact algorithms for a scheduling problem with unrelated parallel machines and sequence and machine-dependent setup times". Computers and Operations Research Journal, 35, 2008, pp. 1250-1264.
30. T.T. Tran, A. Araujo, and J.C. Beck, "Decomposition methods for the parallel machine scheduling problem with setups". INFORMS Journal on Computing, 28, 2016, pp. 83-95.
31. E. Vallada, and R. Ruiz, "A genetic algorithm for the unrelated parallel machine scheduling problem with sequence-dependent setup times". EJOR, 211, 2011, pp. 612–622.
32. V. V. Vazirani, "Scheduling on Unrelated Parallel Machines". Approximation Algorithms, Springer, Berlin, Heidelberg, 2003.
33. A. Wotzlaw, "Scheduling Unrelated Parallel Machines—Algorithms, Complexity, and Performance". VDM Verlag Dr. Mueller e.K., 2007.
34. J. R. Zeidi, and Sa. M. Hosseini, "Scheduling unrelated parallel machines with sequence-dependent setup times". The International Journal of Advanced Manufacturing Technology, 81, 2015, pp. 1487–1496.