International Journal of Applied Science and Engineering

Heuristics for parallel machine scheduling with GoS eligibility constraints

Chao-Tang Tseng*, Shu-Fu Zhang

Department of Industrial Engineering and Management, Chaoyang University of Technology, Taichung City 413310, Taiwan

ABSTRACT

The parallel machine scheduling problem with the consideration of GoS (grade of service) constraints originates from the service provision of different membership levels in the service industry. This scheduling problem has been widely found in industrial manufacturing and sustainable manufacturing practices. For example, in a sustainable manufacturing production line, multiple machines are set up as parallel machines to support each other to achieve shared manufacturing. However, the machines still have processing constraints with different service level constraints and do not support all jobs. This study investigates this parallel machine scheduling problem and uses the total completion time as the objective criterion. This criterion has not been studied in the literature for this scheduling problem. Several heuristics are proposed to solve the two-machine and multi-machine scheduling problems. The structure of these heuristics is mainly divided into two stages: the SPT rule and the insertion method. In addition, a fast method for calculating the total completion time is developed from the insertion method. The experimental results prove that the proposed heuristics are effective in solving this scheduling problem. For both small- and large-size problems, the proposed heuristics can obtain good solutions quickly and are potentially suitable for practical use.

Keywords: Eligibility constraints, Heuristic, Parallel machine, Scheduling, Total completion time.

Received: April 23, 2023
Revised: May 28, 2023
Accepted: June 11, 2023

Corresponding Author:
Chao-Tang Tseng
cttseng@cyut.edu.tw

Publisher:
Chaoyang University of Technology
ISSN: 1727-2394 (Print)
ISSN: 1727-7841 (Online)

1. INTRODUCTION

The problem of conventional parallel machine scheduling has been extensively studied, and in practice, eligibility constraints often arise (Berthier et al., 2022; Maecker et al., 2023). This means that job is restricted to certain machines. In recent years, the types of eligibility constraints commonly considered in parallel machine scheduling problems are general, interval, nested, grade of service (GoS), etc. (Lim, 2010; Leung and Li, 2016; Bektur and Saraç, 2019; Li et al., 2021; Mecler et al., 2022). In terms of GoS constraints, many practical examples can be found in service industries, industrial manufacturing, and sustainable manufacturing, as shown in Table 1, indicating the growing importance of this issue. The following is a detailed description.

Table 1. The practical cases of GoS constraints

Service industry	Industrial manufacturing	Sustainable manufacturing
Bank (Hwang et al., 2004)	Flour mill (Glass and Kellerer, 2007)	Tempered glass processing (Liu et al., 2020)
Credit card (Hwang et al., 2004)	Loading/unloading cranes (Ou et al., 2008)	Shared manufacturing (Ji et al., 2022)
Online antivirus (Tseng et al., 2017)	Painting processing (Mateo et al., 2018)	
	Sugarcane unloading systems (Kusoncum et al., 2021)	

The research on GoS constraints started with the scheduling problem in the service industry. The service industry provides different services according to customer membership levels (Hwang et al., 2004). In practice, companies classify customers into different classes and set up multiple identical service windows for each class. For example, airlines divide their customers into economy class and business class and above. Check-in counters are also divided into counters for economy class passengers and business class passengers and above, with multiple counters for each class. Economy class passengers will check in at the economy class counter, while passengers of business class and above will check in at the business class counter to ensure better service. However, when the business class counters are available, economy class passengers may be directed to the business class counters by service personnel. Passengers and counters are subject to a specific service level constraint. This is a typical parallel flight scheduling problem that takes into account GoS constraints. It is noted that this flight scheduling problem is only an example to illustrate GoS constraints. This scheduling problem considers the different arrival times of customers and the random processing time, which is not considered in this study, as detailed in the following research limitations. Similar GoS constraints are often found in other service industries, such as banking, credit card services, and online real-time anti-virus services (Tseng et al., 2017).

GoS constraints are also applied in sustainable manufacturing. In sustainable manufacturing, energy-efficient PPC (production planning and control) is an important issue, especially for scheduling (Akbar and Irohara, 2018). Scheduling in sustainable manufacturing, which is called the sustainable scheduling, considers the sustainable manufacturing indicator as a criterion. Akbar and Irohara (2018) reviewed sustainable scheduling research and found that three types of objectives are considered: economic-oriented, environment-oriented, and social-oriented objectives. The research on sustainable scheduling with the GoS constraint has focused on an environment-oriented objective. Liu et al. (2020) proposed the power consumption as a scheduling criterion, which is an environment-oriented objective. They examined the three-stage process of tempered glass: cutting, printing, and tempering, which are composed of high-energy-consuming machines. From a sustainability perspective, the GoS constraint arises, where different machines have processing constraints in terms of glass thickness conditions and different power consumptions. The GoS constraint is used to meet the environment-oriented objective. Ji et al. (2022) investigated the problem of shared manufacturing schedules with the concept of sustainable manufacturing. Each job can be processed in a set of machine sets, and such a new constraint is called a processing set. And the GoS constraint is one of its special cases. This scheduling problem applies the GoS constraint to achieve capacity utility maximization, which is an environment-oriented objective. Therefore, in the above literature, the GoS constraint has been implemented in the service industry, industrial manufacturing, and even in the future of sustainable manufacturing.

The parallel machine scheduling environment with GoS constraints is described as follows: There are m machines ($M_i; i = 1,2,\ldots,m$), and each machine M_i has its service level $g(M_i)$. Here the service level is 1 for the highest level, 2 for the second level, and so on. There are n jobs ($J_j; j = 1,2,\ldots,n$) and each job J_j has its service level $g(J_j)$ and processing time p_j. Job J_j is assigned to a machine that satisfies the condition $g(J_j) \geq g(M_i)$. For an example of 2 jobs, 2 machines, and 2 GoS levels, assume that $g(J_1) = 1$, $g(J_2) = 2$, $g(M_1) = 1$, and $g(M_2) = 2$. Job 1 can only be machined on Machine 1 because it satisfies the condition $g(J_1) \geq g(M_1)$. Machine 2 is a machine with a lower GoS level that cannot process Job 1 with a higher GoS level, i.e., it cannot satisfy the condition $g(J_1) \geq g(M_2)$. Conversely, if job 2 is a lower GoS level job, then both machines 1 and 2 can be machined. Also, the research limitations are to:

1. All the processing time and number of jobs are known and deterministic.
2. All jobs arrive at the same time, and we do not consider on-line scheduling model.
3. The process time p_j that job J_j spends on any machine is the same.
4. Job J_j requires a single operation and may be processed on any one of the m machine that belongs to a given subset.
5. Once a job on a machine has started, it cannot be stopped until it is completed.

There is no study in the literature that uses the minimization of total completion time as a criterion, i.e., $P_m|GoS|\sum C_j$. To the best of our knowledge, this study is the first paper to investigate this scheduling problem. For $1||\sum C_j$, the shortest processing time (SPT) rule can be used to obtain the optimal solution. For $P_m||\sum C_j$, the SPT rule can also be used to obtain the optimal solution. However, the problem $P_m||\sum C_j$ becomes an NP-hard problem by adding constraints such as the GoS constraint (Mokotoff, 2001). The main contributions and innovations of this study are to:

1. A new scheduling problem of minimizing total completion time on parallel machines with GoS constraint is considered.
2. Effective heuristics are proposed to solve the considered problem.
3. A fast method for calculating the total completion time is developed from the insertion method.
4. All proposed heuristics outperform the metaheuristic in literature and are potentially suitable for practical use.

This study is organized as follows. Section 2 presents a review of the literature. In Sections 3 and 4, the heuristics are developed to solve the two- and multi-machine problems. Section 5 evaluates the performance of these heuristics. Finally, conclusions are provided in Section 6.

2. LITERATURE REVIEW

Most of the studies on the above parallel machine scheduling environment take minimizing makespan as the objective function, which can be denoted as $P_m|GoS|C_{max}$, and it is an NP-hard (as shown in Table 2). Since the development of heuristics by Hwang et al. (2004) and Glass and Kellerer (2007), researchers have started to propose a series of related polynomial time approximation schemes (PTAS) (Ji and Cheng, 2008; Ou et al., 2008; Li and Zhang, 2009; Woeginger, 2009). Besides, some literature refers to the GoS constraint as inclusive processing set restriction (Ou et al, 2008), both of which have the same meaning. In addition, more complex scheduling contexts have been studied, such as the inclusion of start time constraints (Huo et al., 2009; Li and Wang, 2010; Li and Li, 2015) or uniform parallel machine environments (Epstein and Levin, 2011). Literature reviews are presented in Leung and Li (2008), Lim (2010), Leung and Li (2016), and they also review studies that consider online and semi-online contexts.

In recent years, studies on GoS constraints have focused on exploring different parallel machine scheduling environments or minimizing other objectives. For different parallel machine scheduling environments, Li (2017) developed two fast algorithms for parallel batch machine scheduling with start time constraints and even proposed PTAS as well. Leung and Ng (2017), on the other hand, extended the study of uniform parallel machine scheduling to improve the PTAS proposed by Epstein and Levin (2011). For minimizing other objectives, Ou et al. (2016) solved the bi-objective parallel machine scheduling problem of minimizing the penalty cost of makespan and exiting jobs, and developed PTAS. Later, researchers started to use metaheuristics to solve parallel machine scheduling problems with different objectives. Tseng et al. (2017) proposed an electromagnetic-like algorithm to efficiently solve the parallel machine scheduling problem that minimizes the total weighted tardiness. Liu et al. (2020) used difference algorithm to solve sustainable scheduling problems with bi-objective of minimizing makespan and total electricity cost. Liao et al. (2020) considered outsourcing cases to minimize makespan and outsourcing costs and proposed VNS-NKEA algorithm to solve it. Kusoncum et al. (2021) developed the VaNSAS algorithm to solve the GoS unrelated parallel machine scheduling problem under more constraints.

3. TWO-MACHINE PROBLEM

This section examines the case of two machines with two service levels $P_2|GoS = 2|\sum C_j$. Without loss of generality, it is assumed that Machine 1 (M_1) provides a high service level $(g(M_1) = 1)$ and Machine 2 (M_2) provides a low service level $(g(M_2) = 2)$. Two heuristics are proposed to solve this scheduling problem, as detailed below.

3.1 SPT-I

The proposed first heuristic called SPT-I uses a two-stage approach.

Table 2. Summary of the GoS papers

Type	Criterion	Constraint	Method	Sources
P_m	Makespan	x	Heuristic	Hwang et al. (2004); Glass and Kellerer (2007)
P_m	Makespan	x	PTAS	Ji and Cheng (2008); Ou et al. (2008); Li and Zhang (2009); Woeginger (2009)
P_m	Makespan	Start time Preemptions	Algorithm	Huo et al. (2009)
P_m	Makespan	Start time	PTAS	Li and Wang (2010)
P_m	Makespan	Start time Process	Algorithm	Li and Li (2015)
Q_m	Makespan	x	PTAS	Epstein and Levin (2011)
B_m	Makespan	Start time	PTAS	Li (2017)
Q_m	Makespan	x	PTAS	Leung and Ng (2017)
P_m	Makespan Cost	x	PTAS	Ou et al. (2016)
P_m	Total tardiness	x	Metaheuristic	Tseng et al. (2017)
HFS	Makespan Total electricity cost	x	Metaheuristic	Liu et al. (2020)
P_m	Makespan Outsourcing cost	Group	Metaheuristic	Liao et al. (2020)
R_m	Makespan	Setup time	Metaheuristic	Kusoncum et al. (2021)
P_m	Total completion time	x	Heuristic	This study

In Stage 1, each job is assigned to the machine corresponding to GoS according to the GoS constraint, and the SPT rule is used to find the optimal solution for minimizing the total completion time of a single machine, so the jobs of both machines are used to find the total completion time in Stage 1 according to the SPT rule. There are other dispatching rules, such as the EDD rule (the earliest due date), but those rules are not suitable for use in this scheduling problem. For $1||\sum C_j$ or $P_m||\sum C_j$, the SPT rule can be used to obtain the optimal solution. Here, the local optimal solution is used as a good initial solution in the proposed heuristic for the considered scheduling problem. For the $\sum C_j$ criterion, other dispatching rules cannot obtain better initial solutions than the SPT rule (Pinedo, 2002). Therefore, the first stage of all proposed heuristics uses the SPT rule. In Stage 2, each job from the low service level machine is tried to be inserted into the high service level machine to reduce the total completion time. The detailed steps are as follows.

```
Stage 1: SPT
Step 1.1: Let $S_1$ be the set of jobs on machine $M_1$ with high
          service level and $S_2$ be the set of jobs on machine
          $M_2$ with low service level. Put all jobs into $S_1$ and
          $S_2$ respectively according to the service level.
Step 1.2: Arrange the jobs in $S_1$ and $S_2$ in descending order
          of processing time, and calculate the total
          completion time.
Stage 2: Insertion
Step 2.1: Let $Z$ be the unscheduled sets of jobs, $k = 1$, $Z =$
          $S_2$ and $n_2$ be the number of jobs in $Z$.
Step 2.2: Select the job in the $k$th position of $Z$ and insert it
          into $S_1$ according to the order of SPT rule.
Step 2.3: Recalculate the total completion time. If the total
          completion time improves, remove the job from
          $S_2$ and insert it into $S_1$. Otherwise, do not insert it.
Step 2.4: Consider the next job $k = k + 1$ and go back to
          Step 2.2 until $k = n_2$ and the algorithm stops.
```

Stage 1 is to arrange the corresponding service level jobs on the machine according to the SPT rule. First, in Step 1.1, the jobs are divided into two groups according to the service levels and assigned to the corresponding machines. High service level machines process high service level jobs and low service level machines process low service level jobs. Due to the classification, the scheduling problem for each of the two machines becomes a single machine scheduling problem with minimizing total completion time ($1||\sum C_j$). In Step 1.2, the SPT rule is used to find the optimal solution for each machine. Finally, the total completion time of the two machines is summed and this is the optimal solution in the case of the classification.

Next, M_1 is a high service level machine that can machine not only high service level jobs but also low service level jobs. Stage 2 tries to insert the job from the low service level machine into the high service level machine to check if the total completion time can be reduced. In Step 2.1, let $k = 1$ and start selecting jobs from the first position on machine M_2 and do not stop until n_2 jobs. In Step 2.2, the jobs selected on machine M_2 are inserted into S_1. Since the optimal solution can be obtained by the SPT rule on this machine, the selected insertion position must also be inserted into the corresponding position in S_1 by maintaining the SPT rule. It is noted that there is no condition that must be met first before inserting the selected job into the corresponding position. This is because the processing time and number of jobs are known. In Step 2.3, the total completion time is recalculated, and if it improves, it is inserted, otherwise it is not inserted. In Step 2.4, Steps 2.2-2.3 are repeated until all jobs in Z are considered to reduce the total completion time, and the algorithm stops.

This heuristic is illustrated by an example of 7 jobs, 2 machines, and 2 service levels. The relevant information is shown in Table 3, where p_j and $g(J_j)$ represent the processing time and service level of job J_j. $g(M_1) = 1$ and $g(M_2) = 2$. The steps of the heuristic are described as follows.

Table 3. Job data for example

J_j	1	2	3	4	5	6	7
p_j	7	5	15	12	3	20	18
$g(J_j)$	1	1	2	2	2	1	2

```
Stage 1: SPT
Step 1.1: According to the service level of the job, $S_1 =$
          $\{J_1, J_2, J_6\}$ and $S_2 = \{J_3, J_4, J_5, J_7\}$ can be obtained.
Step 1.2: The jobs in $S_1$ and $S_2$ can be obtained as $S_1 =$
          $\{J_2, J_1, J_6\}$ and $S_2 = \{J_5, J_4, J_3, J_7\}$ according to the
          SPT rule. As shown in Fig. 1, the total completion
          time of Machine $M_1$ is 49, the total completion
          time of Machine $M_2$ is 96, and the total
          completion time of the objective function is 145.
```

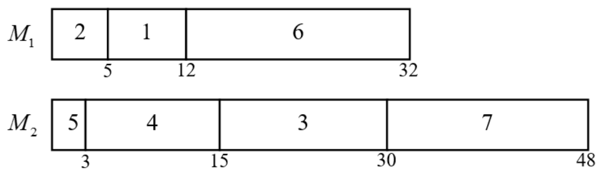

Fig. 1. Schedule after Phase 1

```
Stage 2: Insertion
Step 2.1: Let $k = 1$, $Z = \{J_5, J_4, J_3, J_7\}$ and $n_2 = 4$
Step 2.2: Select the job $J_5$ at the first position in $Z$. Next,
          insert $J_5$ before $J_2$ in $S_1$, as the best total
          completion time for the machine is obtained by
          following the SPT rule. If $J_5$ is inserted in another
          position, the total completion time is not shorter.
Step 2.3: Recalculate the total completion time and find that
          the total completion time of $M_1$ is 61, the total
          completion time of machine $M_2$ is 84, and the
          total completion time of the objective function is
```

145. the total completion time is not improved, so the selected job is not inserted.

Step 2.4: Go back to step 2.2 if $k = 2$.

Step 2.2: Select the job J_4 at the second position in Z. Next, insert J_4 between J_1 and J_6 in S_1 according to the SPT rule.

Step 2.3: Recalculate the total completion time, which is 85 for machine M_1, 57 for machine M_2, and 142 for the objective function, and improve the total completion time by removing job J_4 from S_2 and inserting it into S_1. Therefore, the new $S_1 = \{J_2, J_1, J_4, J_6\}$ and $S_2 = \{J_5, J_3, J_7\}$.

Step 2.4: Go back to Step 2.2 if $k = 3$

$\vdots$

Finally, neither the insertion of J_3 nor J_7 can improve the total completion time. As shown in Fig. 2, $S_1 = \{J_2, J_1, J_4, J_6\}$ and $S_2 = \{J_5, J_3, J_7\}$ can be obtained, and the total completion time for the objective function is 143.

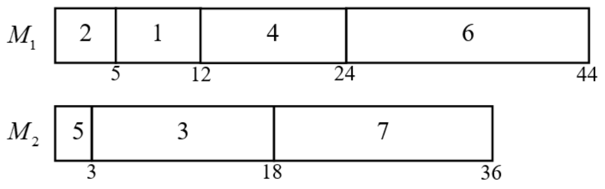

Fig. 2. Final schedule

3.2 SPT-FI

The proposed second heuristic, called SPT-FI, mainly accelerates the calculation speed of the SPT-I heuristic. In Step 2.3 of the SPT-I heuristic in the previous section, the job insertion requires recalculation of the total completion time of both machines. This section discusses the characteristics of insertion and removal and proposes a fast calculation method that can reduce the calculation time by a significant amount. It is divided into two parts to illustrate: (1) the change of the selected job after its removal from machine M_2 and (2) the change of the selected job after its insertion into machine M_1.

First, discuss the change in the total completion time after the selected jobs are moved out of the machine M_2. Assume that M_2 has a jobs. Let $S_2 = \{J_{[1]}, \ldots, J_{[k-1]}, J_{[k]}, J_{[k+1]}, \ldots, J_{[a]}\}$ is the schedule of M_2, where $J_{[k]}$ means the job of the kth position and is selected to move out of the job. Let $C_{[k]}$ and $p_{[k]}$ represent the completion time and machining time of $J_{[k]}$. The total completion time $\sum_{i=1}^{a} C_{[i]}$ of M_2 is calculated and the formula is developed as follows.

$$
\begin{gathered}
C_{[1]} = p_{[1]} \\
C_{[2]} = p_{[1]} + p_{[2]} \\
C_{[3]} = p_{[1]} + p_{[2]} + p_{[3]} \\
\vdots \\
C_{[k-1]} = p_{[1]} + p_{[2]} + p_{[3]} \cdots + p_{[k-1]} \\
C_{[k]} = p_{[1]} + p_{[2]} + p_{[3]} \cdots + p_{[k-1]} + p_{[k]} \\
C_{[k+1]} = p_{[1]} + p_{[2]} + p_{[3]} \cdots + p_{[k-1]} + p_{[k]} + p_{[k+1]} \\
\vdots \\
C_{[a]} = p_{[1]} + p_{[2]} + p_{[3]} \cdots + p_{[k-1]} + p_{[k]} + p_{[k+1]} \cdots \\
+ p_{[a]}
\end{gathered}
$$

Expanding from the above equation, it can be seen that after moving out $J_{[k]}$, the total completion time will be reduced by $C_{[k]}$, the completion time of $J_{[k]}$, and $(a-k)$ times $p_{[k]}$, the processing time of $J_{[k]}$, as follows.

$$C_{[k]} + (a-k) \times p_{[k]} \tag{1}$$

Then, the change in the total completion time after inserting the selected job into the machine M_1 is discussed. Assume that M_1 has b jobs. Let $S_1 = \{J_{[1]}, \ldots, J_{[i]}, J_{[j]}, \ldots, J_{[b]}\}$ be the schedule of M_1. Further, assume that $J_{[k]}$ is inserted between $J_{[i]}$ and $J_{[j]}$ on M_2. The equation for the total completion time $\sum_{i=1}^{b} C_{[i]}$ after the insertion of $J_{[k]}$ on M_1 is expanded as follows.

$$
\begin{gathered}
C_{[1]} = p_{[1]} \\
C_{[2]} = p_{[1]} + p_{[2]} \\
\vdots \\
C_{[i]} = p_{[1]} + p_{[2]} \cdots + p_{[i]} \\
C'_{[k]} = p_{[1]} + p_{[2]} \cdots + p_{[i]} + p_{[k]} \\
C_{[j]} = p_{[1]} + p_{[2]} \cdots + p_{[i]} + p_{[k]} + p_{[j]} \\
\vdots \\
C_{[b]} = p_{[1]} + p_{[2]} \cdots + p_{[i]} + p_{[k]} + p_{[j]} \cdots + p_{[b]}
\end{gathered}
$$

Expanding from the above equation, the total completion time after inserting $J_{[k]}$ will increase $C'_{[k]}$, the completion time of $J_{[k]}$, and $(b-k)$ times $p_{[k]}$, the processing time of $J_{[k]}$, as follows.

$$C'_{[k]} + (b-k) \times p_{[k]} \tag{2}$$

Finally, this study integrates the reduction in Equation (1) and the increase in Equation (2) into the following equation.

$$\varDelta = [C'_{[k]} + (b-k) \times p_{[k]}] - [C_{[k]} + (a-k) \times p_{[k]}] \tag{3}$$

where $\varDelta$indicates the change in the total completion time. Therefore, Step 2.3 in SPT-I is completely recalculated and the total completion time is changed to the following.

Step 2.3: Equation (3) is calculated, and if $\varDelta < 0$, it means the total completion time is improved, then remove the job from S_2 and insert it into S_1. Otherwise, it is not inserted.

In this step, instead of recalculating the total completion time of the two machines, the new total completion time can be calculated quickly according to Equation (3) to determine whether to insert or not. We call this heuristic SPT-FI.

4. MULTI-MACHINE PROBLEM

This section examines the case of multiple machines with multiple service levels $P_m|GoS|\sum C_j$. Assume that there are m machines and g service levels. This scheduling problem becomes more complicated when there are more than 2 machines at each service level. The concepts of the two heuristic algorithms in the previous section will be applied to solve this problem. The details are as follows.

4.1 mSPT-I

The difference with SPT-I in Section 3.1 is that there are multiple machines at a given service level and the selected job can be inserted into a larger selection of machines. In Stage 1 of the heuristic, jobs are assigned to the appropriate set of service level machines according to the service level constraints. Each set forms the $P_m||\sum C_j$ problem. The optimal solution is found for each set according to the SPT rule. In Stage 2, we try to insert each job from the low service level set into each service level set higher than it. The most improved solution is chosen from each insertion solution. The detailed steps are as follows.

```
Stage 1: SPT
Step 1.1: Let $S_1, S_2, \ldots, S_g$ be the set of service levels
          $1,2,\ldots,g$. Put all jobs into $S_1, S_2, \ldots, S_g$
          according to their service levels.
Step 1.2: Arrange the jobs in $S_1, S_2, \ldots, S_g$ in descending
          order of processing time.
Step 1.3: Assign the jobs in $S_1, S_2, \ldots, S_g$ to the earliest idle
          machine under each service level according to the
          processing order, and calculate the total
          completion time.
Stage 2: Insertion
Step 2.1: Let $l = g$
Step 2.1.1: Let $Z$ be the unscheduled sets of jobs, $k = 1$,
            $Z = S_l$, and $n$ be the number of jobs in $Z$
Step 2.1.2: Select the job at the $k$th position of $Z$
Step 2.1.3: Evaluate the different insertion solutions. Try to
            insert the selected job $J_{[k]}$ into the solutions
            $S_1, S_2, \ldots, S_{j-1}$ according to the SPT rule and
            calculate the total completion time for each
            solution. Select the solution with the shortest
            total completion time.
Step 2.1.4: If the selected insertion solution can improve the
            total completion time, update the service level
            sets, otherwise, leave them unchanged.
Step 2.1.5: Consider the next job, $k = k + 1$. Return to Step
            2.1.2 until $k = n$. Go to the next step.
Step 2.2: $l = l - 1$, go back to Step 2.1.1 until $l = 1$, and
          the algorithm stops.
```

First, in Stage 1, the jobs are assigned to the machines according to the SPT rule. In Step 1.1, each job is placed in the service level sets $S_1, S_2, \ldots, S_g$ according to its service level. In Step 1.2, the jobs in each set are arranged according to the SPT rule. Since the jobs in each set are processed on multiple parallel machines, Step 1.3 assigns the jobs to the machines in the order of the sets using the ASAP (as soon as possible) method. Finally, the total completion time is calculated.

Secondly, Stage 2 starts the improvement of scheduling using the insertion method. In Step 2.1, the jobs are considered to be moved from the lowest service level set S_g to the higher service level sets. In Steps 2.1.1 to 2.1.2, the number of jobs in the selected service level set is confirmed, and the next step is performed starting with the job at the first position. In Step 2.1.3, the total completion time of each solution for the selected jobs $J_{[k]}$ is evaluated for insertion into the higher service level sets, and the optimal solution is selected. Since there are multiple parallel machines in a set, the jobs are reassigned to the machines according to the new machining sequence and ASAP method. Therefore, the total completion time has to be recalculated. In Step 2.1.4, if an improvement solution can be found, then the change will be made, otherwise, it will remain unchanged. In Step 2.1.5, Steps 2.1.2-2.1.4 are repeated until all jobs in the selected set have been considered before proceeding to the next step. In step 2.2, the next higher service level set is considered until the highest service level is reached and the algorithm is stopped.

4.2 mSPT-FI

In the previous section, mSPT-I cannot be directly applied to Equation (3) in Section 3.2 to speed up the calculation of the new total completion time when the processing order in a set changes, because Equation (3) is limited to the case where each set is a single machine. mSPT-I is used to consider the case where each set is a parallel machine, and when the processing order in a set changes, it is necessary to follow the ASAP method, and the jobs may be assigned to the machines with the original processing order. The machines to which the jobs are assigned will be changed and the calculation has to be done again.

To improve the computational performance of the heuristic with the use of Equation (3), the Insertion steps of mSPT-I in Section 3.1 are changed to evaluate different insertion solutions based on a single machine. The heuristic starts with the same step as Step 1 of mSPT-I and generates a schedule according to the SPT rule. To save time in calculating the new total completion time and to achieve a better solution search, Steps 2.1.3-2.1.4 of mSPT-I are changed to the following.

```
Step 2.1.3: Evaluate the different insertion solutions. Try to
            insert the selected job $J_{[k]}$ into each machine
            that satisfies $g(M_i) \le g(J_{[k]})$ as a solution.
```

Apply Equation (3) and calculate the change in total completion time for each solution. Select the solution with the best improvement.

Step 2.1.4: If the selected insertion solution can improve the total completion time, update the scheduling and service level sets on the machine, otherwise do not change them.

There are two main differences between mSPT-FI in this section and mSPT-I in the previous section: (1) mSPT-FI uses accelerated computation; (2) mSPT-FI is inserted based on the machine, while mSPT-I is based on the service level set.

5. RESULTS AND DISCUSSION

To test the performance of the proposed heuristics, this study conducts comparative experiments on a series of problems. The experimental examples are generated by referring to the way Tseng et al. (2017) generated the benchmark instances for $P_m|GoS|\sum W_jT_j$. The processing time and GoS level of each job are randomly generated from the uniform distribution of $U[1,100]$ and $U[1,G]$. Tseng et al. (2017) also developed an Electromagnetism-like Mechanism (EM) algorithm that proved to be effective in solving $P_m|GoS|\sum W_jT_j$ problems. Therefore, this study applies the EM proposed by Tseng et al. (2017) to solve $P_m|GoS|\sum C_j$ and compare it with the proposed heuristics to verify their performance. All algorithms were coded using Visual C ++ and executed on an Intel®Core™ i7-3687U CPU 2.10GHz PC with 8GB RAM memory. The following experimental results for a series of problems are described as follows.

5.1 Two-machine Problem

In the case of considering 2 machines and 2 service levels, one machine is at a high service level and the other is at a low service level. The experiments are structured on 6 sets of job numbers (n =20, 50, 100, 200, 500 and 1000). According to Tseng et al. (2017), 20 instances are randomly generated for each set of jobs, for a total of 120 instances. The performance is measured using the Relative Percentage Increase (RPI) index with the following formula.

$$H_i - min(H_j, \forall j)\ /min(H_j, \forall j) \times 100 \qquad (4)$$

H_i denotes the total completion time generated by Algorithm i. $min(H_j, \forall j)$ denotes the minimum value among all compared algorithms. The compared algorithms are SPT, SPT-I, SPT-FI and the EM proposed by Tseng et al. (2017), where SPT is the Stage 1 solution in SPT-I/SPT-FI, and EM denotes the EM proposed by Tseng et al. (2017) that has been shown to effectively solve the problem of $P_m|GoS|\sum W_jT_j$. Their EM is modified here to solve the problem considered in this study. EM is an algorithm with random search properties. Therefore, the same instance EM is run for 10 trials, and the smallest total completion time is selected as the comparative solution. The parameters of EM are set using the settings of Tseng et al. (2017). The experimental results are shown in Table 4, presenting the average RPI (MRPI) and average computation time (CT) of the four algorithms. The results of SPT-I and SPT-FI are considered the same here. Since the proposed SPT-I and SPT-FI produce the same solution considering two machines and two service levels, the difference in CT is also small enough to be negligible.

First, it can be seen from Table 4 that the proposed SPT-I/SPT-FI is competitive. Its MRPI is all 0 from small-size to large size jobs, which means that the solution of SPT-I/SPT-FI is the best for all problems and instances. Secondly, SPT in the table is the method of SPT-I/SPT-FI in Stage 1. By comparing the MRPI values of SPT with those of SPT-I/SPT-FI, the improved performance of SPT-I/SPT-FI in the Stage 2 method can be seen. In Table 3, the MRPI values for SPT are slightly larger for small-size jobs than for large size jobs. This shows that SPT performs less well on small-size jobs. In other words, it is implied that the SPT-I/SPT-FI in Stage 2 works better for small-size jobs. On the contrary, from the MRPI values, SPT performs very well for large size jobs, and the effect of Stage 2 is very small, especially for jobs of 1000.

Thirdly, EM has been able to obtain very good performance in solving similar problems, which is a good basis for comparison. By observing the MRPI of EM and SPT-I/SPT-FI, we can understand the performance of SPT-I/SPT-FI. In Table 4, the MRPI of EM increases with the number of jobs, indicating that the performance of EM in solving with larger numbers of jobs is not good. Although EM can already obtain good performance in solving similar problems, this result also implies that SPT-I/SPT-FI performs better for large jobs. The MRPI of EM is 58.4% for a job number of 1000. In other words, the SPT-I/SPT-FI performance is better than the EM performance of 58.4%. Overall, the SPT-I/SPT-FI average performance is better than the EM average performance of 38.4%. In terms of execution time, SPT-I/SPT-FI also has a considerable advantage. When the number of jobs is 1000, EM takes 16.5 sec, while SPT-I/SPT-FI takes only 0.037 sec. In the next section, more machines and more service levels are considered to further understand if the performances of the proposed heuristics are different.

5.2 Multi-machine Problem

To further validate the performance of the proposed heuristics at multi-machine and multi-service levels, small-size and large size job experiments are constructed. For the first group, experimental data of small-size jobs was directly used from the benchmark instances proposed by Tseng et al. (2017). There are three job numbers (n = 20, 50 and 100), two machine numbers (m = 4 and 10), and two service level numbers (GoS = 2 and 3), for a total of 12 combinations.

Each combination has 80 instances, for a total of 960 instances. For the second group, experimental data of large jobs is constructed on the job numbers (n = 200, 500 and 1000). The combinations of machine numbers and service level numbers are considered the same as the first group of small job experiments. Since there is no benchmark instance for the large size jobs, each instance in this study is randomly generated using the method of Tseng et al. (2017). There are 80 instances for each combination, totaling 960 instances.

5.2.1 Small-size Job Experiment

First, as shown in Table 5, the average MRPIs of mSPT-I and mSPT-FI are 0.438% and 0.471% respectively, which means that the performances of the proposed mSPT-I and mSPT-FI are significantly better than those of SPT and EM. Furthermore, the performances of mSPT-I and mSPT-FI are compared in detail. In Table 5, mSPT-I has the best performance for a GoS of 2, regardless of the number of jobs and the number of machines. The results of mSPT-FI at GoS of 3, however, are mostly the best performance except for 50/10/3 and 100/10/3. Therefore, the small-size job experiments show that mSPT-I has a greater advantage in solving the GoS of 2, while mSPT-FI has a greater advantage in solving the GoS of 3.

Table 6 presents the number of instances won by comparing the solutions of mSPT-I and mSPT-FI. Tie indicates that the solutions of mSPT-I and mSPT-FI are the same. For example, for a 20/4/2 problem with 80 instances, mSPT-I has 33 better solutions, and 44 ties, and mSPT-IF has only 3 better solutions. In terms of the number of instances, the results of the overall and individual performances of mSPT-I are consistent with the results in Table 5, but it is observed that there is a difference in mSPT-FI. Table 5 shows that mSPT-FI performs better in 20/4/3, 20/10/3, 50/4/3 and 100/4/3, but in Table 6, mSPT-FI does not get more good solutions than mSPT-I. This shows that although the number of better instances for mSPT-FI is small, the good solutions are much better than those for mSPT-I, which allows the values in Table 5 to be relatively low. This shows that SPT-FI is very effective in solving certain sets of problems. In terms of execution time, SPT-FI outperforms SPT-I as shown in Table 5.

Then, the performance of mSPT-I/mSPT-FI in Stage 1 can be obtained by observing the performance of SPT. In Table 5, the STPs of 20/4/2 and 20/4/3 are 14.1% and 9.84%, while the STPs of 20/10/2 and 20/10/3 are 4.76% and 12.1% for a job number of 20. The results show that the SPT performance is better when there are fewer machines/higher GoS or more machines/lower GoS. In other words, it is implied that the performance of mSPT-I/mSPT-FI in Stage 2 is worse when the numbers of machines and GoS are in this particular condition.

Table 4. Relative performance of different algorithms for two-machine problems

	SPT		SPT-I/SPT-FI		EM	
n/m/G	MRPI%	CT	MRPI%	CT	MRPI%	CT
20/2/2	1.90	0.0008	0	0.0015	12.0	0.644
50/2/2	0.914	0.0015	0	0.0022	24.0	0.933
100/2/2	0.106	0.0015	0	0.0037	36.9	1.72
200/2/2	0.018	0.0022	0	0.0037	45.3	3.16
500/2/2	0.048	0.0045	0	0.0120	53.8	7.49
1000/2/2	0.012	0.0113	0	0.0370	58.4	16.5
Mean	0.500	0.0036	0	0.0100	38.4	5.08

Table 5. Relative performance of different algorithms for small-size problems

	SPT		mSPT-I		mSPT-FI		EM	
n/m/G	MRPI%	CT	MRPI%	CT	MRPI%	CT	MRPI%	CT
20/4/2	14.1	0.0013	**1.05**	0.0019	1.22	0.0015	2.21	0.753
20/4/3	9.84	0.0008	1.40	0.0017	**1.02**	0.0015	2.94	0.755
20/10/2	4.76	0.0009	**0.389**	0.0017	0.567	0.0009	0.147	0.654
20/10/3	12.1	0.0009	1.19	0.0015	**1.16**	0.0008	0.333	0.694
50/4/2	8.33	0.0008	**0.008**	0.0015	0.093	0.0013	11.7	1.26
50/4/3	6.55	0.0008	0.388	0.0015	**0.247**	0.0011	14.5	1.19
50/10/2	6.42	0.0013	**0.011**	0.0013	0.222	0.0009	6.77	1.29
50/10/3	8.00	0.0006	**0.277**	0.0009	0.374	0.0009	6.95	1.38
100/4/2	6.57	0.0036	**0.005**	0.0049	0.014	0.0032	21.1	2.20
100/4/3	4.60	0.0036	0.323	0.0042	**0.196**	0.0038	25.8	2.07
100/10/2	3.16	0.0028	**0.001**	0.0040	0.154	0.0036	16.9	2.49
100/10/3	6.99	0.0026	**0.206**	0.0034	0.388	0.0013	18.6	2.65
Mean	7.62	0.0017	**0.438**	0.0024	0.471	0.0017	10.7	1.45

Table 6. Comparison with mSPT-I and mSPT-FI for small-size problems

n/m/G	mSPT-I	Tie	mSPT-FI
20/4/2	33	44	3
20/4/3	23	38	19
20/10/2	17	58	5
20/10/3	26	38	16
50/4/2	35	40	5
50/4/3	34	31	15
50/10/2	28	44	8
50/10/3	43	16	21
100/4/2	24	42	14
100/4/3	28	28	24
100/10/2	35	40	5
100/10/3	48	15	17
Total	374	434	152

The same results are obtained for job numbers 50 and 100 as for job number 20. In addition, overall, the SPT performance is better as the job number increases.

In Table 5, most of the MRPI values for EM are higher than the proposed heuristics, and the values increase as the number of jobs increases. This result shows that the average performance of EM is worse than the average performance of the other algorithms, and the performance is even worse for large jobs. However, the MRPI of EM is the lowest of all the algorithms with 20 jobs and 10 machines, which means that the EM performance is the best performance under this condition. Overall, the average performance of mSPT-I/mSPT-FI is better than the average performance of EM. In terms of execution time, mSPT-I/mSPT-FI is also better than EM.

5.2.2 Large-size Job Experiment

As shown in Table 7, the average MRPI of mSPT-I and mSPT-FI are 0.119% and 0.191%, respectively. This means that the proposed mSPT-I and mSPT-FI have significantly better performance than SPT and EM. Compared with the results in Section 5.2.1, it can be found that mSPT-I and mSPT-FI have better performance in large-size jobs. Further, comparing the performance of mSPT-I and mSPT-FI in detail, mSPT-I has the best performance except for 200/4/3, 500/4/3 and 1000/4/3. The mSPT-FI has a greater advantage in solving the $m = 4$/GoS = 3 problem. Table 8 shows that the number of instances with better solutions for mSPT-FI does not always exceed the number of instances for mSPT-I. This result is also consistent with the conclusion in Section 5.2.1, which shows that mSPT-FI is effective in solving certain sets of problems. In terms of execution time, mSPT-FI outperforms mSPT-I as shown in Table 7.

For the performance of SPT, the results in Table 7 show the same results as in Section 5.2.1 that the SPT performance is better when there are fewer machines/higher GoS or more machines/lower GoS. Overall, the SPT performance is better as the number of jobs increases. For the performance of EM, the MRPI value of EM is much larger than the proposed algorithm, and the value increases as the number of jobs increases. This result proves that the performance of mSPT-I/mSPT-FI for large-size jobs is quite good and better than the average performance of EM. The mSPT-I/mSPT-FI is also very efficient in terms of execution time.

5.3 Discussion

The most related studies of $P_m|GoS|\sum C_j$ problem in this study are $P_m||\sum C_j$, $P_m|GoS|Cmax$ and $P_m|GoS|\sum W_jT_j$ problems as shown in Table 9. First, the comparison of the proposed heuristics and the SPT rule is discussed. $P_m||\sum C_j$ problem is not an NP-hard problem. The SPT rule can be used to obtain the optimal solution. (Conway et al., 1967; Pinedo, 2002). The SPT rule is also a dispatching rule, which is used in practice, for NP-hard scheduling problems. It can be used quickly and simply to obtain an acceptable solution, especially for different scheduling problems of $\sum C_j$criterion (Planinić et al., 2022).

Table 7. Relative performance of different algorithms for large-size problems

	SPT		mSPT-I		mSPT-FI		EM	
n/m/G	MRPI%	CT	MRPI%	CT	MRPI%	CT	MRPI%	CT
200/4/2	8.01	0.0023	**0.001**	0.0032	0.011	0.0024	29.6	3.99
200/4/3	4.97	0.0023	0.391	0.0038	**0.124**	0.0034	35.1	3.77
200/10/2	6.85	0.0028	**0.000**	0.0047	0.162	0.0026	28.1	4.97
200/10/3	12.7	0.0028	**0.357**	0.0053	0.544	0.0013	29.4	5.17
500/4/2	6.70	0.0062	**0.000**	0.0120	0.001	0.0058	40.0	9.24
500/4/3	4.07	0.0058	0.172	0.0133	**0.140**	0.0064	48.4	8.85
500/10/2	3.52	0.0062	**0.000**	0.0128	0.154	0.0054	40.9	12.0
500/10/3	9.68	0.0053	**0.195**	0.0139	0.364	0.0062	43.5	12.3
1000/4/2	7.71	0.0113	**0.000**	0.0338	0.000	0.0135	46.4	17.8
1000/4/3	4.02	0.0115	0.195	0.0432	**0.135**	0.0135	56.9	16.9
1000/10/2	3.00	0.0141	**0.000**	0.0377	0.181	0.0120	48.2	23.8
1000/10/3	9.89	0.0124	**0.111**	0.0428	0.477	0.0131	53.3	24.9
Mean	6.76	0.0070	**0.119**	0.0189	0.191	0.0071	41.6	12.0

Table 8. Comparison with mSPT-I and mSPT-FI for large-size problems

n/m/G	SPT-I	Tie	SPT-FI
200/4/2	36	37	7
200/4/3	23	26	31
200/10/2	31	43	6
200/10/3	44	15	21
500/4/2	23	40	17
500/4/3	32	27	21
500/10/2	34	42	4
50/010/3	55	6	19
1000/4/2	28	38	14
1000/4/3	31	27	22
1000/10/2	39	40	1
1000/10/3	59	8	13
Total	435	349	176

$P_m|GoS|\sum C_j$ problem is a NP-hard problem. Therefore, the SPT rule is applied in all proposed heuristics to obtain a good initial solution. Quick and effective methods are then developed to improve the initial solution. The structure of proposed heuristics is quick and simple, which is similar to the dispatching rule. The results show that the proposed heuristics outperform the SPT rule in all experiments. In terms of execution time, the FI-series heuristics and the SPT rule are the same. The average computation times in small-size and large-size problems are 0.0017 and 0.0071 sec, respectively. The proposed heuristics can be used quickly and simply to obtain acceptable solutions.

Next, similar scheduling problems and methods from past are discussed to explore the solution quality of the proposed heuristics. As shown in Table 9, the polynomial time approximation scheme (PTAS) is proposed to obtain the approximate solution for $P_m|GoS|Cmax$ (Ji and Cheng, 2008; Ou et al., 2008; Li and Zhang, 2009; Woeginger, 2009). PTAS cannot be applied to solve the considered problem ($P_m|GoS|\sum C_j$). EM, which is a metaheuristic, has been able to obtain very good performance in solving $P_m|GoS|\sum W_jT_j$ problem (Tseng et al., 2017). Metaheuristic is better than SPT rule, but is more complex and time-consuming (Xie et al., 2022). EM can be easily modified to solve $P_m|GoS|\sum C_j$. The results of modified EM and EM by Tseng et al. (2017) have the same pattern. When $m = 10$, they perform well. All proposed heuristics are compared with the modified EM (denoted EM) in this study. For multi-machine problem, the experiments were conducted on three job numbers (n =20, 50 and 100) and two machine numbers ($m = 4$ and 10). The results show that the proposed heuristics perform better than EM and require only the same execution time as the SPT rule. The average MRPIs of mSPT-I, mSPT-FI and EM are 0.438%, 0.471% and 10.7%, respectively.

Sensitivity analysis in scheduling problems is usually performed by changing two parameters (number of jobs or machines) to demonstrate the robustness of the proposed methods. It is performed by changing the parameter n (number of jobs). The experiments were conducted on three job numbers (n =200, 500 and 1000). The large-size job experiments have not been explored in previous studies. Three comparative experiments are detailed in Section 5.2.2: (1) mSPT-I and mSPT-FI; (2) mSPT-I/mSPT-FI and SPT; (3) mSPT-I/mSPT-FI and EM. The results show that all proposed heuristics are consistent with the conclusions in small-size job experiments. In addition, the performances of the proposed mSPT-I and mSPT-FI are significantly better than those of EM. Their execution times are similar to the SPT rule. The proposed heuristics can be used to converge to a very good solution by using a similar execution time for the SPT rule.

From the above discussion, we describe the comparison of the proposed heuristics with other methods and explain research gaps in Table 9. In summary, the considered problem in this study is a new NP-hard scheduling problem. The proposed heuristics have the quick and simple characteristics of the SPT rule and have the effect of solving the considered problem. Other methods may be too cumbersome for practical use, and heuristics provide faster solution times for near-optimal solutions (ReVelle et al., 2008; Zhao et al., 2019). The proposed heuristics demonstrate their effectiveness and efficiency and, therefore, their potential suitability for practical use.

6. CONCLUSIONS

This study investigates the $P_m|GoS|\sum C_j$ problem. To the best of our knowledge, this scheduling problem has not been addressed in the literature. Two heuristics, SPT-I and SPT-FI, are proposed to solve this problem based on a two-stage design for two machines and two service levels. Stage 1 applies the SPT rule to assign jobs to the machine. Stage 2 applies the insertion method to insert jobs from a particular machine to a specific machine to reduce the total completion time. The major difference between the two heuristics is that SPT-FI is an accelerated insertion.

Table 9. The most related studies of $P_m|GoS|\sum C_j$ problem

Problem	NP-hard	Method	Solution	Method characteristic
$P_m\|\|\sum C_j$	No	SPT rule	Optimal solution	Quick and simple
$P_m\|GoS\|Cmax$	Yes	PTAS	Approximate solution	Quick and complex
$P_m\|GoS\|\sum W_jT_j$	Yes	Metaheuristic	Near-optimal solution	Time consuming and complex
$P_m\|GoS\|\sum C_j$	Yes	The proposed heuristics	Near-optimal solution	Quick and simple

That is, after the insertion, it can be calculated using the derived equation without recalculating the total completion time to speed up the calculation time. Second, to extend the problem conditions to multiple machines and multiple service levels, two heuristics, mSPT-I and mSPT-FI, are proposed based on the above design. The insertion method of the two heuristics is different, as the former is based on a set of service levels and the latter is based on machines. In addition, mSPT-FI uses an accelerated insertion calculation.

All the proposed heuristics are compared with the EM algorithm in a series of experiments, which is an algorithm that has been proven to be effective in solving the $P_m|GoS|\sum W_jT_j$ problem. For the experiment with two-machine problem, the results show that the average performance of SPT-I/SPT-FI is better than the average performance of EM by 38.4%. In terms of execution time, SPT-I/SPT-FI also has a considerable advantage. In terms of the number of jobs 1000, SPT-I/SPT-FI takes only 0.037 sec, while EM takes 16.5 sec. For the experiment with multi-machine problem, the proposed mSPT-I and mSPT-FI have significantly better performance than EM. They are more efficient than the experiments with two-machine problem in terms of execution time. The performance of mSPT-I and mSPT-FI is further compared. In the small-size job experiment, mSPT-I performs better for the GoS = 2 problem, while mSPT-FI performs better for the GoS = 3 problem. In the large-size job experiment, mSPT-FI has better performance for the problem with $m = 4$/GoS = 3, and mSPT-I is superior for the remaining problems. Therefore, the proposed mSPT-I and mSPT-FI can support each other's problems with different sizes of jobs and $P_m|GoS|\sum C_j$ can be solved efficiently. Therefore, the proposed heuristics are effective and potentially suitable for practical use.

From the perspective of research limitations in Section 1, scheduling criterion, and solution methods, future research directions can further explore the following: (1) Relaxing the limits of processing time becomes a stochastic scheduling problem; (2) Online scheduling problem is considered and on-line heuristic is developed, that is, the jobs come in at different times and randomly; (3) It is also worthwhile to investigate other types of parallel machines under the same constraint and criterion, e.g., unrelated parallel machine; (4) The preemptions are allowed; (5) The scheduling criterion of total weighted completion time can further be investigated; (6) The proposed heuristics may be applied in a metaheuristic as a good initial solution for other similar scheduling problems.

REFERENCES

Akbar, M., Irohara, T. 2018. Scheduling for sustainable manufacturing: A review. Journal of Cleaner Production, 205, 866–833.

Bektur, G., Saraç, T. 2019. 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. Computers & Operations Research, 103, 46–63.

Berthier, A., Yalaoui, A., Chehade, H., Yalaoui, F., Amodeo, L., Bouillot, C. 2022. Unrelated parallel machines scheduling with dependent setup times in textile industry. Computers & Industrial Engineering, 174, 108736.

Conway, R.W., Maxwell, W.L., Miller, L.W. 1967. Theory of scheduling. Addison-Wesley Reading. Massachusetts. U.S.A.

Epstein, L., Levin, A. 2011. Scheduling with processing set restrictions: PTAS results for several variants. International Journal of Production Economics, 133, 586–595.

Glass, C.A., Kellerer, H. 2007. Parallel machine scheduling with job assignment restrictions. Naval Research Logistics, 54, 250–257.

Huo, Y., Leung, J.Y.T., Wang, X. 2009. A fast preemptive scheduling algorithm with release times and inclusive processing set restrictions. Discrete Optimization, 6, 292–298.

Hwang, H.C., Chang, S.Y., Lee, K. 2004. Parallel machine scheduling under a grade of service provision. Computers & Operations Research, 31, 2055–2061.

Ji, M., Cheng, T.C.E. 2008. An FPTAS for parallel-machine scheduling under a grade of service provision to minimize makespan. Information Processing Letters, 108, 171–174.

Ji, M., Ye, X., Qian, F., Cheng, T.C.E., Jiang, Y. 2022. Parallel-machine scheduling in shared manufacturing. Journal of Industrial and Management Optimization, 18, 681–691.

Kusoncum, C., Sethanan, K., Pitakaso, R., Hartl, R.F. 2021. Heuristics with novel approaches for cyclical multiple parallel machine scheduling in sugarcane unloading systems. International Journal of Production Research, 59, 2479–2497.

Leung, J.Y.T., Li, C.L. 2008. Scheduling with processing set restrictions: A survey. International Journal of Production Economics, 116, 251–262.

Leung, J.Y.T., Li, C.L. 2016. Scheduling with processing set restrictions: A literature update. International Journal of Production Economics, 175, 1–11.

Leung, J.Y.T., Ng, C.T. 2017. Fast approximation algorithms for uniform machine scheduling with processing set restrictions. European Journal of Operational Research, 260, 507–513.

Li, S. 2017. Parallel batch scheduling with inclusive processing set restrictions and non-identical capacities to minimize makespan. European Journal of Operational Research, 260, 12–20.

Li, W., Li, J., Zhang, T. 2009. Approximation schemes for scheduling on parallel machines with GoS levels. Lecture Notes in Operations Research and Its Applications, 10, 53–60.

Li, C.L., Wang, X. 2010. Scheduling parallel machines with inclusive processing set restrictions and job release times.

European Journal of Operational Research, 200, 702–710.

Li, C.L., Li, Q. 2015. Scheduling jobs with release dates, equal processing times, and inclusive processing set restrictions. Journal of the Operational Research Society, 66, 516–523.

Li, D., Wang, J., Qiang, R., Chiong, R. 2021. A hybrid differential evolution algorithm for parallel machine scheduling of lace dyeing considering colour families, sequence-dependent setup and machine eligibility. International Journal of Production Research, 59, 2722–2738.

Liao, B., Song, Q., Pei, J., Yang, S., Pardalos, P.M. 2020. Parallel-machine group scheduling with inclusive processing set restrictions, outsourcing option and serial-batching under the effect of step-deterioration. Journal of Global Optimization, 78, 717–742.

Lim, K. 2010. Parallel machines scheduling with GoS eligibility constraints: A survey. Journal of the Korean Institute of Industrial Engineers, 36, 248–254.

Liu, M., Yang, X., Chu, F., Zhang, J., Chu, C. 2020. Energy-oriented bi-objective optimization for the tempered glass scheduling. Omega, 90, 101995.

Maecker, S., Shen, L., Mönch, L. 2023. Unrelated parallel machine scheduling with eligibility constraints and delivery times to minimize total weighted tardiness. Computers & Operations Research, 149, 105999.

Mateo, M., Teghem, J., Tuyttens, D. 2018. A bi-objective parallel machine problem with eligibility, release dates and delivery times of the jobs. International Journal of Production Research, 56, 1030–1053.

Mecler, D., Abu-Marrul, V., Martinelli, R., Hoff, A. 2022. Iterated greedy algorithms for a complex parallel machine scheduling problem. European Journal of Operational Research, 300, 545–560.

Mokotoff, E. 2001. Parallel machine scheduling problems: A survey. Asia-Pacific Journal of Operational Research, 18, 193–242.

Ou, J., Leung, J.Y.T., Li, C.L. 2008. Scheduling parallel machines with inclusive processing set restrictions. Naval Research Logistics, 55, 328–338.

Ou, J., Zhong, X., Qi, X. 2016. Scheduling parallel machines with inclusive processing set restrictions and job rejection. Naval Research Logistics, 63, 667–681.

Pinedo, M. 2002. Scheduling: Theory, algorithm, and system. Second Ed. Prentice-Hall Press. New Jersey. U.S.A.

Planinić, L., Backović, H., Đurasević, M., Jakobović, D. 2022. A comparative study of dispatching rule representations in evolutionary algorithms for the dynamic unrelated machines environment. IEEE Access, 10, 22886–22901.

ReVelle, C., Scholssberg, M., Williams, J. 2008. Solving the maximal covering location problem with heuristic concentration. Computers & Operations Research, 35, 427–435.

Tseng, C.T., Lee, C.H., Chiu, Y.S.P., Lu, W.T. 2017. A discrete electromagnetism-like mechanism for parallel machine scheduling under a grade of service provision. International Journal of Production Research, 55, 3149–3163.

Woeginger, G.J. 2009. A comment on parallel-machine scheduling under a grade of service provision to minimize makespan. Information Processing Letters, 109, 341–342.

Xie, Y., Sheng, Y., Qiu, M., Gui, F. 2022. An adaptive decoding biased random key genetic algorithm for cloud workflow scheduling. Engineering Applications of Artificial Intelligence, 112, 104879.

Zhao, G., Liu, J., Tang, L., Zhao, R., Dong, Y. 2019. Model and heuristic solutions for the multiple double-load crane scheduling problem in slab yards. IEEE Transactions on Automation Science and Engineering, 17, 1307–1319.