Two-stage no-wait hybrid flow shop with inter-stage flexibility for operating room scheduling

Mohamed-Naceur Azaiez, Anis Gharbi, Imed Kacem, Yosra Makhlouf, Malek Masmoudi

Title Page

Title:

Two-stage no-wait hybrid flow shop with inter-stage flexibility for operating room scheduling

Author names and affiliations:

Mohamed Naceur AZAIEZ

Business Analytics and DEcision Making Laboratory (BADEM)

Tunis Business School, Université de Tunis

Tunis 2059, Tunisia

Anis GHARBI

Department of Industrial Engineering

King Saud University

Riyadh 11421, Saudi Arabia

Imed Kacem

Université de Lorraine, LCOMS

3, Rue Augustin Fresnel, Metz, France

imed.kacem@univ-lorraine.fr

Yosra Makhlouf (corresponding author)

Université de Lorraine, LCOMS

3, Rue Augustin Fresnel, Metz, France

yosra.makhlouf@univ-lorraine.fr

Malek MASMOUDI

University of Sharjah, College of Engineering, United Arab Emirates

Université Jean Monnet Saint-Etienne, 42000 Saint Étienne, France

Declarations of interest: none

Two-stage no-wait hybrid flow shop with inter-stage flexibility for operating room scheduling

Abstract

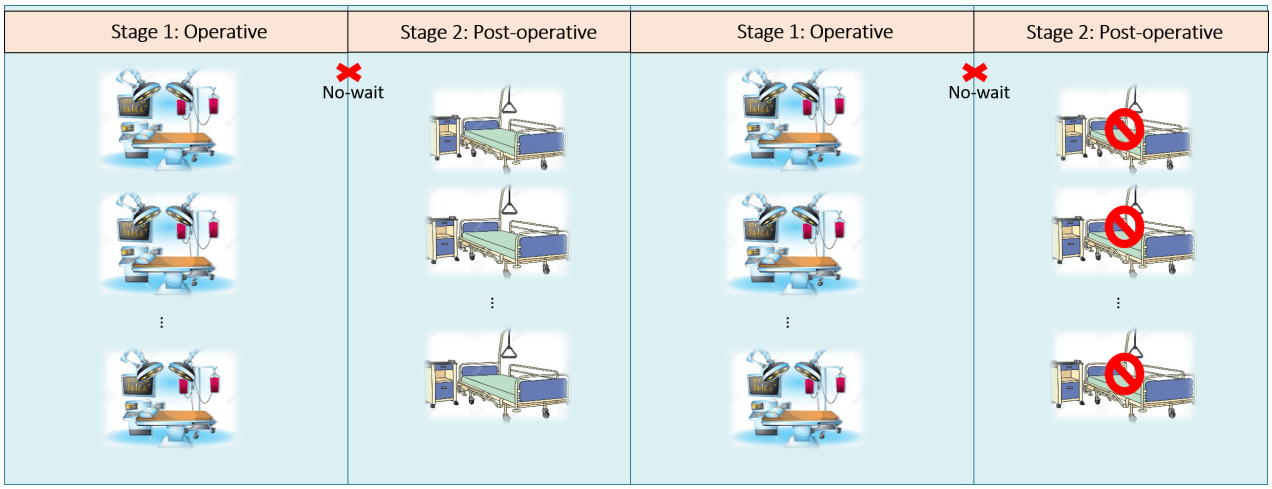
Operating rooms are amongst the most critical resources in hospitals. Appropriate schedules of surgical interventions increase the surgeries' success rates. Indeed, surgery outcomes strongly depend on the timing of each step in the surgery process. Therefore, effective and efficient surgery schedules can ease patients' suffering and even save their lives while making good use of limited hospital resources. This paper studies the two-stage no-wait hybrid flow shop scheduling problem with inter-stage flexibility. The problem is inspired from hospital operating room scheduling under limited healthcare resources. We propose a time-indexed mixed integer linear programming formulation of the problem. We also introduce valid inequalities along with four lower bounds and four heuristics to handle the large scale of the problem. The proposed model is tested on randomly generated instances based on realistic data for operating room scheduling. Experimental results on the performance of the model and comparisons among the lower bounds and heuristics are reported for the different sizes of instance classes.

Keywords: hybrid flow shop, mixed integer programming, time-indexed formulation, lower bounds, heuristics, valid inequalities

1. Introduction

Hybrid flow shop scheduling is a generalization and combination of parallel machines and classical flow shops. In classical flow shop scheduling, machines are placed in a series configuration with one machine in each stage. Jobs follow the same order of passage and have to be processed in a sequential fashion, one stage after another. There is only one machine in each stage. In hybrid flow shops, however, at least one stage contains two or more parallel machines. In all cases, an operation of a job can only be processed by a single machine at a time and at each stage. The problem is two-fold : machine assignment and job sequencing. The machine assignment part consists in deciding, for each job at each stage, which machine will process the operation. The sequencing part corresponds to determining the starting and ending times of the operations on the assigned machines. Both decisions may be made simultaneously in an integrative manner or separately in a hierarchical (sequential) approach.

Hybrid flow shops are also known as flexible flow shops. The concept includes the feature of the machine assignment flexibility introduced by such a configuration compared to a classical flow shop. Many researchers studied the hybrid flow shop scheduling problem with and without additional constraints.



(a) In normal situations, recovery takes place immediately after surgery in a recovery bed

(b) In case no recovery bed is available, recovery starts immediately after surgery in the operating room

Figure 1: Illustration of recovery in operating rooms when no recovery beds are available

Interested readers are referred to the reviews in [1], [2], and [3], and more recently, the one in [4].

1.1. The paper scope

In this paper, we focus on the hybrid flow-shop with no-wait variant of the problem. Specifically, we investigate the general version of the two-stage no-wait hybrid flow shop with inter-stage flexibility. Indeed, in the operating room scheduling context, a patient may be considered as a job requiring two operations; namely, surgery, then recovery. A stage-1 machine represents an operating room, and a stage-2 machine corresponds to a recovery bed. Inter-stage flexibility refers to the fact that the second operation can take place on a machine of either the first or the second stage. Actually, when no recovery beds are available, recovery may take place at the operating room itself as modeled by [5] and illustrated in Figure 1. Thus, our problem is closely related to the flow-shop scheduling problem with no-wait and multi-task flexibility.

1.2. Literature review

In [6], the authors consider the problem of makespan minimization in a special case of a two-stage no-wait hybrid flow shop (where the first stage operation is divided into a compulsory suboperation and an optional suboperation with the latter only taking place if no second stage machine is available). The work is motivated by the operating room scheduling problem first considered by [5] where patient recovery after surgery is allowed to start in the operating room if no recovery bed is available. Indeed, immediately after surgery, the patient must fully recover from anesthesia and its lingering effects. The recovery unit, also called Post-Anesthesia Care Unit, has a limited number of identical beds, where the patient's recovery normally occurs. The equipment needed for post-anesthesia recovery is also available in the operating room. Thus, recovery can take place in the operating room if recovery beds are all occupied. This scenario

has also been discussed more recently in [7] and [8]. More generally, recent works on operating room scheduling can be found in [9], [10], and [11].

The work in [6] proves that the problem is strongly NP-hard. The solution method accounts for developing constant factor approximation algorithms for the variants with only one machine in either stage 1 or stage 2. The proposed algorithms are based on greedy list scheduling. The authors present two approximation algorithms for the case where stage one has only one machine and stage two has $m \geq 2$ machines with approximation ratios $3 - \frac{2}{m+1}$ and $2 - \frac{1}{m+1}$, and a 2-approximation algorithm for the problem with one machine at stage two and $m \geq 2$ machines at stage one. The first algorithm is a list scheduling algorithm modified to account for inter-stage flexibility. The second algorithm one is similar, except that the jobs are sequenced according to the Longest Processing Time (LPT) rule. More precisely, in non-increasing order of their second operations' processing times. Only worst-case analyses are conducted without experimental testing. The variant with multiple machines in both stages is however not investigated.

The work in [12] investigates the complexity of the no-wait shop scheduling problems and proves that the no-wait two-stage hybrid flow shop problem with one machine on stage 1 and two parallel machines on stage 2 is strongly NP-hard for makespan minimization. The work in [13] presents approximation algorithms for the two-stage no-wait hybrid flow shop with makespan criterion and performs worst-case analyses for various cases of the problem. In [14], the authors study the problem of two-stage no-wait hybrid flow shop scheduling with a single machine on the first stage and several identical parallel machines on the second. They prove that the problem is strongly NP-hard and calculate lower bounds, dominance rules, and suggest heuristics, which they feed to a branch-and-bound algorithm. They also offer a mixed integer programming formulation of the problem and perform experiments on three different randomly generated problem classes. Comparison between MIP and branch-and-bound shows that performance varies from a class of problems to another.

In [15], the authors address a two-machine flow shop problem with task flexibility, also called alternative operations, where at least one of the two machines can perform both types of tasks. They prove the problem to be NP-hard and develop a branch-and bound technique to provide optimal solutions. In [16], the authors also study a two-machine flow shop problem with task flexibility and focus on the case where the first task can be processed by either the first or second machine. They present a fully-polynomial time approximation scheme for the case with infinite buffer capacity, and optimal algorithms for the case with identical jobs and finite buffer capacity, including when the buffer capacity is null. [7] investigates the general case of the no-wait two-stage flowshop with multi-task flexibility of the first machine and develop an approximation algorithm with a worst-case performance of 13/8. However,the considered model reduces to the case of a single machine on each stage in which the task that can be processed on either machine is the first task. This situation can be seen as the mirror problem of a relaxed version of our problem.

Many methods for solving the hybrid flow shop problem and its extensions are presented in the literature.

Solution approaches are classified into exact and approximate approaches. Many of these approaches such as branch-and-bound and branch-and-cut require solving a mathematical model of the problem. To formulate a mathematical model, mixed integer linear programming is commonly used in the scheduling literature. In [17], the authors use mathematical programming for hybrid flow shop scheduling with a makespan minimization criterion. They classify the solution representations existing in the literature into three groups and suggest a fourth class. They offer a mixed integer programming model for each representation. The classification is based on two axes, namely job sequencing rules and machine assignment rules. Therefore, the resulting four mixed integer linear programming do not necessarily yield optimal solutions. It is noteworthy that all four modeling classes representing the most common approaches for hybrid flow shop scheduling in the literature use continuous-time variables.

To the best of our knowledge, using time-indexed variables in mathematical models is almost non-existent in hybrid flow shop scheduling literature. The only exception constitutes the work in [18] where the authors develop a time-indexed model and valid inequalities for a variant of the hybrid flow shop scheduling problem with additional constraints. In [19], Bowman was the first to suggest a discrete time model for a scheduling problem, namely the job shop scheduling problem. The author uses 0-1 decision variables to determine whether or not a job is being processed at a given time period. The formulation was deemed inefficient as it requires a consequent number of variables and constraints. In [20], Pritsker, Waiters and Wolfe (PWW) modify the model in the context of project scheduling. They suggest another time-indexed formulation, where 0-1 decision variables indicate whether a job is completed in a specific time period or not. Although more compact, PWW [20] formulation would be less practical than Bowman [19] formulation in the case of operation splitting, as pointed out in [21]. Always in the context of project scheduling, [22] reuse [20]'s formulation and embed it in a minimum bounding algorithm. The algorithm starts from a lower bound of the makespan then increments the lower bound by one at a time until the time-indexed formulation proves feasible. We use a similar technique in our approach. [23] relies on Bowman [19] formulation as a basis for the introduced simulation model for a job shop scheduling case with additional considerations. In [24], the formulation by [20] is extended in a goal programming approach for job shop scheduling. In [25], yet another variable definition is attempted where the time-indexed variable determines whether or not a job starts at a specific time period. Based on the latter decision variable definition, [26] suggests a time-indexed formulation for single machine scheduling and proposes several valid inequalities to strengthen it. [27] adds to Bowman [19] formulation a secondary decision variable to express whether or not a job is completed at a given time period. Other types of additional time-indexed variables are presented in [28] as part of a mathematical model for production planning and scheduling in general. Later on, discrete-time formulations are adapted to the batch scheduling problem in chemical process engineering (e.g. the review in [29]). The Bowman [19] formulation is regarded as weak because it requires a great number of variables, especially with real-size problems where the number of time periods can become consequent. However, [30] empirically shows that models involving a large number of binary

variables perform better than smaller ones in terms of the computational time required to yield an optimal solution. In [31], the authors focus on mathematical modelling of flexible job-shop scheduling problems. They present a mathematical model based on time-indexed four-dimensional variables.

1.3. Contributions

This paper studies the generalized version of two-stage no-wait hybrid flow shop scheduling problem with inter-stage flexibility. This problem is scarcely investigated in the literature except for the works in [6] and in [32]. However, [32] investigated the variant where there is only one machine in each stage. In other words, the environment is a classical flow shop and not a hybrid flow shop with multiple machines at each stage. [6] treated the variants where there is only one machine in either the first or second stage. In this paper, we consider the general version where there are multiple machines on both stages. The fact that there are multiple machines at each stage is a complicating factor when it comes to solving the problem. Indeed, with only one machine per stage, the only decision that is required is the sequencing decision. The solution representation can be given by the sequence of jobs on the only machine of the first stage. Whereas in the case with multiple machines at each stage, assignment decisions are required in addition to the sequencing decisions in order to determine which machine processes which job. To the best of our knowledge, this is the first attempt to investigate the general case of the problem variant with no-wait and flexibility and with multiple machines in each stage.

Our approach includes the development of a mixed integer linear programming formulation as well as heuristics. The proposed mixed integer programming model is the first mathematical formulation for this variant of the problem. Although there is a plethora of mathematical models for hybrid flow shop scheduling in the literature, they are all continuous models with big-M constraints. Our approach uses a time-indexed formulation in order to avoid the big-M method. We also develop valid inequalities to strengthen the model. We propose heuristics to provide solutions for large instances. Further, we conduct numerical testing on realistic data and provide related discussions.

The remaining of the paper is organized as follows. In Section 2, we explain the methodology of our solution technique. First, we provide a detailed description of the problem and formulate the mixed integer linear model. Then, we propose lower bounds, valid inequalities, and heuristics. In Section 3, we report experimental testing and results. In section 4, we offer a thorough discussion about our findings. Section 5 serves for final conclusions and a brief outline of future research.

2. Mathematical Model

We start by providing a detailed problem statement. Then, we offer a corresponding formulation through a mathematical programming model.

2.1. Problem description

There is a set J of n jobs to be processed in a two-stage hybrid flow-shop environment. Each job $j \in J$ has two operations O_{1j} and O_{2j}. The hybrid flow shop has m_1 parallel identical machines at stage 1 and m_2 parallel identical machines at stage 2.

Processing requirements are as follows.

The first operation of a job must be entirely processed for p_{1j} time by exactly one machine at stage 1.

The second operation is processed for p_{2j} time following one of three possible patterns.

- Pattern 1 : Operation is entirely executed by exactly one machine on stage 2
- Pattern 2 : Operation is entirely executed by the same stage-1 machine which processed the first operation of the same job.
- Pattern 3 : Execution starts on the same stage-1 machine which processed operation 1, then continues on exactly one stage-2 machine.

No part of operation 2 can be executed at stage 1 while a stage-2 machine is free.

No preemption is allowed at either stage. Waiting between stage 1 and stage 2 is not allowed either. In other terms, once a job starts execution, no interruptions are permitted until the job exits the system.

In the second and third patterns, after finishing operation 1, the job stays on the stage-1 machine for the processing of operation 2. In pattern 2, no stage-2 machine becomes available so the stage-1 machine completely finishes processing operation 2. Then, the job exits the system without passing on any stage-2 machine. In pattern 3, the stage-1 machine starts processing operation 2 until a stage-2 machine becomes available, so the job is immediately transferred to that stage-2 machine to process the remaining part of operation 2.

We suppose that transfer times are negligible.

The objective is to minimize the maximum completion time; i.e., the makespan.

Using the tertiary notation, the problem is denoted by $F2(Pm_1, Pm_2 - miss)|no-wait, flex(1 \to 2)|C_{max}$. $F2(P_{m_1}, P_{m_2})$ refers to the scheduling environment as a two-stage flow shop with m_1 and m_2 parallel identical machines on stage 1 and stage 2, respectively.

2.2. Notations

J : set of jobs, indexed by j

I : set of stages, indexed by i

M_i : set of resources on stage i, indexed by k

T : set of time periods, indexed by t

p_{ij} : processing time of operation i for job j

2.3. Integer Linear Model formulation

The planning horizon is subdivided into time slots or periods, which reduces complexity and facilitates the modeling, making for a discrete-time model.

2.3.1. Decision variables

$A^k_{ij,t} = 1$ if machine k of stage i is active processing job j at time t, 0 otherwise

$B1^k_{j,t} = 1$ if at time t machine k of stage 1 continues processing job j after p_{1j} units have already been processed, 0 otherwise.

C_{max} = the makespan

2.3.2. Formulation

$$Minimize\ C_{max} \tag{1}$$

subject to

$$\sum_{j\in J} A^k_{ij,t} \leq 1\ (i \in I, t \in T, k \in M_i) \tag{2}$$

$$\sum_{k'\in M_i : k' \neq k} A^{k'}_{ij,t'} \leq 1 - A^k_{ij,t}\ (i \in I, j \in J, t \in T, t' \in T, k \in M_i) \tag{3}$$

$$\sum_{k\in M_1} A^k_{1j,t} \leq 1 - \sum_{l\in M_2} A^l_{2j,t'}\ (j \in J, t \in T, t' \in T : t' \leq t) \tag{4}$$

$$\sum_{t\in T}\sum_{k\in M_1} A^k_{1j,t} \geq p_{1j}\ (j \in J) \tag{5}$$

$$\sum_{t\in T}\sum_{k\in M_1} A^k_{1j,t} \leq p_{1j} + p_{2j}\ (j \in J) \tag{6}$$

$$\sum_{t\in T}(\sum_{k\in M_1} A^k_{1j,t} + \sum_{l\in M_2} A^l_{2j,t}) = p_{1j} + p_{2j}\ (j \in J) \tag{7}$$

$$B1^k_{j,t} \geq A^k_{1j,t} + A^k_{1j,t-p_{1j}} - 1\ (j \in J, t = p_{1j}+1, .., T, k \in M_1) \tag{8}$$

$$B1^k_{j,t} \leq \frac{1}{2}(A^k_{1j,t} + A^k_{1j,t-p_{1j}})\ (j \in J, t = p_{1j}+1, .., |T|, k \in M_1) \tag{9}$$

$$\sum_{k\in M_1} B1^k_{j,t} \leq \sum_{j\in J} A^l_{2j,t}\ (j \in J, t = p_{1j}+1, .., |T|, l \in M_2) \tag{10}$$

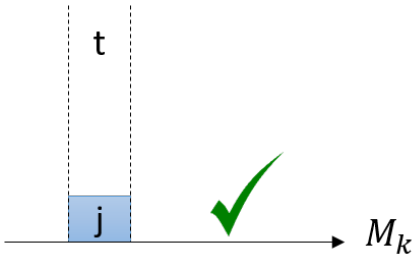
$$\sum_{k\in M_1} A^k_{1j,t} \leq 1 - \sum_{i\in I}\sum_{k_i\in M_i} A^{k_i}_{ij,t'}\ (j \in J, t \in T, t' \in T : t' \geq t + p_{1j} + p_{2j}) \tag{11}$$

$$C_{max} \geq t \times \sum_{k_i \in M_i} A_{ij,t}^{k_i} \ (i \in I, j \in J, t \in T) \tag{12}$$

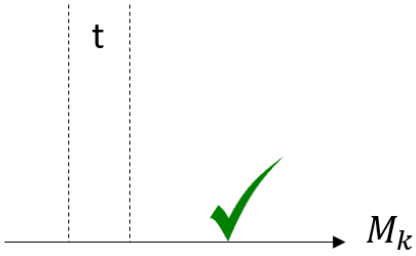
$$A_{ij,t}^{k} \in \{0,1\} \ (i \in I, j \in J, t \in T, k \in M_i) \tag{13}$$

$$B1_{j,t}^{k} \in \{0,1\} \ (j \in J, k \in M_1, t = p_{1j} + 1, .., |T|) \tag{14}$$

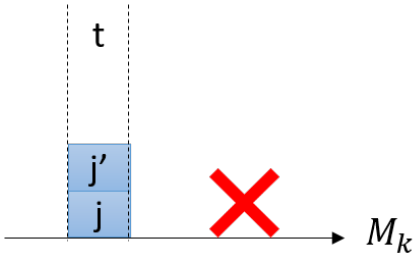
- Constraints 2 ensure that a machine processes one job at a time. (Figure 2)



(a) At time t, machine k can be busy with one job j



(b) At time t, machine k can be idle



(c) At time t, machine k cannot be busy with two jobs j and j' simultaneously

Figure 2: A machine processes one job at a time.

180

- Constraints 3 ensure that a job is processed by only one machine on each stage.(Figure 3)

- Constraints 4 prevent any piece of operation 2 from being processed before a piece of operation 1, i.e., once a unit of job j is executed at stage 2, job j cannot be executed at stage 1 anymore.(Figure 4)

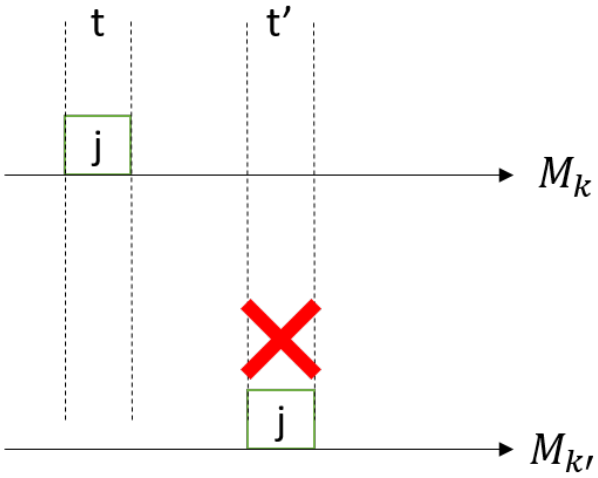


Figure 3: If a machine k processes job j at time t, no other machine k' of the same stage can process the same job j at any time t'

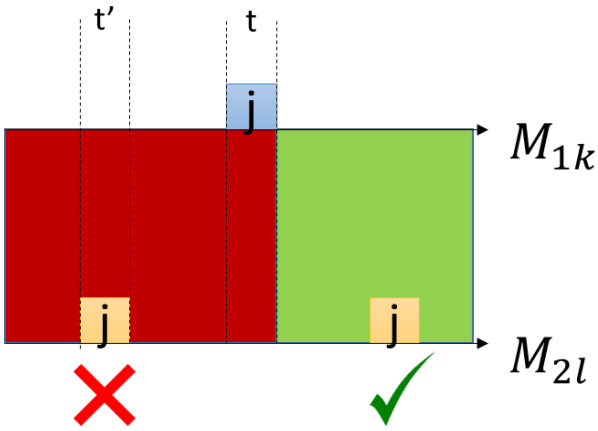


Figure 4: If a stage 1 machine k processes job j at time t', no stage 2 machine l can process the same job j at time earlier than t'. The red area specifies the time space where execution of job j on stage 2 is prohibited.

- The processing durations of job j are bounded in constraint groups 5-7 as follows. The job is executed at stage 1 for at least p_{1j} and at most $p_{1j} + p_{2j}$ units of time (Figure 5a). The total processing time is exactly equal to the total time requirements on both stages (Figure 5b).

- In constraints 8-9, machine k of stage 1 is active processing job j while p_{1j} units of job j have already been processed on the same machine k if and only if machine k is active both at time t and at time $t - p_{1j}$ (Figure 6)

- Constraints 10 verify that once p_{1j} units of job j have been processed on a stage 1 machine k, execution at stage 1 cannot continue at time t if at least one stage 2 machine is free at time t. (Figure 7)

- Constraints 11 ensure contiguity and no-wait : if a stage 1 machine k processes a unit of job j at

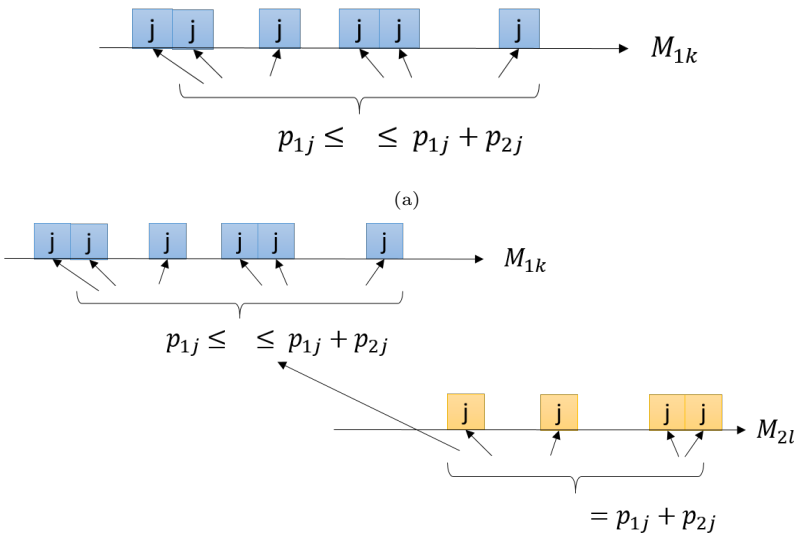


Figure 5: Processing time requirements

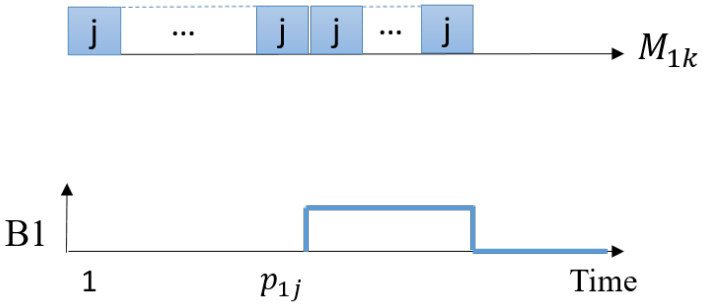


Figure 6: For each job j, starting from time $p_{1j}+1$, $B1^k_{j,t}$ is activated if the corresponding $A^k_{1j,t}$ is active. $B1$ gets deactivated as soon as as $A^k_{1j,t}$ becomes 0

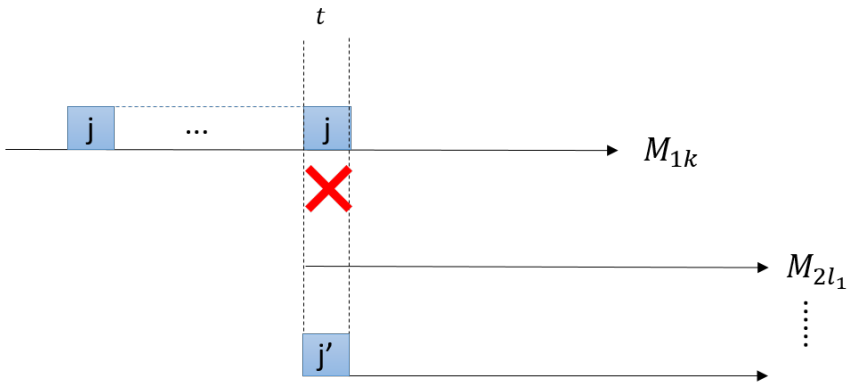


Figure 7: At time t, stage 2 machine M_{2l_1} is free. Therefore, execution of a unit of p_{2j} of job j is not allowed to continue on stage 1.

time t, no other units of the job can be executed after time $t + p_{1j} + p_{2j}$.(Figure 8) Since $p_{1j} + p_{2j}$

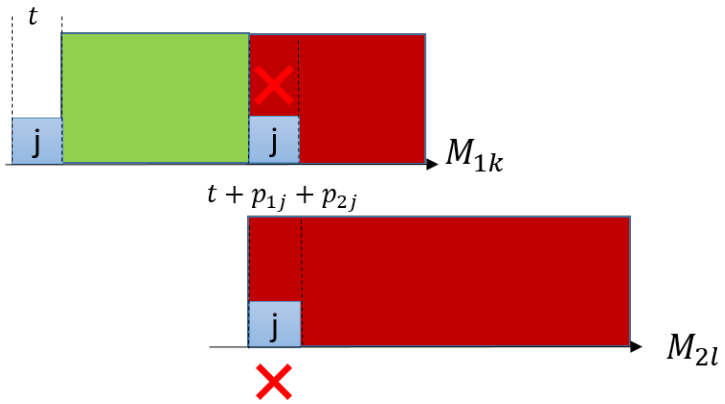


Figure 8

195 units must be executed, this constraint forces the units of the same job to be processed contiguously and without any interruptions.

- The makespan is calculated in constraints 12 as the latest time a machine is busy.
- Finally, 13-14 are the binary-variable constraints.

2.4. Lower bounds

200 #### *2.4.1. Lower bound 1*

A first trivial lower bound is

$$LB_1 = \max_{j \in J}\{p_{1j} + p_{2j}\} \tag{15}$$

2.4.2. Lower bound 2

Another lower bound is given by

$$LB_2 = \left\lceil \frac{1}{m_1} \sum_{j \in J} p_{1j} \right\rceil \tag{16}$$

2.4.3. Lower bound 3

We relax our problem as a parallel machine scheduling problem with $m_1 + m_2$ parallel identical machines and release dates.

We suppose w.l.o.g that the jobs are sorted in non-decreasing order of p_{1j}, i.e. $p_{11} \leq p_{12} \leq ... \leq p_{1N}$. Then the following is a lower bound.

$$LB_3 = \left\lceil \frac{1}{m_1 + m_2} (\sum_{j=1}^{m_2} p_{1j} + \sum_{j \in J} (p_{1j} + p_{2j})) \right\rceil \tag{17}$$

The proof is inspired from Carlier's lower bound in [33] which is based on summing the times of inactivity and the times of activity on a machine. In our case, the first m_1 machines are active at earliest at time 0. For the remaining m_2 machines, the first machine is necessarily idle from time 0 to time p_{11} since the operation must be executed on one of the first m_1 machines during the interval $[0, p_{11}]$. Similarly, the second stage-2 machine is idle from time 0 to time p_{12}, and the lth machine from time 0 to time p_{1l}. Thus, the periods of inactivity are at least equal to the m_2 smallest p_{1j} and the result follows.

2.4.4. Lower bound 4

Consider the problem $Pm|p_j, q_j|L_{max}$, denoted (ΠL), where :

p_j is the processing time of job j

q_j is the delivery time (or tail) of job j, i.e. the time spanning from when job j is completed at time C_j until when it exits the system.

$L_{max} = \max_{j \in J}\{L_j\}$ is the maximum lateness, $L_j = C_j + q_j$.

Suppose w.l.o.g that the jobs are sorted according to non-increasing order of their delivery times q_j, i.e. $q_1 \geq q_2 \geq ... \geq q_N$ (Jackson's rule). A lower bound to problem (ΠL) is given by :

$$\left\lceil \frac{1}{m}(\sum_{j \in J} p_j + \sum_{j=N}^{N-m+1} q_j) \right\rceil \tag{18}$$

If we choose $m = m_1$, $p_j = p_{1j}$ and $q_j = p_{2j}$, the maximum lateness in (ΠL) is equal to the makespan in our problem. Therefore, the lower bound is applicable.

$$LB_4 = \left\lceil \frac{1}{m_1}(\sum_{j \in J} p_{1j} + \sum_{j=N}^{N-m_1+1} p_{2j}) \right\rceil \tag{19}$$

2.5. Valid inequalities

Two types of valid inequalities are added as cuts in an attempt to strengthen the model.

2.5.1. Valid inequality 1

A feasible solution for our problem, denoted by (P), is a feasible solution for the problem $P_m|pmtn|\sum C_j$, denoted by ($\Pi 0$), of preemptively scheduling n jobs on $m = m_1 + m_2$ machines, where each job j has a processing time $p_j = p_{1j} + p_{2j}$.

Denote by C_j the final completion time of job j.

$$\sum_{j \in J} C_j(P) \geq \sum_{j \in J} C_j^*(\Pi 0) \tag{20}$$

[34] showed that preemption only worsens the mean average completion time. Thus, the optimal schedule for $P_m|r_j; pmtn|\sum C_j$ is non-preemptive.

In particular, if $r_j = 0 \ \forall \ j$, the problem $P||\sum C_j$ is polynomially solvable by applying the Shortest

Processing Time (SPT) rule and assigning the jobs according to the First Available Machine (FAM) rule. ([35])

$$\sum_{j\in J} C_j^*(\Pi 0) \geq \sum_{j\in J} C_j^{SPT}(\Pi 0) \tag{21}$$

where $C_j^*(\Pi 0)$ denotes the ending time of job j in the optimal solution of problem ($\Pi 0$) and $C_j^{SPT}(\Pi 0)$ is the ending time of job j in the solution given by applying the SPT rule to problem ($\Pi 0$).

$$C_j(P) = \max_{t\in T}\{t \times A_{ij,t}^k\}(i \in I, j \in J, k \in M_i) \tag{22}$$

The following are valid inequalities.

$$\sum_{j\in J} C_j(P) \geq \sum_{j\in J} C_j^{SPT}(\Pi 0) \tag{23}$$

$$C_j(P) \geq t \times A_{ij,t}^k(i \in I, j \in J, k \in M_i) \tag{24}$$

Consider the numerical example in Table 1 for illustration.

$n = 5;\ m_1 = 2;\ m_2 = 1;\ p_{1j} = \{1, 3, 4, 2, 5\};\ p_{2j} = \{5, 3, 2, 1, 4\};\ T = 15$

Table 1: Small illustrative example

j	1	2	3	4	5	
p_{1j}	1	3	4	2	5	
p_{2j}	5	3	2	1	4	
p_j	6	6	6	3	9	
SPT rank	2	3	4	1	5	sum
$C_j^{SPT}(\Pi 0)$	6	6	9	3	15	39
C_j^*(P)	6	12	6	9	10	43

2.5.2. Valid inequality 2

Proposition.

$$p_j C_{max} \geq \sum_{t\in T}(t - \frac{1}{2}) \times (\sum_{k\in M_1} A_{1j,t}^k + \sum_{l\in M_2} A_{2j,t}^l) + \frac{1}{2}p_j^2 \tag{25}$$

where $p_j = p_{1j} + p_{2j}$

Proof. The execution of a job j in the time-indexed model can be illustrated as follows, by a set of contiguous unit-length chunks. The distance between the first and last chunks is $p_j = p_{1j} + p_{2j}$.

Let S_j be the starting time of job j; the ending time is $C_j = S_j + p_j$

Denote $X = \sum_{t=1}^{p_j} S_j + t - \frac{1}{2}$, as shows Figure 9

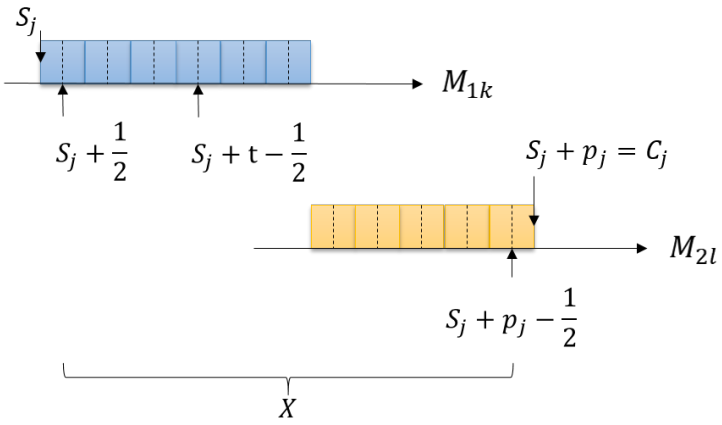


Figure 9: Proof illustration of valid inequality 2

$X = p_j S_j + \frac{p_j(p_j+1)}{2} - \frac{p_j}{2}$

$= p_j S_j + \frac{p_j^2}{2}$

$= p_j(S_j + \frac{p_j}{2})$

$= p_j(C_j - p_j + \frac{p_j}{2})$

$= p_j(C_j - \frac{p_j}{2})$

Thus $p_j C_j = \frac{p_j^2}{2} + X$

Since $X = \sum_{t=1}^{p_j} S_j + t - \frac{1}{2} = \sum_{t=1}^{p_j}(S_j + t - \frac{1}{2})(\sum_{k\in M_1} A_{1j,t}^k + \sum_{l\in M_2} A_{2j,t}^l)$

$= \sum_{t\in T}(t - \frac{1}{2}) \times (\sum_{k\in M_1} A_{1j,t}^k + \sum_{l\in M_2} A_{2j,t}^l)$

and $C_{max} \geq C_j$ forall j

2.6. Heuristics

We consider four simple heuristics to determine an upper bound on the number of time periods.

2.6.1. H0

Heuristic H0 is a list scheduling algorithm similar to the algorithm in [6].

2.6.2. Heuristic H1

Heurictic H1 first sorts the jobs in non-increasing order of their processing times on stage 2 (p_{2j}), then executes H0.

2.6.3. Heuristic H2

Heurictic H2 sorts the jobs in non-increasing order of their processing times on stage 1 : p_{1j}. Then, H0 is executed.

2.6.4. Heuristic H3

In Heuristic H3, H2 is executed to obtain a machine assignment of the jobs. Then, the jobs assigned to each stage 1 machine are sorted in non-increasing order of their stage 2 processing times : p_{2j}. Schedule is then rearranged based on the new sorted job sequences.

2.6.5. Illustrative example

Consider the same example given in Table 1 with 5 jobs, 2 stage-1 machines and 1 stage-2 machine. The solutions given by the four heuristics are illustrated in figure 10.

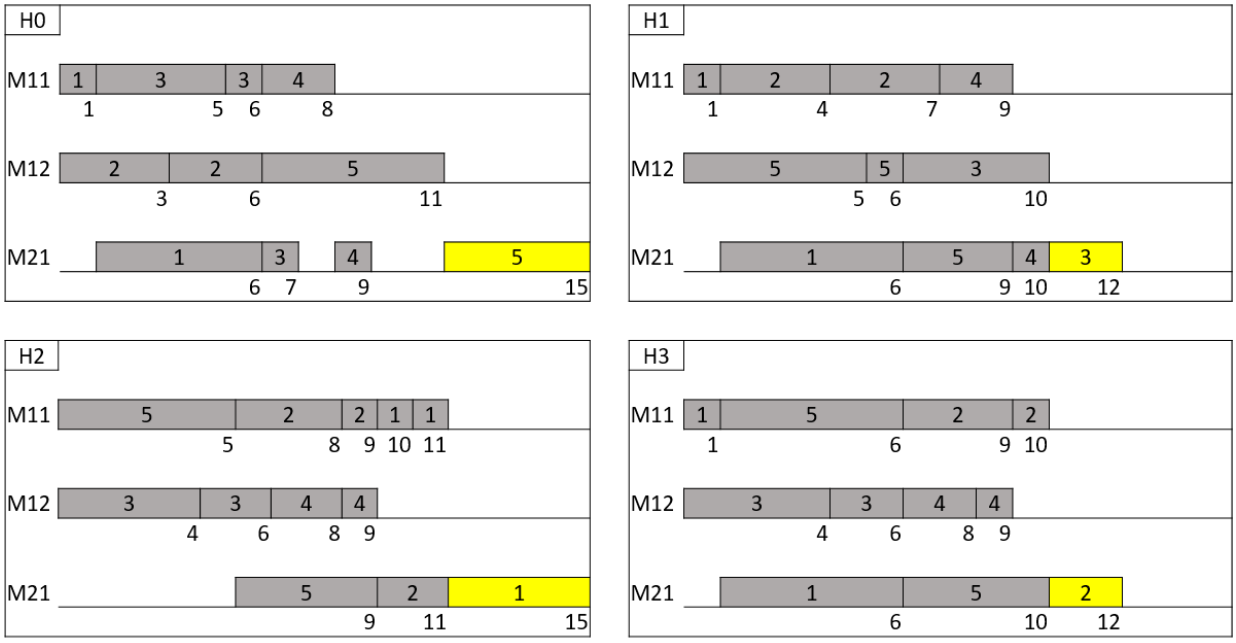


Figure 10: Schedules obtained by each heuristic

.

2.7. Binary Search

A binary search algorithm between the best lower bound LB and upper bound UB was implemented as detailed in the pseudo-code 1. TIM(LB, UB) refers to applying the Time-Indexed Model between LB and UB. The idea is to divide the search space in half in a recursive manner and run the model starting by the first half until a solution is found or the interval $[LB, UB]$ reduces to one. In this way, if a solution is found, it will be the smallest possible. When the model is run using the optimizer on a given search interval, there are four possible outcomes : optimal status, feasible status, infeasible status, and timeout status. The algorithm details the course of action corresponding to each outcome.

We give two examples to illustrate the execution of the algorithm. In the first example presented in Table 2, the initial search interval is [25,28]. The binary search procedure divides the interval and allows

Table 2: Sample execution of the binary search algorithm : example 1

Step	LB0	UB0	UB1	Status	Next step	BestCmax
0	25	28			BinarySearch(25,28)	28
1	25	28	26	Infeasible	BinarySearch(27,28)	28
2	27	28	27	Optimal; OPT=27		27

Table 3: Sample execution of the binary search algorithm : example 2

Step	LB0	UB0	UB1	Status	Next step	BestCmax
0	24	29			BinarySearch(24,29)	29
1	24	29	26	Timeout	BinarySearch(24,26)	29
2	24	26	25	Infeasible	BinarySearch(26,29)	29
3	26	29	27	Timeout	BinarySearch(26,27)	29
4	26	26	26	Timeout	BinarySearch(27,29)	29
5	27	29	28	Feasible; Cmax=28	BinarySearch(27,28)	28
6	27	28	27	Timeout		28

the model to find the optimal solution after eliminating the infeasible values of 25 and 26.

The second example in Table 3 starts with a search interval between 24 and 29. First, the model is run on the interval [24, 26]. The time limit is reached without finding a solution. Once the interval is reduced to [24,25] however, the model exits with the infeasible status, which improves the lower bound to 26. The new search interval is thus reduced to [26,29]. On one hand, a feasible solution is found with the makespan value of 28. On the other hand, the optimal solution remains unknown because of the timeout status in steps 3 and 4. Moreover, the best makespan value is only improved from 29 to 28. Therefore, the binary search method is not always effective.

If using the binary search algorithm does not yield a feasible solution, a minimum bounding search is conducted by iterating step by step starting from the lower bound until reaching a feasible solution or the upper bound, as shows algorithm 2.

Preliminary tests revealed that the binary search algorithm fails to find feasible solutions for all tested instances. Thus, the incremental minimum bounding strategy is adopted hereafter.

3. Experimental Results

Experiments are conducted on a machine with 32.0 GB RAM and Intel(R) Xeon(R) CPU E5-2620 v4 @ 2.10GHz, under Linux Operating System. Lower bounds and heuristics are implemented using Visual

Algorithm 1: BinarySearch(LB,UB)

```
Data: LB, UB
BestC_max ← UB;
begin
    UB' ← ⌊(LB+UB)/2⌋;
    TIM(LB,UB');
    switch Status do
        case Optimal do
            BestC_max ← OPT;
        end
        case Feasible do
            BestC_max ← C_max;
            if C_max ≠ LB then
                BinarySearch(LB,C_max);
            end
        end
        case Infeasible do
            if LB = UB − 1 then
                BestC_max = UB;
            else
                LB ← UB' + 1;
                BinarySearch(LB,BestC_max);
            end
        end
        case Timeout do
            if LB ≠ UB' then
                BinarySearch(LB,UB');
            else
                LB ← UB' + 1;
                BinarySearch(LB, BestC_max);
            end
        end
    end
end
```

Algorithm 2: MinimumBounding(LB,UB)

```
Data: LB, UB
begin
    for c ← LB to UB do
        TIM(c,c);
        switch Status do
            case Optimal do
                BestC_max ← OPT;
                break;
            end
            case Feasible do
                if C_max < BestC_max then
                    BestC_max ← C_max;
                    UB ← C_max;
                end
            end
        end
    end
end
```

C++ 2019. The mathematical model and valid inequalities are implemented using IBM ILOG CPLEX 12.8 Concert Technology.

3.1. Test instances

Test instances are generated based on the generation data provided in [5]. There are 6 classes of instances. Each class contains 60 instances, grouped in 4 sub-classes. Every sub-class represents a combination $N - m_1 - m_2$ and contains 15 instances with different values of processing times p_{1j} and p_{2j}. Processing times are generated following a uniform distribution. Table 4 provides data generation details. First results

Table 4: Test instances information

Class	**n**	m_1	m_2	p_{1j}	p_{2j}
1	10	2,4	2,4	uniform[4,22]	uniform[6,24]
3	15	2,4	2,4	uniform[4,22]	uniform[6,24]
4	20	2,4	2,4	uniform[4,22]	uniform[6,24]
5	30	2,6	2,6	uniform[4,22]	uniform[6,24]
6	10	1,2	1,2	uniform[4,22]	uniform[6,24]
7	10	2,4	2,4	uniform[18,24]	uniform[18,24]

showed that the model could not find feasible solutions for class 1 with 10 jobs.
Smaller instances were generated by reducing processing times as detailed in table 5.

Table 5: Reduced instances information

Class	**n**	m_1	m_2	p_{1j}	p_{2j}
CL1	10	2,4	2,4	uniform[2,11]	uniform[3,12]
CL3	15	2,4	2,4	uniform[2,11]	uniform[3,12]
CL4	20	2,4	2,4	uniform[2,11]	uniform[3,12]
CL5	30	2,6	2,6	uniform[2,11]	uniform[3,12]
CL6	10	1,2	1,2	uniform[2,11]	uniform[3,12]
CL7	10	2,4	2,4	uniform[9,12]	uniform[9,12]

Moreover, we generate another set of 6 larger instance classes in order to further evaluate the heuristics. We generate the large instances following the same rules adapted to the hospital context (Augusto, 2010). The newly generated instances have up to 300 patients and up to 60 operating rooms and 60 recovery beds as detailed in Table 6. It is possible to add another instance class with 500 patients although this number is not realistic in the operating room context.

Table 6: Large instances information

Class	**n**	m_1	m_2	p_{1j}	p_{2j}
L1	100	20,40	20,40	uniform[2,11]	uniform[3,12]
L3	150	20,40	20,40	uniform[2,11]	uniform[3,12]
L4	200	20,40	20,40	uniform[2,11]	uniform[3,12]
L5	300	20,60	20,60	uniform[2,11]	uniform[3,12]
L6	100	10,20	10,20	uniform[2,11]	uniform[3,12]
L7	100	20,40	20,40	uniform[9,12]	uniform[9,12]

3.2. Time-Indexed Model and valid inequalities

An instance is labeled as solved if a feasible solution is found by the optimizer.

Note that all CPLEX parameters were set to default except the starting algorithm in the root node of the branching tree. This parameter was set to the primal algorithm, which yielded better results than the default CPLEX setting on all tested instances.

An instance is labeled as optimally solved if either $LB^* = UB$ or $LB^* = B^*$. In such a case, we denote OPT the value of LB^*. B^* denotes the best value found by the optimizer for the makespan. We also denote BEST the best solution value found either by the optimizer or by the heuristics.

Generally speaking, $LB^* = LB$. However, LB sometimes is proved infeasible by the optimizer. In such cases, LB^* takes the smallest value greater than LB that does not prove infeasible.

The majority of instances with 10 jobs is solved. However, the rate of instances solved optimally remains low.

The results seem to depend not only on instance sizes (numbers of machines) but also on shop configurations.

We test different cut settings.

- B denotes the makespan value obtained by CPLEX default cut setting.
- B1 denotes the makespan value obtained by applying cplex default cuts and valid inequality 1.
- B2 denotes the makespan value obtained by applying cplex default cuts and valid inequality 2.
- B12 denotes the makespan value obtained by applying cplex default cuts and valid inequalities 1 and 2.
- B-0 denotes the makespan value obtained by applying no cuts.
- B1-0 denotes the makespan value obtained by applying valid inequality 1 only.
- B2-0 denotes the makespan value obtained by applying valid inequality 2 only.

- B12-0 denotes the makespan value obtained by applying valid inequalities 1 and 2 only.

To assess the performance of the valid inequalities, Table 7 reports the gaps between LB* and the value obtained by a given cut setting. Results show that CPLEX default cut settings yield the lowest average gaps for classes CL1, CL3, CL6 and CL7. Average gaps vary between 5% and 18%. For the sub-class CL1_10_2_2, valid inequality 2 yields the lowest average gap. For class CL4 with 20 jobs, CPLEX default cut settings fail to find feasible solutions, whereas the valid inequalities are more effective. For sub-class CL4_20_2_2, 3 cut configurations with valid inequality 1 yield an average gap of 19%. For subclass CL4_20_4_4, all configurations with valid inequalities 1 and/or 2 yield an average gap of 27%. Overall, the smallest average gap of 23% is achieved for class CL4 by configurations B12, B12-0, and B1-0.

3.3. Lower Bounds Comparison

We compare among the four lower bounds in terms of the relative average gap from the best solution value found.

$$GapLBi = \frac{BEST - LBi}{BEST}(\%)$$

Results reported in Table 8 and Figure 11 show that LB4 outperforms the other lower bounds on all instances. The smallest gaps in each row are reported in bold.

LB3 performs equally as well as LB4 on all instances where the number of stage-1 machines exceeds the number of stage-2 machines, more specifically where $m_1 = 2\, m_2$. We notice that LB3=LB4 for all instances in this case.

LB3 yields a better gap than LB2 on average. However, LB2 outperforms LB3 in the subgroup where $m_1 < m_2$. LB1 has the worst average performance, except for the subgroup $CL1\_10\_4\_4$ for which LB1 yields the second best gap.

3.4. Heuristics Comparison

3.4.1. Results for small instances

Heuristics results are reported in Table 9 and in Figure 12.

The relative average gaps from the best upper bound UB are calculated as follows.

$$GapHi = \frac{UBi - BEST}{BEST}(\%)$$

where UBi denotes the upper bound obtained by heuristic Hi.

The gaps are the smallest within the second subgroup for all classes. Average gaps vary between 8% and 20%.

No heuristic outperforms the others on all instances. However, heuristic H1 has the best average performance for all instance classes.

Heuristic H3 performs better than Heuristic H2 on average.

Table 7: Gaps from LB* (%)

Class	N_M1_M2	B	B12	B1	B2	B-0	B12-0	B1-0	B2-0
	10_2_2	10	16	15	15	10	17	17	**6**
	10_2_4	**4**	-	14	-	5	-	-	-
CL1	10_4_2	**12**	18	18	19	16	21	22	20
	10_4_4	**10**	15	17	19	17	16	16	16
	Average	**10**	17	17	18	14	19	19	17
	15_2_2	**18**	20	20	20	**18**	20	20	20
	15_2_4	-	-	-	-	-	-	-	-
CL3	15_4_2	**13**	-	-	-	23	-	-	-
	15_4_4	**23**	24	24	24	25	24	24	24
	Average	**18**	22	22	22	24	22	22	22
	20_2_2	-	**19**	-	-	-	**19**	**19**	-
	20_2_4	-	-	-	-	-	-	-	-
CL4	20_4_2	-	-	-	-	-	-	-	-
	20_4_4	-	**27**	**27**	**27**	-	**27**	**27**	**27**
	Average	-	**23**	-	27	-	**23**	**23**	27
	10_1_1	**6**	9	10	10	7	11	-	8
	10_1_2	**2**	9	9	4	4	-	-	9
CL6	10_2_1	**3**	8	8	9	6	9	9	16
	10_2_2	**10**	-	-	21	13	-	-	21
	Average	**5**	8	9	11	8	10	9	13
	10_2_2	-	-	-	-	-	-	-	-
	10_2_4	-	-	-	-	-	-	-	-
CL7	10_4_2	**15**	-	-	-	-	-	-	-
	10_4_4	-	-	-	-	-	-	-	-
	Average	**15**	-	-	-	-	-	-	-

3.4.2. Results for large instances

Heuristics results for the large instances are reported in Table 10.

Once again, if we compare among the second sub-group, the smallest gaps are those of sub-class 2 for all classes of instances. Average gaps vary from 15% for class L7 to 45% for class L1.

If we compare among the heuristics, heuristic H1 has the best average performance. Heuristic H2 performs

Table 8: Lower bounds relative average gaps

Class	N_M1_M2	GapLB1	GapLB2	GapLB3	GapLB4
CL1	10_2_2	50%	23%	12%	**9%**
	10_2_4	46%	12%	31%	**2%**
	10_4_2	29%	43%	**11%**	**11%**
	10_4_4	12%	33%	21%	**9%**
	Average	34%	28%	19%	**8%**
CL3	15_2_2	64%	13%	4%	**3%**
	15_2_4	61%	7%	29%	**0%**
	15_4_2	46%	34%	**5%**	**5%**
	15_4_4	35%	23%	15%	**9%**
	Average	51%	19%	13%	**4%**
CL4	20_2_2	71%	9%	3%	**1%**
	20_2_4	68%	5%	31%	**0%**
	20_4_2	55%	30%	**0%**	**0%**
	20_4_4	43%	13%	3%	**1%**
	Average	60%	14%	9%	**1%**
CL5	10_2_2	80%	7%	1%	**0%**
	10_2_6	77%	3%	44%	**0%**
	10_6_2	60%	39%	**0%**	**0%**
	10_6_6	40%	12%	2%	**0%**
	Average	64%	15%	12%	**0%**
CL6	10_1_1	74%	14%	8%	**6%**
	10_1_2	69%	**7%**	27%	**1%**
	10_2_1	59%	34%	**5%**	**5%**
	10_2_2	51%	23%	11%	**9%**
	Average	63%	19%	13%	**5%**
CL7	10_2_2	62%	15%	9%	**0%**
	10_2_4	62%	15%	33%	**0%**
	10_4_2	40%	33%	**2%**	**2%**
	10_4_4	35%	27%	14%	**0%**
	Average	50%	22%	14%	**0%**

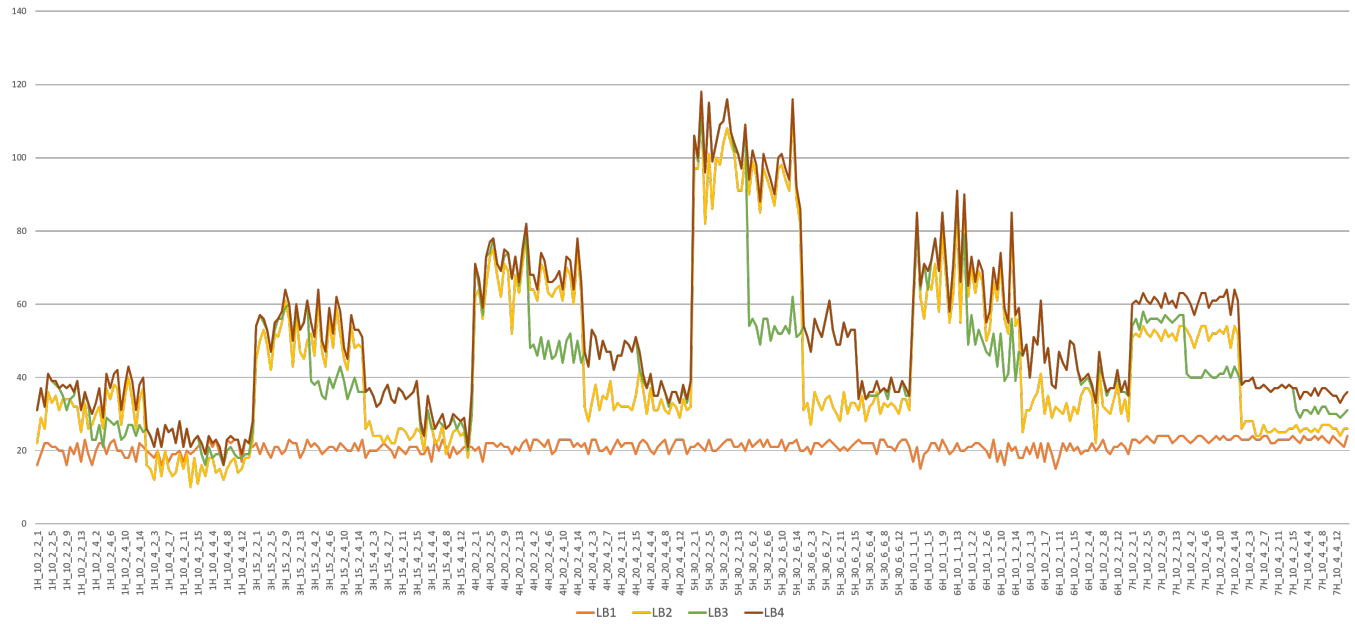


Figure 11: Lower bounds values per instance

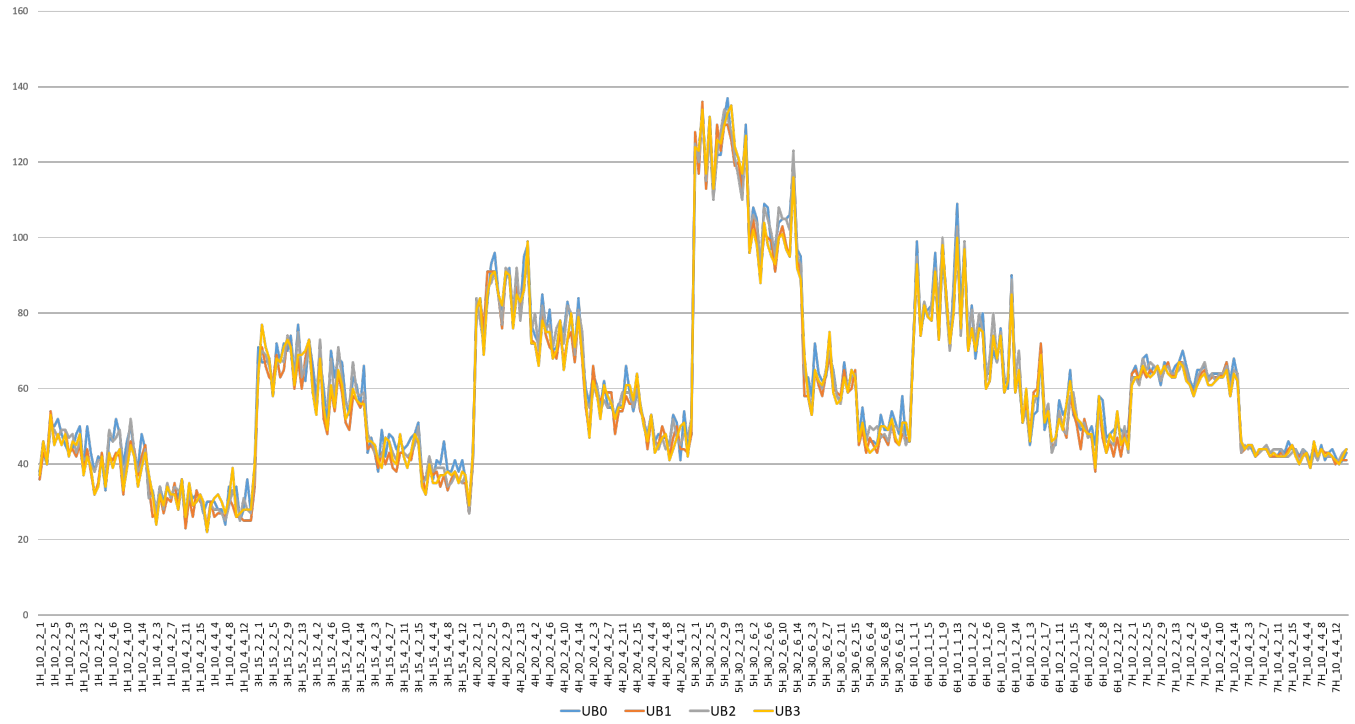


Figure 12: Upper bounds values per instance

slightly better than heuristic H3 on average. Heuristic H0 remains the weakest overall.

Table 9: Upper bounds relative average gaps

Class	N_M1_M2	GapH0	GapH1	GapH2	GapH3
	10_2_2	14%	**10%**	13%	**10%**
	10_2_4	18%	7%	15%	**5%**
CL1	10_4_2	18%	**11%**	16%	16%
	10_4_4	25%	**13%**	18%	23%
	Average	19%	**10%**	15%	13%
	15_2_2	19%	**15%**	17%	19%
	15_2_4	15%	**6%**	13%	7%
CL3	15_4_2	22%	**14%**	20%	18%
	15_4_4	22%	**14%**	17%	16%
	Average	20%	**12%**	17%	15%
	20_2_2	19%	**17%**	**17%**	**17%**
	20_2_4	11%	**5%**	11%	6%
CL4	20_4_2	22%	19%	20%	20%
	20_4_4	29%	**25%**	**28%**	26%
	Average	20%	**16%**	19%	17%
	10_2_2	17%	**16%**	**16%**	18%
	10_2_6	7%	2%	**7%**	**1%**
CL5	10_6_2	20%	**15%**	17%	17%
	10_6_6	35%	**26%**	32%	30%
	Average	20%	**15%**	18%	17%
	10_1_1	10%	**7%**	9%	**7%**
	10_1_2	6%	**4%**	9%	**4%**
CL6	10_2_1	**10%**	11%	**10%**	12%
	10_2_2	18%	**9%**	13%	13%
	Average	11%	**8%**	10%	9%
	10_2_2	8%	6%	6%	**5%**
	10_2_4	5%	3%	4%	**1%**
CL7	10_4_2	14%	**13%**	14%	14%
	10_4_4	20%	**18%**	**18%**	**18%**
	Average	12%	**10%**	**10%**	**10%**

Table 10: Upper bounds relative average gaps for large instances

Class	N_M1_M2	GapH0	GapH1	GapH2	GapH3
	100_20_20	44%	**32%**	38%	37%
	100_20_40	35%	**17%**	26%	**17%**
L1	100_40_20	56%	**38%**	43%	54%
	100_40_40	45%	**28%**	34%	50%
	Average	45%	**29%**	35%	40%
	150_20_20	33%	**27%**	**27%**	31%
	150_20_40	24%	**13%**	19%	17%
L3	150_40_20	38%	**29%**	31%	34%
	150_40_40	54%	**38%**	43%	49%
	Average	37%	**27%**	30%	33%
	200_20_20_	28%	**23%**	**23%**	26%
	200_20_40	19%	**11%**	16%	14%
L4	200_40_20	30%	**22%**	25%	28%
	200_40_40	48%	**34%**	38%	36%
	Average	31%	**23%**	25%	26%
	300_20_20	21%	19%	**18%**	23%
	300_20_60	13%	6%	10%	**5%**
L5	300_60_20	26%	**18%**	20%	22%
	300_60_60	49%	**34%**	39%	38%
	Average	27%	**19%**	22%	22%
	100_10_10	26%	**22%**	**22%**	26%
	100_10_20	17%	**9%**	15%	12%
L6	100_20_10	27%	**20%**	22%	23%
	100_20_20	43%	**33%**	35%	35%
	Average	28%	**21%**	24%	24%
	100_20_20	14%	**10%**	**10%**	**10%**
	100_20_40	12%	8%	6%	**4%**
L7	100_40_20	25%	**20%**	**20%**	22%
	100_40_40	27%	**22%**	23%	29%
	Average	19%	**15%**	**15%**	16%

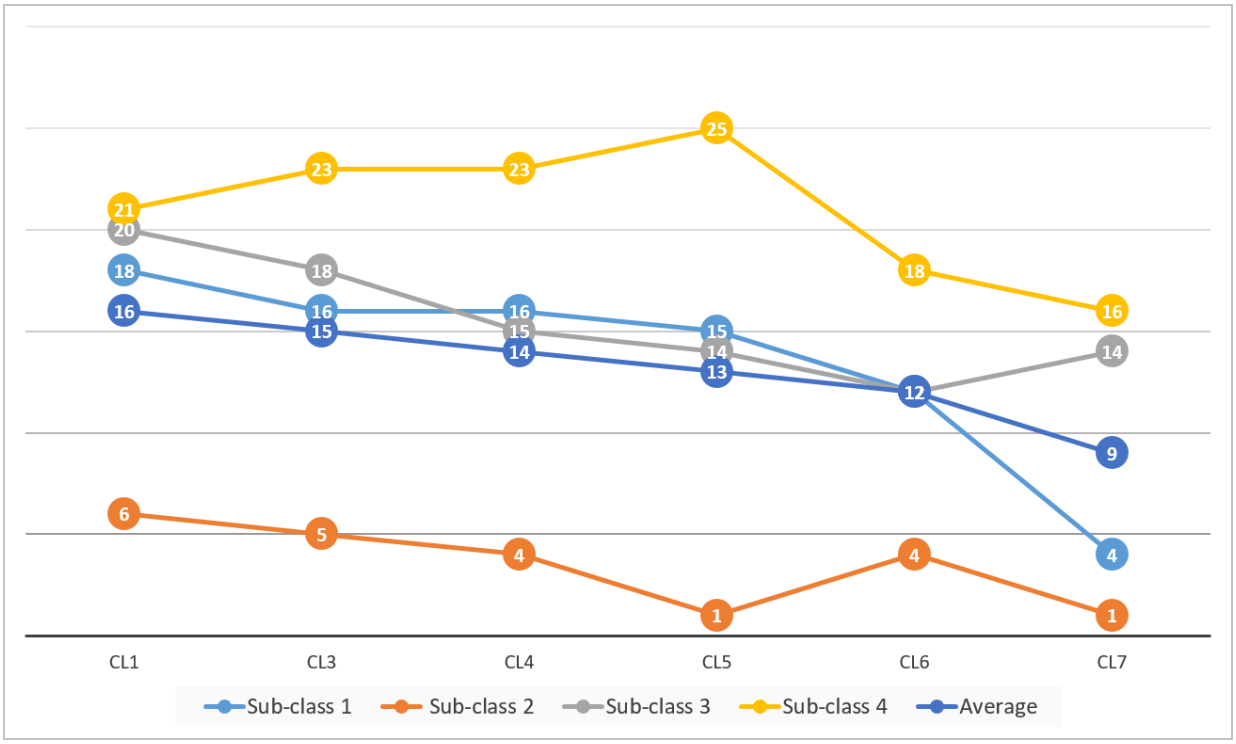


Figure 13: Relative gaps between UB and LB for small instances (%)

3.5. Gaps between the best upper bounds and lower bounds

Figures 13 and 14 respectively present the gaps between the best upper and lower bounds for the small and large instances respectively.

Sub-class 1 refers to the situation where the number of machines is at the same time the lowest and equal in both stages. Sub-class 2 refers to the case where the number of machines in stage 1 is lower than that in stage 2 ($m_1 < m_2$). Sub-class 3 refers to the case where the number of machines in stage 1 is greater than that in stage 1 ($m_1 > m_2$). Sub-class 4 refers to the case where the number of machines is the largest and at the same time equal in both stages.

3.5.1. Results for small instances

The maximum average gap is 16 %.

For each class, the gaps are the largest in sub-class 4 and the lowest in sub-class 2. For the same number of machines and the same distribution parameters of processing times, the gaps slightly decrease with the number of jobs (Classes CL1, CL3 and CL4). For the same number of jobs and the same distribution parameters of processing times, gaps are lower in the class where the number of machines is lower (CL6) compared to CL1. For the same number of jobs and the same number of machines, gaps are lower in the class where the processing times are distributed more tightly (CL7 compared to CL1).

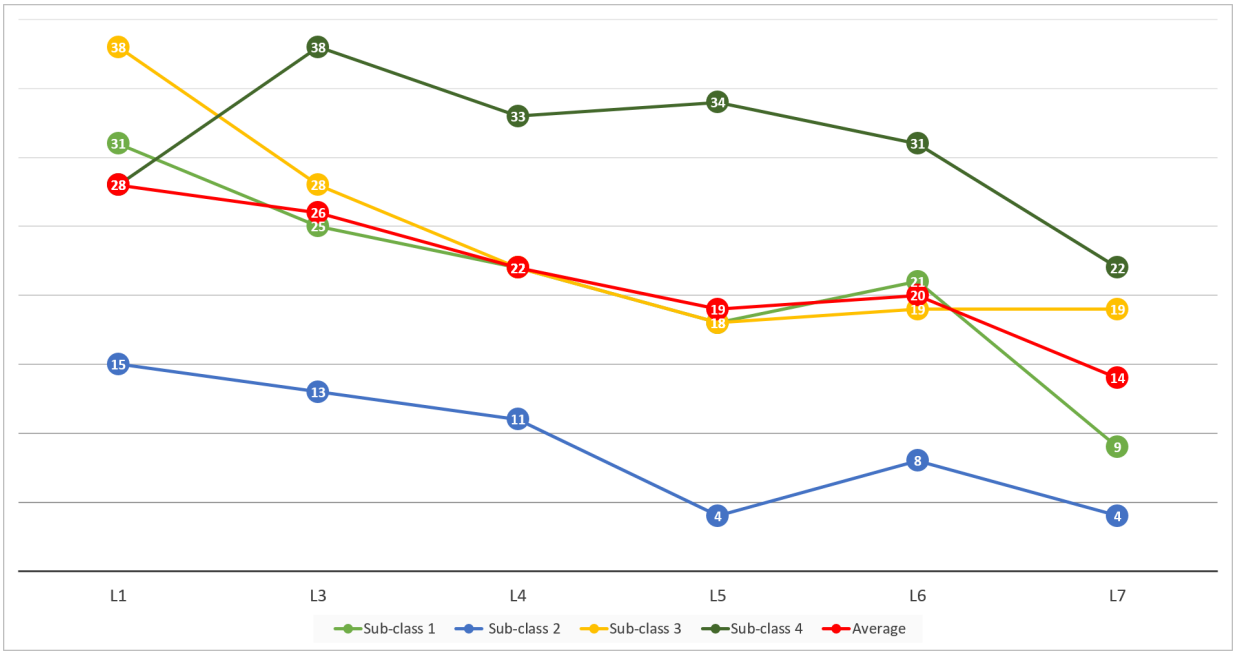


Figure 14: Relative gaps between UB and LB for large instances (%)

3.5.2. Results for large instances

The average gaps vary between 14% and 28%. The lowest gaps are those of sub-class 2 for all classes. The gaps of sub-class 1 and sub-class 3 are similar and close to the average gaps for all classes except for L1 and L7. There is a difference of 7% and 10% respectively between the gaps of sub-class 1 and those of sub-class 3 for classes L1 and L7,respectively, whereas this difference does not exceed 3% for the other classes.The highest gaps are those of sub-class 4 for all classes except L1.

For the same number of jobs and the same distribution of processing times (classes L1 and L6), average gaps are lower for the class with a smaller number of machines (L1). For the classes L1 and L7, similarly to the small classes, average gaps are lower in the class where the processing times are distributed more tightly (L7). For the same number of machines and the same processing times' distribution (classes L1, L3, L4, and L5), the higher the number of jobs, the lower the average gap.

3.6. Percentages of solved and optimally solved instances

Figure 15 details the percentages of solved instances by each cut setting. The ability of the model to find feasible solutions decreases with the instance size. Class CL6 with 10 jobs and a maximum of 2 machines per stage is solved to 95%. Class CL1 with 10 jobs and a maximum of 4 machines per stage is solved to 87%. The percentage of solved instances drops significantly to 12% and 7% respectively for classes CL3 and CL4 respectively.

Classes CL1 and CL7 have the same number of jobs and machines, but only differ by the distribution of processing times. Higher processing times in class CL7 strongly affect the solution performances. Class

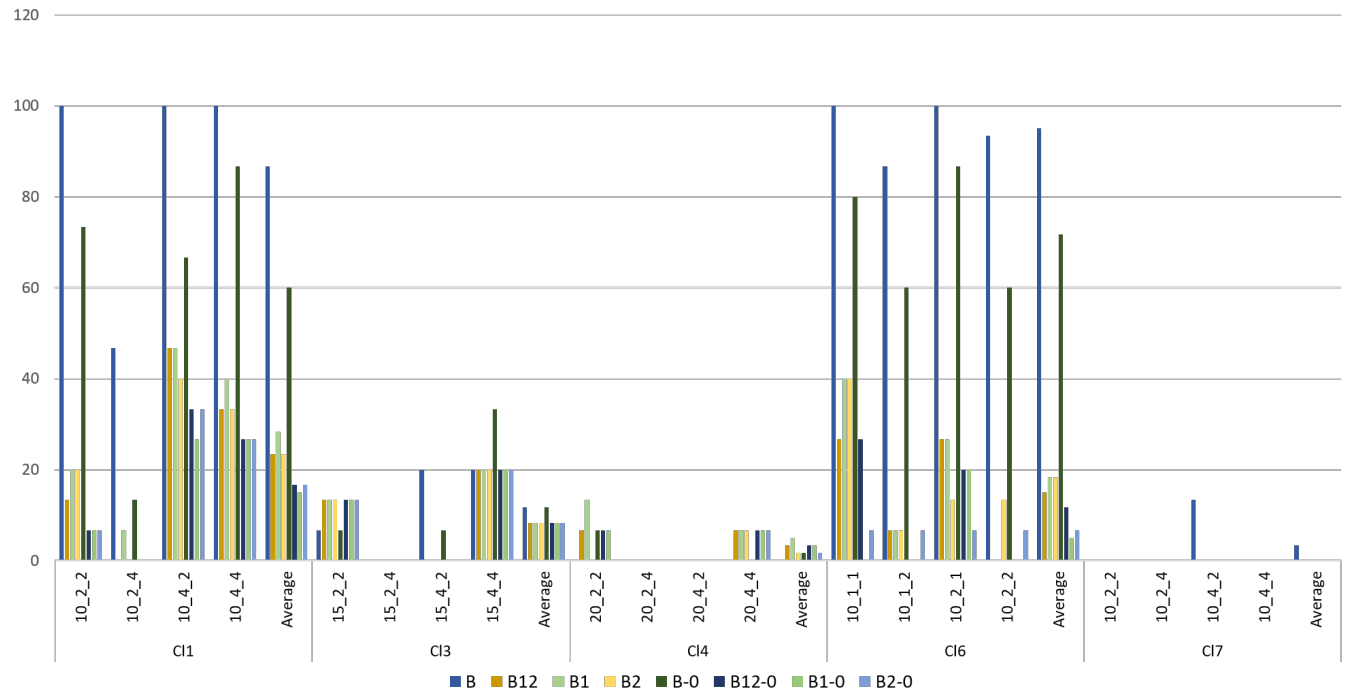


Figure 15: Percentages of solved instances

CL7 with tighter processing times distribution remains practically unsolved.

Figure 16 reports the percentages of optimally solved instances that have been achieved either by the model in each cut setting or by the lower and upper bounds. For classes CL1 and CL6 respectively, only 3% and 10% respectively are solved optimally using the default cut settings. Rates are lower for the remaining cut settings.

We notice the contribution of the heuristics in proving the optimality of 7% of instances in classes CL4, CL6, and CL7. It should be mentioned that all the instances that have been optimally solved by the heuristics belong to the subgroup where $m_1 < m_2$.

4. Discussion

Lower bounds comparison postulates that LB4 may dominate the other lower bounds. One could attempt a proof of this result.

The case where the number of stage-1 machines exceeds the number of stage-2 machines is when inter-stage flexibility is the most likely to occur. We could try to establish whether we have always LB3=LB4 when $m_1 = 2\ m_2$ and whether the result can be generalized to the case $m_1 > m_2$. Results on instances with $m_1 = 3$ and $m_2 = 2$ show that there are cases where $LB3 \neq LB4$, thus this generalization is not possible. Heuristic H0 has the lowest average performance, but there are instances where H0 outperforms all other heuristics. Since the computational times of running the four heuristics are close to zero, it is better to

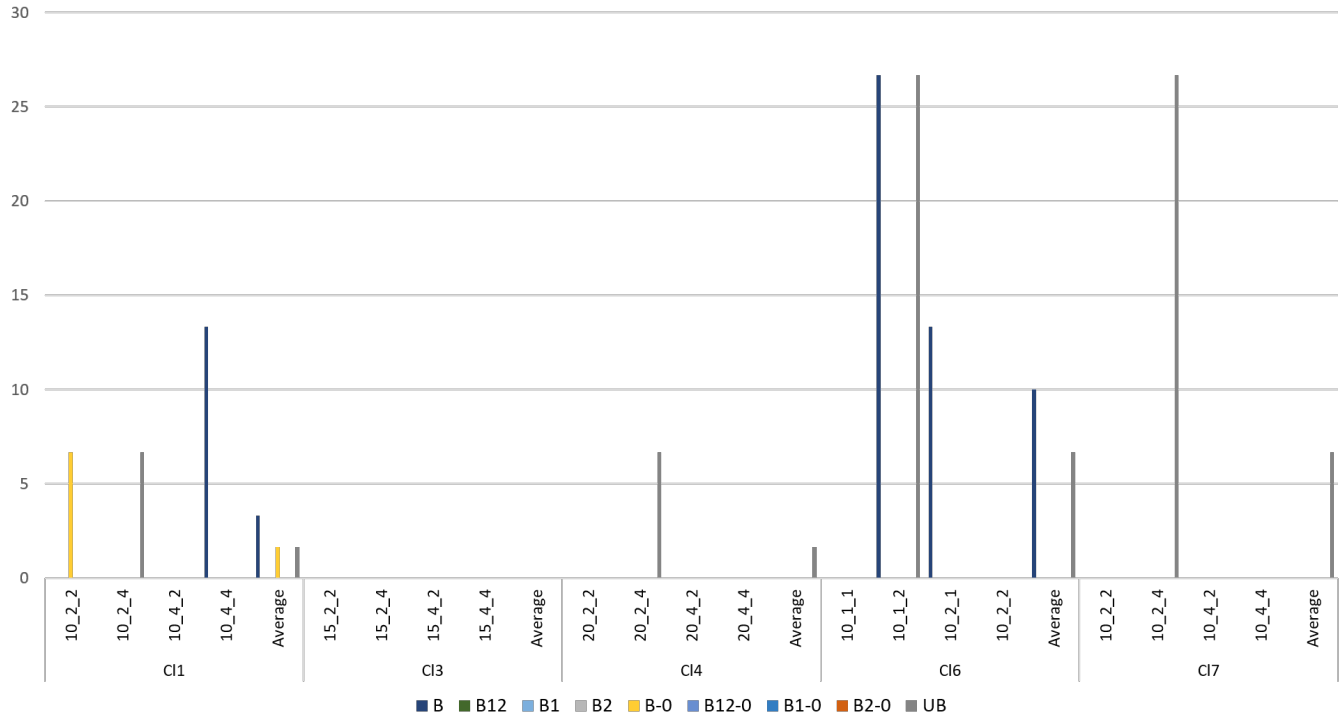


Figure 16: Percentages of optimally solved instances

determine the best heuristic on an instance basis.

H1 and H3 perform the best overall. Both heuristics involve sorting the jobs according to the LPT rule on stage-2.

5. Conclusions

In this paper, we studied a two-stage no-wait hybrid flow shop scheduling problem with inter-stage flexibility for makespan minimization. We developed a time-indexed integer linear programming model for the problem. We proposed four lower bounds, four heuristics as well as two valid inequalities to strengthen the mathematical formulation.

Results show that the model implemented with default CPLEX cut settings yields good solutions for small to medium sized instances with up to 15 jobs and up to 4 machines in each stage. The proposed valid inequalities perform better with the larger instances of up to 20 jobs and 4 machines per stage. For these instances, various combinations of the valid inequalities provide solutions where CPLEX default cuts fail to do so. The developed heuristics yield good results for the small as well as the large instances.

Future research venues include other exact solution strategies as well as approximation algorithms for the problem. Using the developed lower bounds and heuristics in a Branch & Bound algorithm seems to be a promising perspective. Another possible direction is toward approximation algorithms with a

performance guarantee. The time-indexed mathematical model's linear relaxation is a good basis to devise a primal-dual algorithm. Finally, developing another time-indexed model is a third possible venue. We can compare mathematical formulations with different definitions of the decision variables, similar to the work in [36] for flexible job shop scheduling.

References

[1] R. Ruiz, J. A. Vázquez-Rodriguez, The hybrid flow shop scheduling problem, European journal of operational research 205 (1) (2010) 1–18.

[2] I. Ribas, R. Leisten, J. M. Framiñan, Review and classification of hybrid flow shop scheduling problems from a production system and a solutions procedure perspective, Computers & Operations Research 37 (8) (2010) 1439–1454.

[3] B. Naderi, S. Gohari, M. Yazdani, Hybrid flexible flowshop problems: Models and solution methods, Applied mathematical modelling 38 (24) (2014) 5767–5780.

[4] T. Lee, Y. Loong, A review of scheduling problem and resolution methods in flexible flow shop, International Journal of Industrial Engineering Computations 10 (1) (2019) 67–88.

[5] V. Augusto, X. Xie, V. Perdomo, Operating theatre scheduling with patient recovery in both operating rooms and recovery beds, Computers & Industrial Engineering 58 (2) (2010) 231–238.

[6] W. Zhong, Y. Shi, Two-stage no-wait hybrid flowshop scheduling with inter-stage flexibility, Journal of Combinatorial Optimization 35 (1) (2018) 108–125.

[7] J. Dong, H. Pan, C. Ye, W. Tong, J. Hu, No-wait two-stage flowshop problem with multi-task flexibility of the first machine, Information Sciences 544 (2021) 25–38.

[8] D. Khorasanian, F. Dexter, E. Demeulemeester, G. Moslehi, Minimising the number of cancellations at the time of a severe lack of postanesthesia care unit beds or nurses, International Journal of Production Research (2021) 1–14.

[9] M. Mazloumian, M. F. Baki, M. Ahmadi, A robust multiobjective integrated master surgery schedule and surgical case assignment model at a publicly funded hospital, Computers & Industrial Engineering 163 (2022) 107826.

[10] J. Zhang, M. Dridi, A. El Moudni, A two-phase optimization model combining markov decision process and stochastic programming for advance surgery scheduling, Computers & Industrial Engineering 160 (2021) 107548.

[11] J. Park, B.-I. Kim, M. Eom, B. K. Choi, Operating room scheduling considering surgeons' preferences and cooperative operations, Computers & Industrial Engineering 157 (2021) 107306.

[12] C. Sriskandarajah, P. Ladet, Some no-wait shops scheduling problems: complexity aspect, European journal of operational research 24 (3) (1986) 424–438.

[13] C. Sriskandarajah, Performance of scheduling algorithms for no-wait flowshops with parallel machines, European Journal of Operational Research 70 (3) (1993) 365–378.

[14] S. Wang, M. Liu, C. Chu, A branch-and-bound algorithm for two-stage no-wait hybrid flow-shop scheduling, International Journal of Production Research 53 (4) (2015) 1143–1167.

[15] C. Pan, J. Chen, Scheduling alternative operations in two-machine flow-shops, Journal of the Operational Research Society 48 (5) (1997) 533–540.

[16] Q. Wei, E. Shan, L. Kang, A fptas for a two-stage hybrid flow shop problem and optimal algorithms for identical jobs, Theoretical Computer Science 524 (2014) 78–89.

[17] V. Fernandez-Viagas, P. Perez-Gonzalez, J. M. Framinan, Efficiency of the solution representations for the hybrid flow shop scheduling problem with makespan objective, Computers & Operations Research 109 (2019) 77–88.

[18] C. Gicquel, L. Hege, M. Minoux, W. Van Canneyt, A discrete time exact solution approach for a complex hybrid flow-shop scheduling problem with limited-wait constraints, Computers & Operations Research 39 (3) (2012) 629–636.

[19] E. H. Bowman, The schedule-sequencing problem, Operations Research 7 (5) (1959) 621–624.

[20] A. A. B. Pritsker, L. J. Waiters, P. M. Wolfe, Multiproject scheduling with limited resources: A zero-one programming approach, Management science 16 (1) (1969) 93–108.

[21] R. F. Deckro, J. E. Hebert, Resource constrained project crashing, Omega 17 (1) (1989) 69–79.

[22] J. H. Patterson, W. D. Huber, A horizon-varying, zero-one approach to project scheduling, Management Science 20 (6) (1974) 990–998.

[23] D. Das, W. Dowsland, A model to examine the effect of loading conditions on scheduling rules in a job shop environment, THE INTERNATIONAL JOURNAL OF PRODUCTION RESEARCH 19 (5) (1981) 577–587.

[24] R. F. Deckro, J. E. Hebert, E. Winkofsky, Multiple criteria job-shop scheduling, Computers & Operations Research 9 (4) (1982) 279–285.

[25] G. L. Thompson, D. J. Zawack, A problem expanding parametric programming method for solving the job shop scheduling problem, Annals of Operations Research 4 (1) (1985) 327–342.

[26] J. P. Sousa, L. A. Wolsey, A time indexed formulation of non-preemptive single machine scheduling problems, Mathematical programming 54 (1) (1992) 353–367.

[27] T. Morton, D. W. Pentico, Heuristic scheduling systems: with applications to production systems and project management, Vol. 3, John Wiley & Sons, 1993.

[28] L. A. Wolsey, Mip modelling of changeovers in production planning and scheduling problems, European Journal of Operational Research 99 (1) (1997) 154–165.

[29] C. A. Floudas, X. Lin, Mixed integer linear programming in process scheduling: Modeling, algorithms, and applications, Annals of Operations Research 139 (1) (2005) 131–162.

[30] E. Stafford, F. T. Tseng, J. N. Gupta, Comparative evaluation of milp flowshop models, Journal of the Operational Research Society 56 (1) (2005) 88–101.

[31] Y. Demir, S. K. İşleyen, Evaluation of mathematical models for flexible job-shop scheduling problems, Applied Mathematical Modelling 37 (3) (2013) 977–988.

[32] D. Khorasanian, G. Moslehi, Two-machine flow shop scheduling problem with blocking, multi-task flexibility of the first machine, and preemption, Computers & Operations Research 79 (2017) 94–108.

[33] J. Carlier, Scheduling jobs with release dates and tails on identical machines to minimize the makespan, European Journal of Operational Research 29 (3) (1987) 298–306.

[34] R. McNaughton, Scheduling with deadlines and loss functions, Management Science 6 (1) (1959) 1–12.

[35] V. T'kindt, J.-C. Billaut, Multicriteria scheduling: theory, models and algorithms, Springer Science & Business Media, 2006.

[36] K. Thörnblad, On the optimization of schedules of a multitask production cell, Chalmers Tekniska Hogskola (Sweden), 2011.