Int. J. Appl. Math. Comput. Sci., 2023, Vol. 33, No. 2, 285–295
DOI: 10.34768/amcs-2023-0021

EXACT APPROACHES TO LATE WORK SCHEDULING ON UNRELATED MACHINES

XINBO LIU [a], WEN WANG [b], XIN CHEN [b,*], MALGORZATA STERNA [c], JACEK BLAZEWICZ [c,d]

[a] SolBridge International School of Business
Woosong University
Uam-ro 128, Daejeon 34613, Republic of Korea
e-mail: lxb715@hotmail.com

[b] School of Electronics and Information Engineering
Liaoning University of Technology
Shiying 169, Jinzhou 121001, China
e-mail: wangwen.lut@outlook.com, chenxin.lut@hotmail.com

[c] Institute of Computing Science
Poznan University of Technology
ul. Piotrowo 2, 60-965 Poznan, Poland
e-mail: {malgorzata.sterna, jblazewicz}@cs.put.poznan.pl

[d] European Centre for Bioinformatics and Genomics
Polish Academy of Sciences
ul. Piotrowo 2, 60-965 Poznan, Poland

We consider the scheduling problem on unrelated parallel machines in order to minimize the total late work. Since the problem is NP-hard, we propose a mathematical model and two dedicated exact approaches for solving it, based on the branching and bounding strategy and on enumerating combined with a dynamic programming algorithm. The time efficiencies of all three approaches are evaluated through computational experiments.

Keywords: late work scheduling, unrelated machines, mathematical model, branch-and-bound algorithm, dynamic programming.

1. Introduction

The total late work (Y) is a performance measure which allows minimizing the size of tardy parts of jobs (Błażewicz *et al.*, 2019). Since the proposal of late work in 1984 (Błażewicz, 1984), this measure has been intensively studied for the cases of a single machine, parallel identical machines and dedicated machines (see the survey paper by Sterna (2011)). Scheduling models with late work minimization are quite closely related to the ones with early work maximization (Sterna and Czerniachowska, 2017). When offline optimal solutions are considered, the two types of models share the same essence. However, when approximation solutions are constructed, they are distinguished with regard to problem characteristics (Sterna, 2021).

In this paper, we focus on the basic problem of scheduling jobs characterized by processing time on parallel unrelated machines, which has been very rarely studied in the literature in the context of late work minimization. The total late work for unrelated machines was investigated for more complex models than the one studied in the presented paper. These models, inspired by specific practical applications, include additional constraints and parameters such as precedence constraints, communication delays (Abasian *et al.*, 2014), release time, setup time and machine

*Corresponding author

eligibility (Afzalirad and Rezaeian, 2016). For the problems mentioned mathematical models were designed and some meta-heuristic approaches proposed. In our research we continue the more basic studies on the unrelated machine scheduling problem with the total late work reported by Wang *et al*. (2020), who proposed two meta-heuristic approaches. The natural step was to design exact algorithms for this problem.

The paper is organized as follows. In Section 2 we formally define the problem investigated within our research and propose a mathematical programming model for it. In Section 3 we comment on the computational complexity of the problem with an arbitrary number of unrelated parallel machines ($R||Y$) and with a fixed number of machines ($Rm||Y$), proposing dynamic programming for the latter case.

In Section 4 we present the branch-and-bound algorithm for problem $R||Y$, describing its particular components such as the encoding scheme, the branching scheme, dominance rules, lower and upper bounds. Section 5 is devoted to another exact algorithm based on enumeration, namely, partitioning jobs into machines, and scheduling them by the dynamic programming algorithm. The practical efficiencies of all three proposed approaches—the mathematical model and two exact algorithms—were checked within the computational experiments reported in Section 6. The research is concluded in Section 7.

2. Problem definition

The paper concerns the problem of scheduling a set of n jobs $J = \{J_1, \ldots, J_j, \ldots, J_n\}$ on a set of m parallel unrelated machines $M = \{M_1, \ldots, M_i, \ldots, M_m\}$. The processing time of a job depends on the machine to which it is assigned, so for each job J_j the vector $[p_{ij}]$ is defined, where p_{ij} denotes the processing time of job J_j on machine M_i ($1 \le i \le m$, $1 \le j \le n$). The execution of job J_j should be completed preferably before its due date d_j. If the job completion time C_j exceeds d_j, the late work Y_j occurs for this job, which is equal to the size of its part scheduled late, i.e., $Y_j = \min\{p_{ij},\ \max\{0, C_j - d_j\}\}$, assuming that J_j is assigned to M_i. The goal is to minimize the total late work in the system, i.e., $Y = \sum_{j=1}^{n} Y_j$. In the reported research we study the problem with an arbitrary number of machines, denoted as $R||Y$ by the three-field notation (Graham *et al*., 1979), as well as its special case where the number of machines is fixed, $Rm||Y$.

The problem considered can be formulated as an integer programming model (denoted as IP), based on the following decision variables x_{ijk} where $1 \le j \le n$, $1 \le k \le n$, and $1 \le i \le m$:

$$x_{ijk} = \begin{cases} 1 & \text{if job } J_j \text{ is scheduled in position } k \\ & \text{on machine } M_i, \\ 0 & \text{otherwise,} \end{cases}$$

$$\min \sum_{j=1}^{n} \max_{\substack{1 \le i \le m \\ 1 \le k \le n}} x_{ijk} \cdot \min \left\{ \begin{array}{l} p_{ij}, \\ \max\{0,\ \sum\limits_{r=1}^{k} \sum\limits_{s=1}^{n} x_{isr} p_{is} - d_j\}, \end{array} \right. \tag{1}$$

$$\sum_{i=1}^{m} \sum_{k=1}^{n} x_{ijk} = 1, \quad 1 \le j \le n, \tag{2}$$

$$\sum_{j=1}^{n} x_{ijk} \le 1, \quad 1 \le i \le m, \quad 1 \le k \le n, \tag{3}$$

$$\sum_{j=1}^{n} x_{ij(k+1)} \le \sum_{j=1}^{n} x_{ijk}, \quad 1 \le i \le m, \quad 1 \le k \le n-1, \tag{4}$$

$$x_{ijk} \in \{0,\ 1\}, \quad 1 \le i \le m, \quad 1 \le j \le n, \quad 1 \le k \le n. \tag{5}$$

We minimize the total late work defined with the formula (1). Late work for a job is never negative, so if job J_j is scheduled on M_i in position k, its late work contributes to the criterion value, otherwise the formula for the late work is multiplied by zero (x_{ijk} = 0) and it is irrelevant for the objective function value. To determine the completion time of job J_j, we sum the processing times of jobs in position k and preceding positions ($r = 1, \ldots, k$) on the same machine M_i, so we sum the processing time of J_j and its predecessors on machine M_i (if any, $s = 1, \ldots, n$). The formula (2) ensures that each job J_j is assigned to exactly one position on exactly one machine, which means that each job must be scheduled, while the formula (3) guarantees that each position on particular machines is occupied by at most one job. Due to the formula (4), positions of jobs assigned to each machine form a continuous sequence, i.e., position $k + 1$ can be occupied by a job only if k is occupied. As we have mentioned, all decision variables are integers according to the formula (5).

The above presented mathematical model allowed us to compare the efficiencies of the exact approaches dedicated for the problem considered with the efficiency of a mathematical programming solver, namely, Gurobi (see Section 6 for details).

3. Problem complexity

The problem with an arbitrary number of unrelated machines, $R||Y$, is strongly NP-hard (Garey and Johnson, 1979), since the corresponding problem with identical machines and a common due date, $P|d_j = d|Y$, is already strongly NP-hard (Chen *et al*., 2016). For this reason, to construct optimal solutions, we have to use methods of exponential time complexity such as the branch-and-bound method proposed in Section 4, or other methods based on enumeration of all possible solutions as the approach given in Section 5.

Furthermore, because the problem with two identical machines and a common due date $P2|d_j = d|Y$ is already NP-hard in the weak sense (Chen *et al*., 2016), the model with unrelated machines $Rm|d_j = d|Y$ is also NP-hard. Moreover, we claim that the principle of the dynamic programming method designed for problem $Pm|d_j = d|Y$ proposed by Chen *et al.* (2020a) is also valid for problem $Rm|d_j = d|Y$. We revise this method adjusting it to the specificity of problem $Rm|d_j = d|Y$, and propose a dynamic programming algorithm, denoted by DP_m. We will further use this algorithm as a lower bound technique in our branch-and-bound algorithm for problem $R||Y$ described in Section 4.

Let the function $f(j,\ E_1,\ E_2,\ \ldots,\ E_m)$ denote the minimal total late work for the first j jobs in the input sequence scheduled on m machines, where the early parts of these jobs are executed for at most $E_1,\ E_2,\ \ldots,\ E_m$ units on machine $M_1,\ M_2,\ \ldots,\ M_m$, respectively. Then DP_m runs as in Algorithm 1.

In the same way as in the work of Chen *et al.* (2020a), DP_m guarantees to obtain an optimal schedule for $Rm|d_j = d|Y$ in $O(nd^m)$ time, which is pseudo-polynomial when m is fixed. Therefore, we have the following theorem.

Theorem 1. *Problem $Rm|d_j = d|Y$ is NP-hard in the weak sense.*

Proof. Problem $Rm|d_j = d|Y$ is NP-hard since its special case $P2|d_j = d|Y$ is NP-hard (Chen *et al*., 2016), and $Rm|d_j = d|Y$ can be solved in pseudo-polynomial time. ■

4. Branch-and-bound algorithm

Due to the strong NP-hardness of problem $R||Y$, determining optimal solutions requires exploring the whole space of problem solutions in exponential time. In order to perform the searching process efficiently, we propose a branch-and-bound algorithm, $B\&B$. The $B\&B$ method divides the problem space according to a certain branching scheme adjusted to the chosen representation of the problem solutions. It explores particular subproblems, and the quality of solutions which could be obtained by solving these subproblems is estimated by a lower bound. The quality of the optimal solution is estimated by an upper bound. Some subproblems can be discarded if their lower bounds exceed the upper bound or they are dominated by other subproblems. The whole exploration process can be modelled with a searching tree. In the following subsections we present particular components of the proposed branch-and-bound algorithm for problem $R||Y$.

4.1. Solution representation and the searching process. Following the idea proposed by Chen *et al.* (2020b) and Wang *et al*. (2020), we use an integer vector with $n + m - 1$ elements to present a feasible schedule of problem $R||Y$. This vector contains n positive numbers $(1,\ \ldots,\ n)$ to indicate each job, and $m - 1$ negative ones $\big(-(m-1), \ldots, -1\big)$ to separate the jobs into m subsequences corresponding to subschedules on particular machines. Without loss of generality, we let all the negative numbers be sequenced in descending order. For example, for $n = 9$ jobs and $m = 4$ unrelated machines, the schedule represented by the encoding $(3, 8, 7, -1, 9, 2, -2, 6, 4, -3, 1, 5)$ is shown in Fig. 1.

Consequently, the exploration process of $B\&B$ can start from an empty array, and a searching tree can be constructed step by step. Each non-leaf node in this tree (partial schedule) has several successors (child nodes). To construct such a successor, the branching scheme first copies the encoding from its parent. Then it selects one of the feasible numbers, which does not appear in the current encoding, and puts it into the first available position. As a result, all the feasible complete schedules can be represented by leaf nodes.

To increase the efficiency of the searching process, we proposed a truncation rule to avoid unnecessary explorations. If all the negative numbers, but not all the positive ones, have appeared in a current encoding, the problem in that state (i.e., in that search tree node) can be simplified to a single machine (namely, machine M_m) scheduling problem for a smaller instance, containing only unscheduled jobs. Therefore, we can run the dynamic programming method proposed by Potts and Van Wassenhove (1992) for the single machine problem

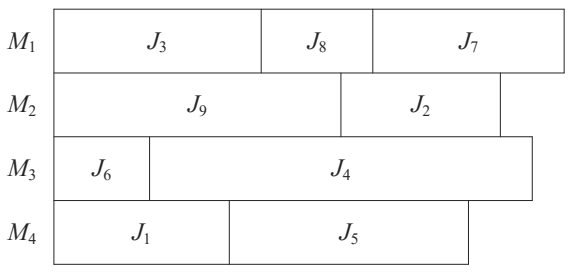


Fig. 1. Illustration of solution encoding.

Algorithm 1. DP_m.

1: Set initial conditions $f(0,\ E_1,\ E_2, \ldots,\ E_m) = 0$ for $0 \le E_i \le d,\ 1 \le i \le m$.
2: Calculate the recurrence function that

$$f(j,\ E_1,\ E_2, \ldots,\ E_m) = \min \begin{cases} f(j-1,\ \max\{0, E_1 - p_{1j}\},\ E_2, \ldots,\ E_m) + \max\{0,\ p_{1j} - E_1\}, \\ f(j-1,\ E_1,\ \max\{0, E_2 - p_{2j}\}, \ldots,\ E_m) + \max\{0,\ p_{2j} - E_2\}, \\ \ldots, \\ f(j-1,\ E_1,\ E_2, \ldots,\ \max\{0,\ E_m - p_{mj}\}) + \max\{0,\ p_{mj} - E_m\} \end{cases}$$

for $1 \le j \le n, 0 \le E_i \le d, 1 \le i \le m$.
3: Determine the optimal total late work as $f(n,\ d,\ d, \ldots,\ d)$.

$1||Y$ to obtain an optimal schedule for the last machine without further explorations. For the sake of completeness we re-call this dynamic programming below, denoted as DP_1 in the remaining part of the paper. Since DP_1 is applied for jobs on machine M_m, in this method the job processing time p_j corresponds to its processing time on machine M_m, i.e., $p_j = p_{mj}$ $(1 \le j \le n)$.

Let $f(j,\ t)$ denote the minimum total late work for the first j jobs, when the last early or partially early one among them is complete at time t $(0 \le t \le b_j,\ b_j = \min\{\sum_{i=1}^{j} p_i,\ \max_{i=1,\ldots,j}\{d_i + p_i - 1\}\})$. Then DP_1 runs according to the recursion process in Algorithm 2, with the time consumption of $O(n \cdot \min\{\sum_{j=1}^{n} p_j,\ \max_{j=1,\ldots,n}\{d_j + p_j\}\})$ as proved by Potts and Van Wassenhove (1992).

To further increase the efficiency of the branch-and-bound algorithm, we propose several dominance rules, which can help us to discard some partial solutions that are dominated by other ones. Suppose that σ is an uncompleted encoding, while h and k are two integers such that $1 \le h, k \le n$ and $h, k \notin \sigma$. Then we can construct two successors of σ, denoted as $\sigma' = \sigma hk$ and $\sigma'' = \sigma kh$. Note that, according to the characteristics of the encoding presented above, $L'_i = L''_i$ for $1 \le i \le m$, where L'_i and L''_i are the loads of machine M_i in σ' and in σ'', respectively.

Let $Y(x)$ be the total late work of a (partial) schedule x. We claim that, if one of the two following conditions is satisfied:

(i) $Y(\sigma') < Y(\sigma'')$,

(ii) $Y(\sigma') = Y(\sigma'')$ and $h < k$,

then σ'' is dominated by σ', which means that σ'' could be removed from the searching tree without further direct exploration.

4.2. Upper bound. At the beginning of the searching process, to estimate the value of the optimal total late work, the initial upper bound has to be determined. For this purpose, we chose the meta-heuristic method proposed for problem $R||Y$ in the previous study on this model (Wang *et al.*, 2020). In the work of Wang *et al.* (2020), two approaches were given: tabu search (TS) and a genetic algorithm (GA). According to the preliminary experiments (in Appendix), we found that, for small size instances (which can be finally solved by the exact methods proposed in this paper), TS and GA shared almost the same performance from the solution quality point of the view. However, TS required less time than GA. Therefore, we chose the tabu search method to generate the initial solution, i.e., the initial upper bound of the $B\&B$ algorithm (see the work of Wang *et al.* (2020) for details on TS).

During the exploration of the solution space, the upper bound should be updated to reflect the quality of the best solution obtained so far. To improve the upper bound, for an uncompleted solution, which is represented by the encoding of a non-leaf node on the searching tree, we construct several feasible (complete) solutions by applying some simple heuristic strategies. If the criterion value of the best solution among these heuristic solutions (i.e., with the minimum total late work) is better than the current upper bound, this bound is replaced.

We propose nine such strategies, all of which are two-step approaches and can be denoted as A-B. In the first step A, we sort the unscheduled jobs in a specific order, while in the latter phase B, we assign the unscheduled jobs to one of the machines according to a specific rule. The rules used to order jobs in phase A are as follows:

- EDD (earliest due date first, i.e., non-decreasing order of d_j),
- SPT (smallest processing time first, i.e., non-decreasing order of $\overline{p_j} = \frac{1}{m}\sum_{i=1}^{m} p_{ij}$,
- LPT (largest processing time first, i.e., non-increasing order of $\overline{p_j}$).

The rules applied in phase B are as follows:

- $MinC$: assign a selected job to a machine with the minimum makespan,
- $MinY$: assign a selected job to a machine to keep the minimum current total late work,

Algorithm 2. DP_1.

1: Set $f(0,\ 0) = 0$ and all other initial values to be infinite.
2: Calculate the recurrence function that
$$f(j,\ t) = \begin{cases} \min\{f(j-1,\ t-p_j) + \max\{t-d_j,\ 0\}, f(j-1,\ t) + p_j\}, & t < d_j + p_j, \\ f(j-1,\ t) + p_j, & t \geq d_j + p_j. \end{cases}$$
for $1 \leq j \leq n$, $0 \leq t \leq d$.
3: Determine the optimal total late work as $\min_{t=0,\ldots,b_n}\{f(n,\ t)\}$.

Algorithm 3. Lower bound.

1: Based on the current encoding, calculate the total late work of the scheduled jobs (denoted as Y_s), as well as the load of each machine (denoted as L_i, $1 \leq i \leq m$). Denote the set of unscheduled jobs as $\mathcal{J}_u$, and let $\hat{d} = \max\limits_{J_j \in \mathcal{J}_u}\{d_j\}$.
2: For an auxiliary instance of problem $Rm|d_j = \hat{d}|Y$ run DP_m to get an optimal solution for a set of jobs $\mathcal{J}_u$ and a common due date $\hat{d}$ on m machines, and to get the optimal value Y_u for this auxiliary instance. Note that the range of parameter E_i in DP_m is re-set to be $0 \leq E_i \leq \max\{\hat{d} - L_i,\ 0\}$ for $1 \leq i \leq m$, since there are some jobs already scheduled on M_i.
3: Return $Y_s + Y_u$ as the lower bound of this partial solution.

- $MinP$: assign a selected job (say J_j) to machine M_k so that $k = \arg\ \min_{1 \leq i \leq m}\{p_{ij}\}$.

Summing up, by combining the above mentioned rules A and B we proposed the following nine heuristics for updating the upper bound within our $B\&B$: $EDD\text{-}MinC$, $EDD\text{-}MinY$, $EDD\text{-}MinP$, $SPT\text{-}MinC$, $SPT\text{-}MinY$, $SPT\text{-}MinP$, $LPT\text{-}MinC$, $LPT\text{-}MinY$ and $LPT\text{-}MinP$.

4.3. Lower bound. In a branch-and-bound algorithm, the lower bound is used to cut the redundant branches and avoid unnecessary explorations. Briefly, if the lower bound of a non-leaf node on the searching tree exceeds the current upper bound, this node, as well as its sub-tree, could be discarded, and direct explorations on it could be omitted.

In this paper, we get the lower bound of an uncompleted solution based on the dynamic programming approach presented in Section 3, i.e., DP_m. For the scheduled jobs, we calculate their total late work based on the current encoding, while for the unscheduled jobs, we reduce the problem to a common due date version and use DP_m to obtain an optimal solution. Finally, the sum of the two mentioned values is returned as the lower bound of this partial solution. More precisely, the lower bound is calculated with the procedure in Algorithm 3.

4.4. Frame of the branch-and-bound algorithm. Combining the techniques described above, we obtain the branch-and-bound approach in Algorithm 4.

5. Enumeration algorithm

In a parallel system, which can be modelled by the discussed problem $R||Y$, a solution could be regarded as the division of all jobs to m disjoint sub-sets, which then are assigned to m machines to be executed separately. In this case, the processing time of J_j on M_i (i.e., p_{ij}) is fixed when the partition is completed. To calculate the total late work in the whole system, we can calculate the criterion values for each machine separately, and then add them together.

Therefore, we propose an enumeration algorithm for problem $R||Y$, called PDP (partition and dynamic programming). The main idea of PDP is as follows.

1. Run an enumeration process to generate all the possible partitions for the jobs.
2. For a particular partition, the original problem $R||Y$ is reduced to m instances of the single machine problem $1||Y$, so we can run DP_1 (Potts and Van Wassenhove, 1992) m times to solve them individually.
3. Add m values of the total late work obtained by DP_1 for particular machines to determine the final criterion value of this partition.
4. Choose the optimal one among all the possible partitions as the final output.

To control the enumeration process, we use an n-element integer array Arr to represent a particular partition, where each element in Arr stands for a job. The value of this element lays in the range of $[1,\ m]$, representing the machine assignment for this job. For example, when $n = 9$ and $m = 4$, the encoding (4, 2, 1, 3, 4, 3, 1, 1, 2) means the following partition:

M_1: $\{J_3, J_7, J_8\}$;

M_2: $\{J_2, J_9\}$;

M_3: $\{J_4, J_6\}$;

Algorithm 4. Branch-and-bound (B&B).

```
Get an initial upper bound by TS, denoted as UB;
Construct an empty queue Q for exploration;
Create a root node with an empty encoding, and push it into Q;
while (Q ≠ φ) do
    Pop the first node (denoted as FN) from Q;
    if (all m-1 negative integers have appeared in FN's encoding) then
        Run DP1 to schedule the remaining jobs optimally on M_m;
        Calculate the criterion value of this completed solution, denoted as Y_f;
        if (Y_f < UB) then
            Replace UB by Y_f;
        end if
    else
        Put FN's successors (if any) into a temporary set T;
        for each node SN in T do
            Calculate SN's upper bound (denoted as c) with the nine heuristics;
            if c < UB then
                Replace UB by c;
            end if
            if (SN's lower bound < UB and SN is not dominated) then
                Put SN into Q for further exploration;
            end if
        end for
    end if
end while
Output UB and its schedule as the final solution;
```

Algorithm 5. Constructor($index$).

```
if (index == n) then
    Calculator(Arr); {Arr is completed, so a particular partition is constructed. Then we Calculate the optimal criterion value for this partition.}
else
    for (i = 1 to m) do
        Arr[index] = i;
        Constructor(index + 1); {We construct Arr by each element.}
    end for
end if
```

M_4: $\{J_1, J_5\}$.

(But the exact orders of the jobs on each machine are not represented.) Then, the enumeration of all the possible partitions could be obtained by the recurrence process in Algorithm 5.

To calculate the optimal criterion value for a particular partition (denoted as P, which is represented by a completed array Arr), we need to run DP_1, proposed by Potts and Van Wassenhove (1992) and recalled in Section 4.1, m times. Unfortunately this process is time-consuming despite the fact that DP_1 is a very fast method. To avoid unnecessary executions of DP_1, a lower bound of P can be used as a truncating method.

For a particular machine M_i $(1 \leq i \leq m)$, we can extend the due dates of the jobs assigned to it to the largest due date among these jobs (say, $\hat{d}_i$). In this way, we reduce the problem on M_i to an instance of $1|d_j = \hat{d}_i|Y$. The optimal criterion value for this instance is equal to $\min\{0,\ L_i - \hat{d}_i\}$, where L_i is the current makespan (workload) of M_i. Therefore, the value of $\min\{0,\ L_i - \hat{d}_i\}$ could be considered a lower bound of the total late work of jobs assigned to machine M_i. More precisely, a lower bound of a particular partition P can be calculated as follows:

$$LB_P = \sum_{i=1}^{m} \min\{0,\ (\sum_{J_j \in M_i} p_{ij} - \max_{J_j \in M_i}\{d_j\})\}, \quad (6)$$

where $J_j \in M_i$ means that job J_j is assigned to machine M_i. Furthermore, to use this lower bound effectively, an upper bound is also needed for comparison. Therefore, similarly as in our $B\&B$, as described in Section 4.2, we run the tabu search method TS by Wang *et al.* (2020) to generate an initial solution, which allows determining an initial upper bound value.

Combining the techniques described above, we obtain the following enumerative algorithm PDP, as shown in Algorithms 6 and 7.

Algorithm 6. Partition and dynamic programming (PDP).

1: Get an initial upper bound by TS, denoted as UB;
2: Constructor(0); {Start the recurrence process. For each completed array Arr, call $Calculator(Arr)$ to determine the criterion value.}
3: Output UB and its schedule as the final solution.

Algorithm 7. Calculator(Arr).

1: Decompose Arr to get a particular partition P;
2: Calculate P's lower bound LB_P; {The process is truncated when $LB_P \geq UB$.}
3: **if** ($LB_P < UB$) **then**
4: Run DP_1 m times to get the optimal criterion value of P, denoted as OPT_P;
5: **if** ($OPT_P < UB$) **then**
6: Replace UB by OPT_P;
7: **end if**
8: **end if**

6. Computational experiments

The efficiencies of all the methods proposed in the paper were evaluated in computational experiments. We used the data set generation scheme from Wang *et al*. (2020), which was originally proposed by Lin *et al*. (2006) to construct the benchmark set for a late work scheduling problem. The job processing time p_{ij} was randomly generated from a uniform distribution over [1, 10]. Then we adopt the mean processing time of J_j, i.e., $\overline{p_j} = \frac{1}{m}\sum_{i=1}^{m} p_{ij}$, to generate the job due dates. After sorting the jobs in the non-decreasing order of $\overline{p_j}$, the k-th job's due date was randomly generated from $(\overline{p_k}, \overline{p_k} + \frac{1}{m\beta}\sum_{h=1}^{k}\overline{p_{n-h+1}}]$, in which $\beta \in \{3,\ 5,\ 7\}$. Parameter β can be treated as the due date tightness controller.

The two exact algorithms proposed in the paper, $B\&B$ and PDP, were implemented with the C++ language in the IDE of Visual Studio 2017. Moreover, we solve the mathematical model IP formulated in Section 2 by Gurobi 9.1.1, through an interface of Visual Studio. All the experiments were performed on a laptop with an Intel Core i7-10510U 1.80 GHz CPU and 16 GB DDR3 RAM.

Since all the exact approaches ($B\&B$, PDP and IP) solve the problem optimally, we focus on their time efficiencies, i.e., on the execution time. Due to the limitation on the problem instance sizes, which can be solved by IP, in the first phase of our experiments we set $m \in \{2,\ 3\}$ and $n \in \{7,\ 8,\ 9,\ 10,\ 11\}$. For each combination of $(m,\ n,\ \beta)$, we generated 30 random instances. The average running time of $B\&B$, PDP and IP for small size instances are shown in Table 1. Note that the results of $B\&B$ and PDP are displayed in milliseconds, while the ones for the mathematical model IP are shown in seconds, since this technique is much slower than the former two methods. Moreover, we omitted parameter β in Table 1, since it has no influence on the mathematical model. Its influence on $B\&B$ and PDP is analysed in the further part of this section separately.

Table 1. Execution time of all approaches for small instances.

m	n	$B\&B$ [ms]	PDP [ms]	IP [s]
2	7	5.5042	0.1456	1.0458
	8	7.0701	0.2309	4.0724
	9	7.9352	0.3728	29.3449
	10	8.9827	0.5492	137.2843
	11	15.0333	0.9035	862.0528
3	7	7.0925	0.5595	0.7985
	8	13.0382	1.4298	3.5893
	9	22.5270	4.3527	12.2574
	10	33.4528	14.5336	52.4253
	11	65.2029	37.2883	505.2788

From the results in Table 1 we can see that the Gurobi solver needs significantly more time than the two exact algorithms proposed in this paper. The experiments showed that the dedicated methods, designed for and adjusted to the specific scheduling problem, are much more efficient than the general purpose mathematical programming solver. Moreover, comparing these two methods, we can observe that for small instances the algorithm PDP based on enumeration was more efficient than the more sophisticated approach $B\&B$ based on branching and bounding.

To further reveal the performances of $B\&B$ and PDP, in the second phase of the computational experiments, we extended the size of problem instances. We generated instances with $n \in \{20,\ 22,\ 24,\ 26,\ 28\}$ when $m = 2$, and $n \in \{15,\ 16,\ 17,\ 18,\ 19\}$ when $m = 3$. Moreover, we constructed instances for $m = 4$, and set n to be in $\{11,\ 12,\ 13,\ 14,\ 15\}$. In the same manner as in the first phase of experiments, we generated 30 instances for each combination of parameters $(m,\ n,\ \beta)$. In Table 2 we show the average execution time of $B\&B$ and PDP (in seconds). Due to the limitation of the computation resources, we restricted the running time of the algorithms to less than 300 seconds.

In order to analyze the results in Table 2 more accurately, we aggregated them from the viewpoints of the instance size resulting from the values n, m and the values of parameter β. Figures 2–4 present the average execution time of the two algorithms with respect to the number of jobs n for the fixed number of machines $m = 2, 3$ or 4, respectively, while Fig. 5 shows the performances of $B\&B$ and PDP for different values of parameter β controlling the due date tightness.

It can be seen from Figs. 2–4 that the execution time of $B\&B$ and PDP both increase with the growth of the instance size, which is easy to explain based on the complexities of both algorithms (which are exponential

Table 2. Execution time of $B\&B$ and PDP for large instances.

m	n	β	$B\&B\,(s)$	$PDP\,(s)$	m	n	β	$B\&B\,(s)$	$PDP\,(s)$	m	n	β	$B\&B\,(s)$	$PDP\,(s)$
2	20	3	5.4903	0.3647	3	15	3	1.5344	1.8069	4	11	3	1.0206	0.5168
		5	5.3339	0.3404			5	1.3426	1.9437			5	0.9825	0.4993
		7	2.4351	0.3206			7	0.5686	1.9150			7	0.6986	0.4760
	22	3	14.1122	1.5182		16	3	2.9218	5.9247		12	3	5.4469	1.9794
		5	5.7445	1.3769			5	2.6444	5.5628			5	4.7871	2.2728
		7	5.6124	1.3194			7	1.3234	5.6538			7	4.6274	2.4861
	24	3	28.8911	6.3401		17	3	13.3276	17.3073		13	3	17.4023	8.9590
		5	5.5365	6.1304			5	1.7117	17.1001			5	16.8323	9.3804
		7	1.4005	6.2543			7	1.5791	17.0923			7	7.7800	9.9208
	26	3	150.7031	27.5088		18	3	22.2063	52.9445		14	3	62.4160	37.0048
		5	41.6710	29.6821			5	13.4402	51.7056			5	30.8229	36.0513
		7	31.9849	38.6486			7	1.8716	52.8370			7	24.7598	36.6852
	28	3	231.6608	68.5266		19	3	28.7785	163.6490		15	3	60.3573	128.6590
		5	39.5873	63.3783			5	24.8810	179.7591			5	57.2731	133.5407
		7	3.6135	62.7664			7	7.9166	198.7653			7	52.5165	140.8052

in n). However, the growth of $B\&B$'s running time is slower, compared with the rate observed for PDP. Therefore, for small instances (e.g., $m = 2$ and $n \leq 28$, or $m = 4$ and $n \leq 14$), PDP runs faster than $B\&B$, whereas when the instances turn to be large, the advantage of $B\&B$ is more visible. Due to the limitation of computation resources, we had to restrict the experiment scale. But it is conjectured that, when there are enough resources to solve large-scale instance, $B\&B$ will be a considerably more effective method than the approach based on enumerating PDP.

However, from Fig. 5 we can find that $B\&B$ is more sensitive to parameter β (that is, to the tightness of job due dates). When β is small (indicating that the range of due dates is wide), $B\&B$ requires more time, while when β is big (indicating tight due dates), $B\&B$ runs faster. On the other hand, the influence of parameter β on PDP is not visible. Its running time slightly grows with β, because tight due dates (i.e., small due dates) influence the quality of the lower bound used in PDP.

7. Conclusions

In the paper we studied the classical problem of scheduling jobs on unrelated parallel machines in order to minimize the total late work for these jobs ($R||Y$). We proposed two exact approaches for this strongly NP-hard problem based on two different strategies: branching and bounding as well as on enumeration combined with dynamic programming. Moreover we formulated the mathematical model for the problem considered. The computational experiments showed that the dedicated methods beat the mathematical model in this case. Using mathematical models is a popular strategy to solve intractable problems nowadays due to their fast implementation. However, the algorithms specialized for a problem, based on its specificity, can be still much more effective. Based on the results of computational experiments, we see that in some cases (for small instances, for example) rather simple search strategies (such as the method based on the enumerating all partitions of jobs to machines) can be competitive to more sophisticated search strategies, e.g., the branch-and-bound method. But, in general, $B\&B$ is the effective approach to solve intractable problems.

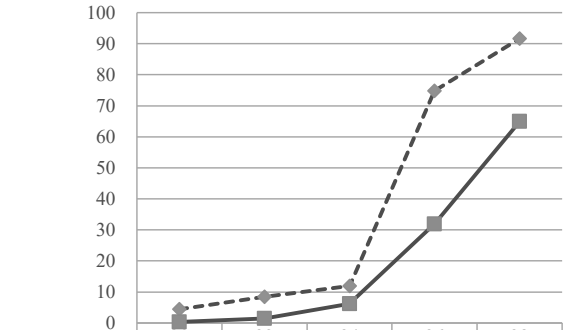
In our future research we would like to focus on the scheduling models with dedicated machines: flow shops (Błażewicz *et al*., 2005; Chen *et al*., 2022) and job shops (Błażewicz *et al*., 2007) with the total late work. Since for these intractable problems dynamic programming approaches are available in the literature, we could use our experience gathered during the reported research to design efficient exact approaches for these models. Moreover, we could consider some generalizations of these due date scheduling problems (e.g., due window assignment (Janiak *et al*., 2013)), or some scenarios for real application (e.g., in transit networks (Liu *et al*., 2022)) with a late work criterion.

Acknowledgment

This research was partially supported by the Foundation Research Project of the Educational Department of Liaoning Province (LJKZZ20220085), the Cooperation Innovation Plan of Yingkou for Enterprise and Doctor (no. 2022-13) and the statutory funds of the Poznan University of Technology (Poland).

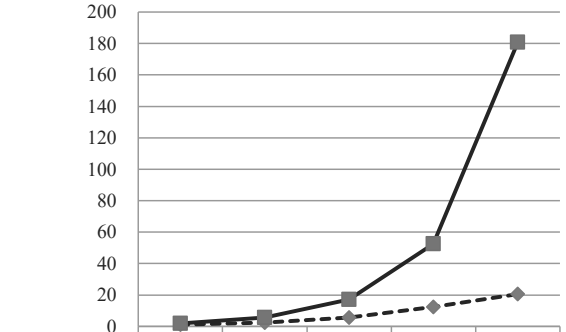
References

Abasian, F., Ranjbar, M., Salari, M., Davari, M. and Khatami, S.M. (2014). Minimizing the total weighted late work in scheduling of identical parallel processors with communication delays, *Applied Mathematical Modelling* **38**(15): 3975–3986.



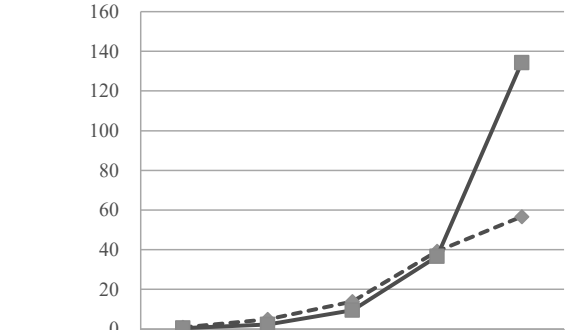
	20	22	24	26	28
B&B (s)	4.4198	8.4897	11.9427	74.7863	91.6205
PDP (s)	0.3419	1.4048	6.2416	31.9465	64.8904

Fig. 2. Execution time of $B\&B$ and PDP with respect to n when $m = 2$.



	15	16	17	18	19
B&B (s)	1.1485	2.2965	5.5395	12.5060	20.5254
PDP (s)	1.8885	5.7138	17.1666	52.4957	180.7245

Fig. 3. Execution time of $B\&B$ and PDP with respect to n when $m = 3$.



	11	12	13	14	15
B&B (s)	0.9006	4.9538	14.0049	39.3329	56.7156
PDP (s)	0.4974	2.2461	9.4201	36.5804	134.3350

Fig. 4. Execution time of $B\&B$ and PDP with respect to n when $m = 4$.

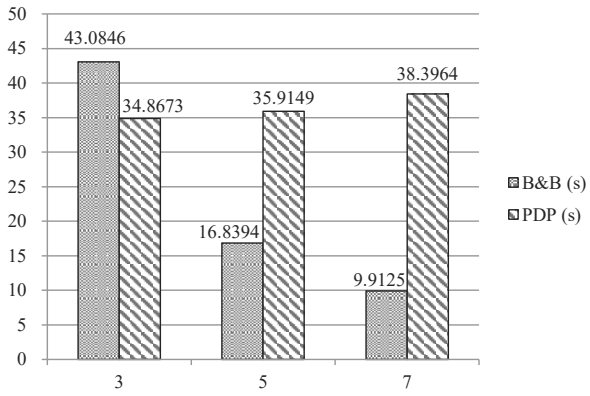


Fig. 5. Execution time of $B\&B$ and PDP with respect to β.

Afzalirad, M. and Rezaeian, J. (2016). Design of high-performing hybrid meta-heuristics for unrelated parallel machine scheduling with machine eligibility and precedence constraints, *Engineering Optimization* **48**(4): 706–726.

Błażewicz, J. (1984). Scheduling preemptible tasks on parallel processors with information loss, *Technique et Science Informatiques* **3**(6): 415–420.

Błażewicz, J., Ecker, K.H., Pesch, E., Sterna, M., Schmidt, G. and Weglarz, J. (2019). *Handbook on Scheduling. From Theory to Practice (2nd Edn)*, Springer, Berlin/Heidelberg/New York.

Błażewicz, J., Pesch, E., Sterna, M. and Werner, F. (2005). The two-machine flow-shop problem with weighted late work criterion and common due date, *European Journal of Operational Research* **165**(2): 408–415.

Błażewicz, J., Pesch, E., Sterna, M. and Werner, F. (2007). A note on the two machine job shop with the weighted late work criterion, *Journal of Scheduling* **10**(2): 87–95.

Chen, X., Liang, Y., Sterna, M., Wang, W. and Błażewicz, J. (2020a). Fully polynomial time approximation scheme to maximize early work on parallel machines with common due date, *European Journal of Operational Research* **284**(1): 67–74.

Chen, X., Miao, Q., Lin, B.M., Sterna, M. and Blazewicz, J. (2022). Two-machine flow shop scheduling with a common due date to maximize total early work, *European Journal of Operational Research* **300**(2): 504–511.

Chen, X., Sterna, M., Han, X. and Błażewicz, J. (2016). Scheduling on parallel identical machines with late work criterion: Offline and online cases, *Journal of Scheduling* **19**(6): 729–736.

Chen, X., Wang, W., Xie, P., Zhang, X., Sterna, M. and Błażewicz, J. (2020b). Exact and heuristic algorithms for scheduling on two identical machines with early work maximization, *Computers & Industrial Engineering* **144**: 106449.

Garey, M.R. and Johnson, D.S. (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W.H. Freeman & Co, New York.

Graham, R., Lawler, E., Lenstra, J. and Kan, A.R. (1979). Optimization and approximation in deterministic sequencing and scheduling: A survey, *Annals of Discrete Mathematics* **5**: 287–326.

Janiak, A., Kwiatkowski, T. and Lichtenstein, M. (2013). Scheduling problems with a common due window assignment: A survey, *International Journal of Applied Mathematics and Computer Science* **23**(1): 231–241, DOI: 10.2478/amcs-2013-0018.

Lin, B.M.T., Lin, F.-C. and Lee, R.C.T. (2006). Two-machine flow-shop scheduling to minimize total late work, *Engineering Optimization* **38**(4): 501–509.

Liu, Z., Yu, B., Zhang, L. and Wang, W. (2022). A hybrid control strategy for a dynamic scheduling problem in transit networks, *International Journal of Applied Mathematics and Computer Science* **32**(4): 553–567, DOI: 10.34768/amcs-2022-0039.

Potts, C.N. and Van Wassenhove, L.N. (1992). Single machine scheduling to minimize total late work, *Operations Research* **40**(3): 586–595.

Sterna, M. (2011). A survey of scheduling problems with late work criteria, *Omega* **39**(2): 120–129.

Sterna, M. (2021). Late and early work scheduling: A survey, *Omega* **104**: 102453.

Sterna, M. and Czerniachowska, K. (2017). Polynomial time approximation scheme for two parallel machines scheduling with a common due date to maximize early work, *Journal of Optimization Theory & Applications* **174**(3): 927–944.

Wang, W., Chen, X., Musial, J. and Blazewicz, J. (2020). Two meta-heuristic algorithms for scheduling on unrelated machines with the late work criterion, *International Journal of Applied Mathematics and Computer Science* **30**(3): 573–584, DOI: 10.34768/amcs-2020-0042.

Xinbo Liu received her BS degree in public management from the Xi'an University of Finance and Economics, China, in 2005 and her MS degree in engineering management from the Liaoning University of Technology, Jinzhou, China, in 2020. She is currently pursuing her PhD degree in global management at the SolBridge International School of Business at Woosong University, Daejeon, South Korea. Her research interests include intelligence optimization in scheduling problems, especially for project scheduling or human resources scheduling.

Wen Wang received her MS degree in software engineering from the Poznan University of Technology, Poland, and her MS degree in computer science and technology from the Liaoning University of Technology, China, both in 2021. Subsequently, she became a PhD student at the School of Business, Sichuan University. Her research interests include intelligent optimization algorithms and their applications, especially for scheduling problems.

Xin Chen received his BS degree in computer science from the Dalian University of Technology, China, in 2005 and his PhD degree in information science from the Poznan University of Technology, Poland, in 2014. He is a full professor at the School of Electronic and Information Engineering in the Liaoning University of Technology, China. His research interests include combinatorial optimization, especially scheduling problems and algorithm design.

Malgorzata Sterna is a full professor at the Poznan University of Technology. Her research interests include scheduling theory, complexity theory, algorithms design, combinatorial optimization and selected aspects of graph theory. She holds a PhD and a habilitation (DSc) in computer science from the Poznan University of Technology.

Jacek Blazewicz is a full professor at the Poznan University of Technology (PUT) and the director of the Institute of Computing Science there. His research interests include algorithm design, computational complexity, scheduling, combinatorial optimization, bioinformatics, and e-commerce. He has a PhD and a habilitation (DSc) in computer science from the Poznan University of Technology. His publication record includes over 400 papers in many outstanding journals. He is also the author and a co-author of over 10 monographs, and an IEEE fellow.

Appendix

To find a suitable initial upper bound for the algorithm $B\&B$ in Section 4.2, we compared the performances of the algorithms TS and GA proposed by Wang *et al.* (2020). With the same experiment setting as in Section 6, i.e., $n \in \{20,\ 22,\ 24,\ 26,\ 28\}$ when $m = 2$, $n \in \{15,\ 16,\ 17,\ 18,\ 19\}$ when $m = 3$, $n \in \{11,\ 12,\ 13,\ 14,\ 15\}$ when $m = 4$, and $\beta \in \{3,\ 5,\ 7\}$ for each pair of $(m,\ n)$, we summarized the results of this preliminary experiment in Table A1. Here the columns "Score" show how often one algorithm beats the other one from the viewpoint of the criterion value, and the columns "Time" reflects the average running time of each algorithm within the experiment (in milliseconds).

Table A1. Comparisons between TS and GA.

TS		GA	
Score	Time [ms]	Score	Time [ms]
124	0.0134	135	0.8275

For each combination of $(m,\ n,\ \beta)$, we generated 30 random instances, i.e., there were 1350 test instances in total. Within these tests, TS won 124 times from the criterion value point of the view, while this number for GA was 135. For the next 1091 tests, both of the algorithms obtained the same criterion value. On the other hand, TS cost less time than GA. Therefore, we chose TS to generate an initial upper bound of $B\&B$.

Received: 10 June 2022
Revised: 25 November 2022
Accepted: 12 January 2023