OR Spectrum
https://doi.org/10.1007/s00291-020-00586-w

REGULAR ARTICLE

Exact and heuristic algorithms for scheduling jobs with time windows on unrelated parallel machines

Giorgi Tadumadze[1] · Simon Emde[2] · Heiko Diefenbach[3]

Received: 23 November 2018 / Accepted: 1 April 2020

Abstract
This paper addresses scheduling a set of jobs with release dates and deadlines on a set of unrelated parallel machines to minimize some minmax objective. This family of problems has a number of applications, e.g., in discrete berth allocation and truck scheduling at cross docks. We present a novel exact algorithm based on logic-based Benders decomposition as well as a heuristic based on a set partitioning reformulation of the problem. We show how our approaches can be used to deal with additional constraints and various minmax objectives common to the above-mentioned applications, solving a broad class of parallel machine scheduling problems. In a series of computational tests both on instances from the literature and on newly generated ones, our exact method is shown to solve most problems within a few minutes to optimality, while our heuristic can solve particularly challenging instances with tight time windows well in acceptable time.

Keywords Scheduling · Unrelated parallel machines · Time windows · Benders decomposition · Berth allocation · Truck scheduling

✉ Simon Emde
siem@econ.au.dk

Giorgi Tadumadze
giorgi.tadumadze@tu-darmstadt.de

Heiko Diefenbach
diefenbach@pscm.tu-darmstadt.de

[1] Fachgebiet Management Science/Operations Research, Technische Universität Darmstadt, Hochschulstraße 1, 64289 Darmstadt, Germany

[2] Department of Economics and Business Economics, Aarhus University, Fuglesangs Allé 4, 8210 Aarhus V, Denmark

[3] Fachgebiet Produktion und Supply Chain Management, Technische Universität Darmstadt, Hochschulstraße 1, 64289 Darmstadt, Germany

Published online: 03 May 2020

1 Introduction

Scheduling problems concern themselves with assigning tasks to limited resources (processors/machines) over time. As such, they play a central role in most logistics and production processes. In this context, this paper deals with variations of the following basic problem.

Given are a set of jobs $J = \{1, \ldots, n\}$, where each job is associated with a release date $r_j \in \mathbb{R}^+$ and a deadline $\overline{d}_j \in \mathbb{R}^+$, and a set of processors (or machines) $P = \{1, \ldots, m\}$, where it takes $p_{ij} \in \mathbb{R}^+$ time units to process job j on machine i. The machines are unrelated, meaning that some jobs may be processed faster on some machines than on others, and preemption is not allowed. On what processor and at what time should each job start processing such that no two distinct jobs are processed at the same time on the same machine, the time windows are observed, and some minmax objective is optimized? In classic machine scheduling literature, using the notation introduced by Graham et al. (1979), this problem is denoted as $[R|r_j, \overline{d}_j|\cdot]$.

A schedule S for $[R|r_j, \overline{d}_j|\cdot]$ is defined as a set of triples $(i, j, t) \in S$, indicating that job $j \in J$ is assigned to processor $i \in P$ starting at time $t \in \mathbb{R}^+$. We say that a schedule is feasible if it meets the following conditions.

- Each job is assigned exactly once, i.e., for each job $j \in J$ there is exactly one triple $(i, j, t) \in S$.
- No job is scheduled before its release date or finishes processing after its deadline, i.e., $\forall (i, j, t) \in S$ it must hold that $r_j \leq t \leq \overline{d}_j - p_{ij}$.
- No two jobs may occupy the same processor at the same time, i.e., for each pair of triples $(i, j, t), (i', j', t') \in S$, $j \neq j'$, it must hold that $i \neq i'$ or $t + p_{ij} \leq t'$ or $t' + p_{i'j'} \leq t$.

Scheduling with time windows on unrelated parallel machines has multiple practical applications. One prime example is berth allocation at maritime container terminals, where calling vessels are assigned time and space at the quay wall. We can equate ships with jobs and berths with processors. Another example is truck scheduling at distribution centers or manufacturing plants, where incoming trucks have to be assigned to dock doors. In this context, incoming trucks correspond to the jobs and dock doors to processors. In both of these applications, it is very common to have given arrival and departure times (i.e., time windows) for each transport device (i.e., vessel or truck). These applications are discussed in more detail in Sect. 2.

Inspired by these applications, we discuss the following generalizations of the base problem.

- *Processing set restrictions* Especially in the berth allocation context, it is possible that certain ships are incompatible with certain berths due to draft limitations (e.g., Xu et al. 2012). In machine scheduling parlance, this means that certain jobs are incompatible with certain machines.
- *Machine availability* Especially in a rolling horizon framework, some machines may not be ready before a given time (e.g., Imai et al. 2001, in the berth scheduling context). In this case, no job may be scheduled on a given machine before the machine is ready.

- *Tails* Jobs may have tails (also called delivery times), where they do not block any machine but still remain active. This is especially relevant for truck scheduling applications, where it may not be enough for a truck to finish processing at a door before the due date, but the shipments need to actually arrive at their destinations (e.g., another door for transshipment) to be considered on-time (e.g., Tadumadze et al. 2019).

We investigate three alternative minmax objectives that are relevant for both of the above-mentioned applications.

- *Makespan* It is typically desirable to quickly clear the terminal (i.e., have the last job finish as soon as possible) for successive planning runs (e.g., Boysen and Fliedner 2010). Consequently, a schedule S is optimal with regard to makespan if it minimizes

$$C(S) = \max_{(i,j,t)\in S} \{t + p_{ij}\}. \tag{1}$$

- *Maximum (weighted) flow time* It is important for terminals to serve every customer (i.e., transport device) in a timely manner. The flow time of a job corresponds to the total service time of the transport device in the terminal (i.e., the time from its arrival until it finishes processing) and is calculated as the difference between its completion time C_j and its release date r_j. In the literature, it is common to minimize the (weighted) sum of flow times $\sum w_j F_j$, where each job $j \in J$ may additionally have a priority weight w_j ($w_j > 0$), indicating its relative importance (e.g., Cordeau et al. 2005; Boysen 2010). While $\sum w_j F_j$ leads to low (weighted) service times for the average customer, it does not preclude long service times for some individual customers. We therefore consider the maximum (weighted) flow time $wF_{\max}$ which aims to minimize the service time of the worst served customer, defined as:

$$wF_{\max}(S) = \max_{(i,j,t)\in S} \{w_j \cdot (t + p_{ij} - r_j)\}.$$

- *Maximum (weighted) lateness* In real life, it is not always possible to achieve a feasible solution without violating any time windows. In such cases, a more useful objective may be to minimize the maximum violation of the due dates. In other words, deadlines $\overline{d}_j$ become due dates d_j and the corresponding constraint becomes soft. The lateness of job j is defined as the difference between its completion time C_j and its due date d_j. For many terminals, it may be reasonable to minimize the maximum (weighted) lateness $wL_{\max}$, defined for our problem as follows:

$$wL_{\max}(S) = \max_{(i,j,t)\in S} \{w_j \cdot (t + p_{ij} - d_j)\}.$$

The main contribution of this paper is a novel logic-based branch and Benders cut scheme, which is shown to solve many realistic problem instances to optimality in reasonable time, decisively outperforming a commercial optimization solver. To the best of our knowledge, it is the first such algorithm to be proposed for this family of unrelated parallel machine scheduling problems with time windows. For even larger

and/or harder instances, we propose a heuristic procedure based on a generalized set partitioning formulation. Our exact and heuristic approaches are very flexible and can be easily adapted to account for problem generalizations and special cases as well as different minmax objectives, such that a broad class of parallel machine scheduling problems with minmax objectives can be solved. Our computational tests show that hundreds of realistically sized instances from both the literature and systematically generated can be solved to optimality in acceptable time.

The remainder of this paper is organized as follows. In Sect. 2, we review the pertinent literature and discuss real-world applications of our scheduling problem in more detail. In Sect. 3, we introduce a novel branch & Benders cut scheme as well as a heuristic column selection algorithm for this problem, respectively. We extend our approaches to cope with additional constraints and alternative minmax objectives in Sect. 4. In Sect. 5, performance of our algorithms is tested on benchmark instances in a computational study. Finally, Sect. 6 concludes the paper.

2 Literature review and applications

In the literature, the problem of scheduling a set of jobs with given release dates and deadlines on unrelated parallel machines to minimize the makespan is expressed by the triple $[R|r_j, \overline{d}_j|C_{\max}]$. For an overview on machine scheduling in general, see Pinedo (2016).

Concerning scheduling on unrelated parallel machines with a makespan objective in particular, it is extensively studied by Lenstra et al. (1990), who show that the approximation ratio for $[R||C_{\max}]$ is at least 3/2 unless $P = NP$. The same authors also develop a polynomial time 2-approximation scheme. An improved $(2 - 1/m)$-approximation scheme with computational time $O(m^2)$ is developed by Shchepin and Vakhania (2005). Furthermore, for the special case where each job can be assigned to at most two machines with the same processing time, Ebenlendr et al. (2014) proposed a 1.75-approximation algorithm. More recently, Knop and Koutecky (2017) introduced a method to solve the problem in $\Theta^{O(\Theta^2)} \cdot n^{O(1)}$ time, where $\Theta = \max_{i \in P, j \in J}\{p_{ij}\}^K$, and K is the number of different types of machines. Mokotoff and Chrétienne (2002) develop an exact and a heuristic algorithm based on a cutting plane scheme to solve $[R||C_{\max}]$. Another successful solution procedure to solve this problem based on a recovering beam search heuristic is proposed by Ghirardi and Potts (2005). In contrast to traditional beam search, their algorithm contains a recovering phase, which allows substitution of dominated solutions by alternative more promising solutions from the previous stage. Moreover, Fanjul-Peyro and Ruiz (2010, 2011) develop heuristics for $[R||C_{\max}]$ based on iterated greedy local search methods and size reduction heuristics. Further metaheuristics for $[R||C_{\max}]$ are presented by Lin et al. (2011) and Sels et al. (2015).

Regarding the inclusion of release dates, Lancia (2000) proposes a branch and bound scheme to schedule jobs with release dates and tails on two unrelated parallel machines, minimizing the makespan $[R2|r_j, q_j|C_{\max}]$. A similar scheduling problem for identical parallel machines subject to job release dates and tails minimizing the

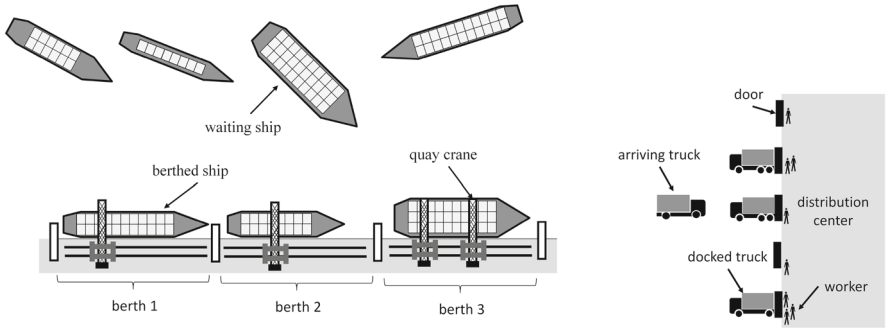
makespan is investigated by Gharbi and Haouari (2002). They develop branch and bound algorithms and propose a new tight branching scheme for $[P|r_j, q_j|C_{\max}]$. In Sect. 4, we discuss how our own algorithms can be extended to account for nonzero delivery times.

Regarding logic-based Benders decomposition for parallel machine scheduling, Jain and Grossmann (2001) propose a decomposition approach which combines solving a mixed-integer linear programming (MILP) model and a constraint programming (CP) model to solve an unrelated parallel machine scheduling problem with time windows to minimize the total cost, which depends on the job-machine assignment. Their approach, which is similar to logic-based Benders decomposition, is tested on instances with up to $n = 20$ jobs and $m = 5$ machines. Tran et al. (2016) study the unrelated parallel machine scheduling problem with sequence and machine-dependent setup times and a makespan objective $[R|sds|C_{\max}]$. They develop a logic-based Benders decomposition approach where the master problem (i.e., assignment of jobs to the machines) is solved via MILP techniques and the subproblems (i.e., sequencing jobs) via a specialized solver for the traveling salesman problem (TSP). In a computational study, instances with up to $n = 60$ jobs and $m = 5$ machines are solved to optimality, while instances with up to $n = 120$ jobs and $m = 8$ machines are solved heuristically. Gedik et al. (2016) develop two logic-based Benders decomposition algorithms for a parallel unrelated machine scheduling problem with sequence dependent setup times. Their problem additionally considers availability intervals, and as each job is associated with a given profit and cost, their objective is to maximize the total profit. The authors compare their decomposition algorithms to extant ILP and CP models for instances with $n \in \{32, 57, 116\}$ jobs and $m \in \{10, 15, 20, 25, 30\}$ machines, where not all instances can be solved to optimality within a 3-hour time limit. Hooker (2007) develops another logic-based Benders decomposition algorithm for several versions of the unrelated parallel machine scheduling problem with regard to three different objectives (makespan, total tardiness and total costs). They additionally consider machine-job specific resource consumption rates c_{ij} and allow more than one job to be processed concurrently on machine i as long as the total resource consumption does not exceed the given resource limit C_i. They solve problems with up to $n = 50$ jobs, where not all instances can be solved to optimality within 2 hours.

In summary, most papers employing logic-based Benders decomposition for parallel machine scheduling use a constraint programming solver for the subproblems, the exception being Tran et al. (2016), who use a TSP solver. In this paper, we develop a novel bounded dynamic programming-based approach, which is shown to perform quite well and can easily be adapted to a broad range of machine scheduling problems (see Sect. 4).

2.1 Berth allocation

Apart from a short dip during the financial crisis of 2008, international seaborne trade has been steadily on the rise for the past decades (UNCTAD 2016). While container vessels are growing more numerous and larger, capacities (e.g., quay cranes and space at the quay) at container terminals are hard to expand due to physical and geographical



(a) Berth allocation. **(b)** Truck scheduling.

Fig. 1 Applications of $[R|r_j, \overline{d}_j|C_{\max}]$

constraints. For this reason, operations research methods have become very prominent in making the best use of limited resources at many container terminals (e.g., Steenken et al. 2004; Vacca et al. 2007). One of the most important problems in this context is the berth allocation problem, which aims to assign a set of calling vessels to both a berthing time and space at the quay wall, which is surveyed in Bierwirth and Meisel (2010, 2015). In this context, our problem is equivalent to the following basic berth allocation problem: Given a set of calling vessels (jobs) and a discrete set of berths (processors), which ship should moor at which berth at what time, such that no vessel berths before its arrival time, every vessel can depart before its deadline, and the last vessel departs as early as possible? In practice, berths may differ in term of the resources (e.g., quay cranes) and each vessel may have an "optimal berthing point" (e.g., depending on the storage location for its containers in the terminal). As a result, the ships' handling times at the quay wall may vary depending on where they are berthed, which makes berths equivalent to unrelated machines. A schematic representation of discrete berth allocation problem is given in Fig. 1a.

In the berth allocation context, minmax objectives have so far only been tackled with regard to very specific applications. The first application of a makespan objective in the berth allocation context is provided by Li et al. (1998), who consider a continuous quay wall, where ships can berth at any arbitrary position as long as they do not collide. Emde et al. (2014) investigate the berth allocation problem at ports that have a so-called mobile quay wall, which can enclose ships berthed at the normal, fixed quay wall. Similarly, the same performance criterion for a combined berth allocation and quay crane scheduling (or assignment) problem is applied by Lee and Qiu Wang (2010) and Blazewicz et al. (2011), respectively.

2.2 Truck scheduling

At many cargo handling facilities, like distribution centers, manufacturing plants or cross docks, calling trucks need to be assigned to a given set of dock doors for (un-)loading. This problem is commonly referred to as truck scheduling, which is surveyed by Boysen and Fliedner (2010). The existing literature almost without exception

assumes that all dock doors process trucks equally fast, which is not always the case (e.g. Tadumadze et al. (2019); Konur and Golias (2017)). Since dock doors may differ with regard to staff levels and equipment, processing times may also depend on which truck is processed at which door. Critical trucks can be processed faster at dock doors with more workers, although not every truck profits equally from additional workers (depending on size and load). In this regard, dock doors can be interpreted as unrelated parallel machines. It is very common that trucks must be scheduled within fixed time windows. A distribution center handling incoming trucks is schematically depicted in Fig. 1b.

3 Solution procedures

In this section, we present novel exact and heuristic algorithms to solve $[R|r_j, \overline{d}_j|C_{\max}]$. We later generalize these approaches to include additional constraints (Sect. 4.1) and alternative minmax objectives (Sects. 4.2.1, 4.2.2). The exact algorithm, based on logic-based branch and Benders cut, is presented in Sect. 3.1. Since machine scheduling with time windows is well known to be strongly NP-hard for any $|P| \geq 1$ (Lenstra et al. 1977), we also propose a heuristic column selection approach in Sect. 3.2.

3.1 Branch and Benders cut for $[R|r_j, \overline{d}_j|C_{\max}]$

Our exact logic-based branch & Benders cut (B&BC) algorithm is based on the classic Benders decomposition. The original algorithm proposed by Benders (1962) decomposes a problem into a master model, which is usually a relaxed version of the original model, and a slave model, which is used to iteratively generate feasibility and optimality cuts to be added to the master model. In our approach, we deviate from the classic Benders scheme by adding so-called logic-based (or combinatorial) Benders cuts in the spirit of Codato and Fischetti (2006) and Hooker (2011). Our master model is a type of bin packing problem with the goal of assigning jobs to processors. The slave problem consists of sequencing a given set of jobs on given processors, where the assignment of jobs to processors is determined by the current master solution, such that the makespan is minimal.

3.1.1 Master problem

Our master model contains only binary variables y_{ij}, which define the assignment of jobs to machines: $y_{ij} = 1$ if and only if job j is assigned to machine i. Moreover, we define $N(j) = \{j' \in J \mid r_{j'} \geq r_j\}$ as the set of jobs that are released no sooner than job j and $\overline{d}_{\max}(j) = \max\{\overline{d}_{j'} \mid j' \in N(j)\}$ as the latest deadline of the jobs from $N(j)$. Finally, let z be an auxiliary continuous variable denoting a lower bound on the completion time of the latest machine. Our master model is a relaxed version of the original problem, where we assume that a no-wait earliest release date (ERD)

schedule is feasible, which it may not be. It is defined as the following mixed integer linear program.

$$\text{[Master] Minimize } F^M(Y, Z) = z \tag{2}$$

subject to

$$\sum_{i \in P} y_{ij} = 1 \qquad \forall j \in J \tag{3}$$

$$r_j + \sum_{j' \in N(j)} y_{ij'} \cdot p_{ij'} \leq z \qquad \forall i \in P;\ j \in J \tag{4}$$

$$r_j + \sum_{j' \in N(j)} y_{ij'} \cdot p_{ij'} \leq \overline{d}_{\max}(j) \qquad \forall i \in P;\ j \in J \tag{5}$$

$$y_{ij} \in \{0; 1\} \qquad \forall i \in P,\ j \in J \tag{6}$$

Objective (2) minimizes the makespan of the relaxed problem. Constraints (3) make sure that each job is assigned to exactly one machine. Valid inequalities (4) enforce that the lower bound z on the makespan cannot be less than lower bound on the makespan of each processor $i \in P$, which is the total processing time of the jobs assigned to processor i starting after time r_j. Valid inequalities (5) forbid such job-to-machine assignments that obviously lead to the violation of deadlines, i.e., when even the lower bound on the makespan on a machine exceeds a potential latest deadline $\overline{d}_{\max}(j)$ of the jobs on that machine. Constraints (6) define the binary variables. Note that it is possible to solve the model as a pure feasibility problem, i.e., without variable z and valid inequalities (4) and (5). However, it is expedient to add these valid inequalities to drive the search into the promising regions of the solution space.

We solve [Master] using a commercial black box solver. Whenever the solver hits upon a candidate integer solution $\bar{Y}$, the slave problem is solved to derive a feasible solution from $\bar{Y}$ (if any) by sequencing the jobs optimally for the given assignment. Therefore, from the viewpoint of the slave problem, the assignment of jobs to processors is already given; what remains is determining the optimal schedule of jobs on each machine.

3.1.2 Slave problem

To solve the slave problem, first off, the problem clearly decomposes along the processors: how we schedule jobs assigned to one processor does not affect how we schedule jobs assigned to another processor. Given an integer solution $\bar{Y}$ for model [Master], the set of jobs assigned to machine i is $\Gamma_i = \{j \in J \mid \bar{y}_{ij} = 1\}$. Now, scheduling jobs Γ_i on machine i is equivalent to solving a single machine scheduling problem with time windows to minimize the makespan, i.e., $[1|r_j, \overline{d}_j|C_{\max}]$. This problem is well known to be NP-hard in the strong sense (Lenstra et al. 1977). It stands to reason, however, that at least in the applications discussed in Sect. 2, the number of jobs per processor will not be too high. Therefore, we propose the following bounded dynamic

programming (BDP) scheme based on the general methodology introduced by Held and Karp (1962).

The dynamic program consists of $|\Gamma_i|+1$ stages (index $l = 0, \ldots, |\Gamma_i|$), each stage containing states (Σ), where $\Sigma \subseteq \Gamma_i$ is the set of jobs already scheduled. Starting in dummy stage $l = 0$ from state $(\emptyset)$, successors in stage $l+1$ are reached by appending one of the not yet scheduled jobs to the schedule, i.e., the successor states of some state (Σ) are $(\Sigma \cup \{j\})$, $\forall j \in \Gamma_i \setminus \Sigma$. Let $\mathcal{C}(\Sigma)$ be the makespan of the (partial) schedule of state (Σ) which can be recursively computed as

$$\mathcal{C}(\Sigma) = \min_{j \in \Sigma} \{\max\{\mathcal{C}(\Sigma \setminus \{j\}), r_j\} + p_{ij}\},$$

where $\mathcal{C}(\emptyset) := 0$. Further, for each state (Σ), we compute the lower bound on the makespan $LB(\Sigma)$ as follows:

$$LB(\Sigma) = \max_{j \in \Gamma_i \setminus \Sigma} \left\{ \max\{\mathcal{C}(\Sigma), r_j\} + \sum_{j' \in N(j) \cap \{\Gamma_i \setminus \Sigma\}} p_{ij'} \right\},$$

the idea being that the remaining jobs are appended in a no-wait ERD schedule, which is optimal but may be infeasible due to the deadlines.

In the dynamic programming graph, the number of nodes (i.e., states) increases exponentially with the number of jobs to be scheduled. However, we can significantly reduce its size by generating only such nodes that satisfy the following criteria.

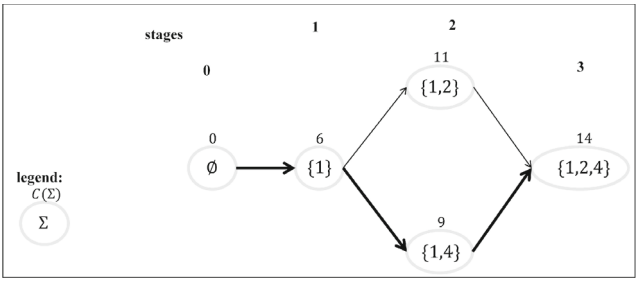
1. It is still possible to schedule every remaining job with regard to its respective time window, i.e., $\mathcal{C}(\Sigma) + p_{ij} \leq \overline{d}_j$, $\forall j \in \Gamma_i \setminus \Sigma$, must hold.
2. The remaining jobs can at least theoretically still be feasibly scheduled without violating the latest deadline, i.e., $\mathcal{C}(\Sigma) + \sum_{j \in \Gamma_i \setminus \Sigma} p_{ij} \leq \max_{j \in \Gamma_i \setminus \Sigma} \{\overline{d}_j\}$ must hold.
3. Let UB be the objective value of the best known feasible solution, i.e., the global upper bound. Then it must hold that $LB(\Sigma) < UB$.

Moreover, if the optimal solution of the relaxed problem, obtained while computing $LB(\Sigma)$, is feasible for the original problem (i.e., every job j from Γ_i is scheduled within its time window), we fathom the state and, if applicable, store the found solution as the new best known feasible solution and the corresponding objective value as the local upper bound $UB(\Sigma)$ on this machine. If the local upper bound of the state is smaller than the global upper bound (i.e., if $UB(\Sigma) < UB$ holds), we replace UB with $UB(\Sigma)$ in criterion 3.

The optimal makespan of processor i given Γ_i equals $\mathcal{C}(\Gamma_i)$. The corresponding sequence of jobs is obtained by backward recovery along the optimal path. To determine the start times of the jobs, each job must be scheduled in the given sequence as early as possible, i.e., either when its predecessor ends or at its release date, whichever comes last. If no final state (Γ_i) exists—either because it is impossible to schedule the jobs in Γ_i without violating any time windows or because there is no schedule with a makespan of less than UB —we consider the "optimal makespan" for this processor to be $\mathcal{C}(\Gamma_i) := UB$.

j	1	2	3	4
$p_{1,j}$	5	6	8	4
$p_{2,j}$	5	5	9	3
r_v	1	3	4	2
d_j	8	19	13	25

(a) Example problem data.



(b) Dynamic programming graph for the slave problem in the example.

Fig. 2 Example data and dynamic programming

Regarding the asymptotic runtime, BDP has $O(2^{|\Gamma_i|})$ states and $O(|\Gamma_i|2^{|\Gamma_i|})$ transitions, making the runtime exponential. However, as mentioned above, in many applications, the number of jobs per processor $|\Gamma_i|$ can be expected to be rather low; hence, solution times may still be acceptable for many practical uses.

Example: Consider the example data given in Table 2a, consisting of $n = 4$ jobs and $m = 2$ processors. Assume that for some [Master] solution $\Gamma_2 = \{1, 2, 4\}$, i.e., jobs 1, 2, and 4 are assigned to processor 2. The resulting dynamic programming graph is in Fig. 2b. One of the two optimal solutions is bold, corresponding to job sequence $\langle 1, 4, 2]$.

Let $\mathcal{C}(\Gamma_i)$ be the optimal makespan of machine i given Γ_i, determined for each machine via BDP. Let $i^* = \arg\max_{i \in P}\{\mathcal{C}(\Gamma_i)\}$ be the machine with the greatest makespan and hence the machine that determines the makespan for the schedule as a whole. If $\mathcal{C}(\Gamma_{i^*})$ is lower than the current best upper bound UB, then a new best solution has been found. The solution is stored and the upper bound is updated, i.e., $UB := \mathcal{C}(\Gamma_{i^*})$. Moreover, the following cut is added to program [Master]:

$$z \leq \mathrm{UB} - \epsilon,$$

where ϵ is a sufficiently small positive number. This way, solutions to model [Master] whose lower bound z is not less than the best upper bound UB will be fathomed. Note that the upper bound can initially be set to $UB := \max_{j \in J}\{\overline{d}_j\}$ or to the best objective value of some heuristic procedure.

Regardless of whether a new upper bound has been found, we determine the set $I = \{i \in P \mid \mathcal{C}(\Gamma_i) \geq UB\}$. To find a better solution with a lower makespan, at least one of the jobs on each of the machines in I must be moved to another machine. To enforce this, we add the following cuts to program [Master].

$$\sum_{j \in \Gamma_i} y_{ij} \leq |\Gamma_i| - 1 \qquad \forall i \in I.$$

Once the cuts are added as constraints, the solver continues solving model [Master] with these new cuts. When it hits upon another integer solution, the slave problem is again solved, new cuts are generated, and so on, until no more unexplored and

Table 1 Additional notation for the GSPP model

M	Set of columns
T	Number of periods (index $t = 1, \ldots, T$)
v_m	Binary variable: 1, column m is selected; 0, otherwise

unfathomed solutions remain for the master model. At that time, the search terminates and the best upper bound is optimal. This procedure is commonly referred to as (logic-based) branch & Benders cut (Rahmaniani et al. 2017; Emde 2017). In Sect. 5.2, we investigate whether it is beneficial to add cuts as so-called lazy or regular constraints.

3.2 Heuristic column selection based on generalized set partitioning

As a first step to solve $[R|r_j, \overline{d}_j|C_{\max}]$ heuristically, we formulate the problem as a generalized set partitioning problem (GSPP). This model formulation assumes all time related parameters (i.e., r_j, $\overline{d}_j$, and p_{ij}) to be integer. For practical purposes, this assumption can be imposed by scaling and rescaling the time unit before and after using the solution procedure, respectively. We divide the entire planing horizon in discrete time periods $t = 1, \ldots, T$. Note that $T = \max_{j \in J}\{\overline{d}_j\}$ is a sufficiently large value for the length of time horizon. Once the $[R|r_j, \overline{d}_j|C_{\max}]$ problem is formulated as a discrete time-indexed scheduling problem, it can be straightforwardly transformed to a GSPP as described below. Similar problem formulations and techniques have also proven successful, for instance, for discrete berth allocation (Buhrkal et al. 2011; Lalla-Ruiz et al. 2016) and for truck scheduling problems (Boysen et al. 2017; Tadumadze et al. 2019).

In the GSPP model, a column represents a feasible assignment of a job j to a machine i starting in period t. Let $M = \{(i, j, t) \in P \times J \times \{1, \ldots, T\} \mid r_j \leq t \leq \overline{d}_j - p_{ij}\}$ be the set of all columns. The following restrictions have to be considered to derive a feasible partition:

- For each job $j \in J$ exactly one column (i, j, t) from set M has to be selected, and
- On each processor $i \in P$ and in each period t ($t = 1, \ldots, T$), at most one job j can be processed.

Among all feasible partitions, we seek one that minimizes the makespan.

We define a binary variable $v_m \in \{0, 1\}$, which is 1 if column $m = (i, j, t)$ is selected. This way, the problem reduces to selecting one column m for each job j while taking into account that in no period t more than one job is processed on each processor i. Additional notation for the GSPP model formulation is summarized in Table 1. We formulate our GSPP as the following integer program.

$$[\text{GSPP}]\ \text{Minimize}\ G(V) = \max_{(i,j,t)=m \in M}\{v_m \cdot (t + p_{ij})\} \tag{7}$$

subject to

$$\sum_{\substack{(i,\, j',\, t) = m \in M\,: \\ j' = j}} v_m = 1 \qquad \forall j \in J \tag{8}$$

$$\sum_{\substack{(i',\, j,\, t') = m \in M\,: \\ i' = i \,\wedge\, t' \le t < t' + p_{ij}}} v_m \le 1 \qquad \forall t = 1, \ldots, T, i \in P \tag{9}$$

$$v_m \in \{0; 1\} \qquad \forall m \in M \tag{10}$$

Objective function (7) minimizes the makespan by minimizing the completion time of the latest job. Constraints (8) ensure that for each job exactly one column is selected. Furthermore, inequalities (9) take care that in no period more than one job is processed on the same processor. Finally, (10) sets the domain of the binary variables.

Finding a feasible solution to GSPP is strongly NP-hard (Garey and Johnson 1979). However, the GSPP is an extensively studied problem and it offers some modeling advantages. This formulation allows us to heuristically reduce M to a small subset of preselected columns. For all $i \in P$ and $j \in J$, let $M_{ij} = \{(i', j', t) \in M \mid i = i' \wedge j = j'\}$ be the subset of columns relevant for job j on machine i. As a simple approach to heuristically reduce M_{ij}, we only select the first and every μth column from the sorted set of columns, where they are sorted in ascending order according to their completion time $t + p_{ij}$, and μ is a predefined integer number ($\mu \ge 1$). As a result, the number of columns reduces to $|M_{ij}| = \left\lceil \frac{\overline{d}_j - r_j - p_{ij} + 1}{\mu} \right\rceil$. This way, by varying the value of parameter μ we can reduce the solution space to the desired size: A greater value of μ reduces $|M_{ij}|$ and the GSPP becomes easier to solve. On the other hand, smaller $|M_{ij}|$ increase the risk that the optimal (or promising) columns are not considered. Note that our heuristic does not guarantee finding a feasible solution (especially if μ has a high value). In such a case, μ must be reduced to a lower value. We conduct our own series of tests with regard to the best value of μ in Sect. 5.2. Also note that if $\mu = 1$, model [GSPP] is equivalent to the original problem (assuming that all parameters are integer). We further investigate this trade-off in our computational study (Sect. 5.2). Note that this simple heuristic column selection scheme is shown to be very successful in related problems and superior among other heuristic rules studied by Tadumadze et al. (2019).

Once the GSPP model is solved, we derive the job sequences for all machines and improve the machine schedules by starting all jobs as early as possible considering the given sequence. To do so, we set the start time of each job either to its release date or to the completion time of its predecessor job on the same machine. The ensuing makespan may be lower than the original GSPP objective value.

4 Extensions

The proposed algorithms can be easily modified for a broader class of parallel machine scheduling problems. In this section, some extensions of our approaches are presented. First, we discuss how the algorithms have to be adjusted if the solution space of the problem is changed, i.e., the problem contains some new or misses some existing constraints. Afterwards, we generalize our approaches for other minmax objectives than the makespan.

4.1 Special cases and generalizations

Since unrelated machine scheduling is a generalization of parallel machine scheduling, the proposed approaches can be also applied for solving its special cases: related (or uniform) machine scheduling with time windows $[Q|r_j, \overline{d}_j|C_{\max}]$ and identical parallel machine scheduling with time windows $[P|r_j, \overline{d}_j|C_{\max}]$. In the case of relaxed deadlines, i.e., $\overline{d}_j = \infty, \forall j \in J$, the problem becomes $[R|r_j|C_{\max}]$, which is still NP-hard in the strong sense (Garey and Johnson 1979). However, in this case, for a given set Γ_i of jobs to be processed on machine i, the slave problem of our B&BC approach becomes $[1|r_j|C_{\max}]$, which can be solved as a single machine scheduling problem with the goal of minimizing the maximum lateness, i.e., $[1||L_{\max}]$, in polynomial time (Lawler 1973). Analogously, the slave problem of special case $[R|\overline{d}_j|C_{\max}]$ (i.e., when release dates are not considered) becomes $[1|\overline{d}_j|C_{\max}]$ and can be solved to optimality by applying the earliest deadline rule (EDD). Note that the set $N(j) := \{j' \in J|\overline{d}_j \leq \overline{d}_{j'}\}$ for [Master] has to be redefined as the set of jobs whose deadlines are not earlier than $\overline{d}_j$. If neither release dates nor deadlines are considered, (i.e. $r_j = 0$ and $\overline{d}_j = \infty$, $\forall j \in J$), the resulting problem $[R||C_{\max}]$ reduces to the decision of assigning jobs to processors. Sequencing jobs on the processors (i.e., the slave problem) becomes trivial as any no-wait schedule is optimal. It hence suffices to only solve the master model.

Regarding generalizations, our approaches allow us to easily include some application specific constraints. For example, we can consider so-called *processing set restrictions*, surveyed by Leung and Li (2008). According to this restriction, for each job j there is a given set of incompatible processors $G(j) \subset P$ on which job j cannot be processed. For instance, in a berth allocation context, some ships may not be compatible with certain berths due to insufficient water depth (e.g., Tong et al. 1999). The corresponding problem is denoted as $[R|r_j, \overline{d}_j, M_j|C_{\max}]$. This restriction can be considered by adding constraints $y_{ij} = 0, \forall j \in J; i \in G(j)$, to [Master]. For the GSPP heuristic, the processing set constraint can be considered by generating sets of columns M_{ij} only for compatible machine and job pairs (i.e., $M_{ij} = \emptyset, \forall j \in J, i \in G(j)$).

Our approaches can also be easily extended to include *machine availability* restrictions, i.e., for the case when each processor i cannot process any job before its start time s_i or after its end time e_i. In the scheduling literature, the resulting problem is denoted as $[R, NC|r_j, \overline{d}_j|C_{\max}]$ (e.g., Sanlaville and Schmidt 1998). Due to its practical relevance, many versions of existing discrete berth allocation problems consider berth availability times (e.g., Imai et al. 2001; Cordeau et al. 2005; Buhrkal et al. 2011).

Especially, consideration of machine start times enables using a rolling scheduling environment.

For B&BC, the machine availability times can be taken into account by modifying inequalities (4) and (5) as $\max\{r_j, s_i\} + \sum_{j' \in N(j)} y_{ij'} \cdot p_{ij'} \leq z$, $\forall i \in P; j \in J$, and $\max\{r_j, s_i\} + \sum_{j' \in N(j)} y_{ij'} \cdot p_{ij'} \leq \min\{\overline{d}_{\max}(j), e_i\}$, $\forall i \in P; j \in J : s_i \leq \overline{d}_{\max}(j) \wedge e_i \geq r_j$, respectively. This tightens the current time window of job j on machine i if the machine availability times are more limiting than the job time windows. Analogously, BDP for the slave problem on a given machine i can be adjusted by setting $\mathcal{C}(\emptyset) := s_i$ and tightening criteria 1 and 2 from Sect. 3.1.2 as $\mathcal{C}(\Sigma) + p_{ij} \leq \min\{e_i; \overline{d}_j\}$, $\forall j \in \Gamma_i \backslash \Sigma$ and $\mathcal{C}(\Sigma) + \sum_{j \in \Gamma_i \backslash \Sigma} p_{ij} \leq \min\{e_i, \max_{j \in \Gamma_i \backslash \Sigma}\{\overline{d}_j\}\}$, respectively.

For the GSPP heuristic, the set M of all columns has to contain only such columns that satisfy both job time window and machine availability-related restrictions (i.e., $M = \{(i, j, t) \in P \times J \times \{1, \ldots, T\} \mid \max\{r_j; s_i\} \leq t \leq \min\{\overline{d}_j, e_i\} - p_{ij}\}$).

We can further extend our problem by considering so-called *tails* (or delivery times) when, for each job $j \in J$, a nonzero delivery time q_j is given. Tail q_j is defined as the amount of time which job j has to spend in the system after completion on a machine. During the delivery step, job j does not occupy any machine but still affects the makespan, i.e., objective function (1) becomes $C(S) = \max_{(i,j,t) \in S}\{t + p_{ij} + q_j\}$.

To modify B&BC for this extension, we add the smallest delivery time $q_{min} = \min_{j \in J}\{q_j\}$ to the left-hand sides of inequalities (4) and (5) of [Master] to tighten the valid inequality. Moreover, BDP for the slave problem is adjusted as follows. We extend the state space by considering states $(\Sigma, \mathcal{C}(\Sigma))$, i.e., a state is defined by the set Σ of jobs already scheduled and the makespan $\mathcal{C}(\Sigma)$, when the last job leaves the machine. Each state $(\Sigma, \mathcal{C}(\Sigma))$ has successor states $(\Sigma \cup \{j\}, \max\{\mathcal{C}(\Sigma), r_j\} + p_{ij})$, $\forall j \in \Gamma_i \backslash \Sigma$. Let $\Xi(\Sigma, \mathcal{C}(\Sigma))$ be the set of states from which a transition to state $(\Sigma, \mathcal{C}(\Sigma))$ exists. We calculate the objective value of a state as

$$\mathcal{C}^{\text{tails}}(\Sigma, \mathcal{C}(\Sigma)) = \min_{(\Sigma', \mathcal{C}(\Sigma)') \in \Xi(\Sigma, \mathcal{C}(\Sigma))} \{\max\{\mathcal{C}^{\text{tails}}(\Sigma', \mathcal{C}(\Sigma)'); \max\{\mathcal{C}(\Sigma)', r_j\} + p_{ij} + q_j\}\},$$

which is the completion time, including the tails, of the (partial) schedule (note that $j = \Sigma \backslash \Sigma'$). Conditions 1 through 3 are similarly adapted to account for tails. Note that if the deadlines are relaxed (i.e., $\overline{d}_j = \infty$, $\forall j \in J$), then the slave problem $[1|r_j, q_j|C_{\max}]$ can be solved by the procedure proposed by Carlier (1982).

To modify our GSPP heuristic, we substitute the objective function of the GSPP model (Eq. (7)) with objective function minimize $G(V) = \max_{(i,j,t)=m \in M}\{v_m \cdot (t + p_{ij} + q_j)\}$. Moreover, we only consider columns that satisfy the deadlines, i.e., $M = \{(i, j, t) \in P \times J \times \{1, \ldots, T\} \mid r_j \leq t \leq \overline{d}_j - p_{ij} - q_j\}$.

4.2 Alternative minmax objectives

Apart from changing the constraint set, our approaches can also be used for different minmax objectives. Representatively, we discuss two alternative performance criteria,

namely the maximum (weighted) flow time and the maximum (weighted) lateness, in the following.

4.2.1 Minimizing the weighted maximum flow time

To adjust [Master] for the case when the maximum weighted flow time is to be minimized, we define sets $N_k(j) \subseteq N(j)$, containing the k jobs with the earliest release dates from $N(j)$ ($k \in \{1, \ldots, |N(j)|\}$). We replace valid inequalities (4) of [Master] with (11)

$$\min_{j' \in N_k(j)} \{w_{j'}\} \cdot \left(r_j + \sum_{j' \in N_k(j)} y_{ij'} \cdot p_{ij'} - \max_{j' \in N_k(j)} \{r_{j'}\} \right) \leq z \quad \forall i \in P;\ j \in J;\ k \in \{1, \ldots, |N(j)|\}. \tag{11}$$

Valid inequalities (11) set the value for the lower bound z on the maximum flow time by relaxing deadlines and weights. If the deadlines and weights of the jobs are not considered, an ERD ordering of jobs is optimal, which is what underlies inequalities (11): If the k successors of some job j are processed in ERD order, they cannot finish processing sooner than $r_j + \sum_{j' \in N_k(j)} y_{ij'} \cdot p_{ij'}$. The latest release date of these jobs is $\max_{j' \in N_k(j)}\{r_{j'}\}$; hence, the maximum flow time on processor i cannot be less than the left side of inequality (11).

The corresponding slave problem $[1|r_j, \overline{d}_j|wF_{\max}]$ for processor i and a given set Γ_i can be solved by adapting our BDP. A state is characterized by tuple $(\Sigma, \mathcal{C}(\Sigma))$, where Σ is the set of completed jobs and $\mathcal{C}(\Sigma)$ is the completion time of the state. Starting from dummy state $(\emptyset, 0)$, there are transitions from some state $(\Sigma', \mathcal{C}(\Sigma)')$ to successor states $(\Sigma, \mathcal{C}(\Sigma))$, such that $\Sigma = \Sigma' \cup \{j\}$ and $\mathcal{C}(\Sigma) = \max\{\mathcal{C}(\Sigma)', r_j\} + p_{ij}$, $\forall j \in \Gamma_i \backslash \Sigma'$.

Let $\Xi(\Sigma, \mathcal{C}(\Sigma))$ be the set of all states from which a transition to $(\Sigma, \mathcal{C}(\Sigma))$ exists. The (partial) objective value of a state $(\Sigma, \mathcal{C}(\Sigma))$ is

$$\mathcal{F}_{\max}(\Sigma, \mathcal{C}(\Sigma)) = \min_{(\Sigma', \mathcal{C}(\Sigma)') \in \Xi(\Sigma, \mathcal{C}(\Sigma))} \{\max\{\mathcal{F}_{\max}(\Sigma', \mathcal{C}(\Sigma)');\ w_j \cdot (\mathcal{C}(\Sigma) - r_j)\}\},$$

where $j = \Sigma \backslash \Sigma'$, and the objective value of the initial state is $\mathcal{F}_{\max}(\emptyset, 0) := 0$.

Moreover, the calculation of the lower bound of state $(\Sigma, \mathcal{C}(\Sigma))$ is modified as follows:

$$LB(\Sigma, \mathcal{C}(\Sigma)) = \max \left\{ \mathcal{F}_{\max}(\Sigma, \mathcal{C}(\Sigma));\ \max_{j \in \Gamma_i \backslash \Sigma; k \in \{1, \ldots, |N(j)|\}} \left\{ \min_{j' \in N_k(j) \cap (\Gamma_i \backslash \Sigma)} \{w_{j'}\} \cdot \left(\max\{\mathcal{C}(\Sigma); r_j\} + \sum_{j' \in N_k(j) \cap (\Gamma_i \backslash \Sigma)} p_{ij'} - \max_{j' \in N_k(j) \cap (\Gamma_i \backslash \Sigma)} \{r_{j'}\} \right) \right\} \right\}.$$

We say that a state $(\Sigma', \mathcal{C}(\Sigma)')$ is dominated by a different state $(\Sigma, \mathcal{C}(\Sigma))$ if the following criteria are satisfied:

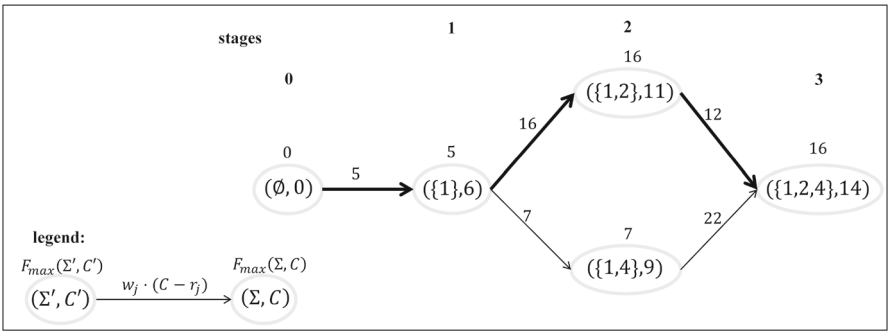


Fig. 3 Dynamic programming graph for the slave problem in the example minimizing $wF_{\max}$

- $\Sigma = \Sigma'$, i.e., both states consider the same set of completed jobs,
- $\mathcal{C}(\Sigma) \leq \mathcal{C}(\Sigma)'$, i.e., the makespan of $(\Sigma, \mathcal{C}(\Sigma))$ is not longer than that of $(\Sigma', \mathcal{C}(\Sigma)')$, and
- $\mathcal{F}_{\max}(\Sigma, \mathcal{C}(\Sigma)) \leq \mathcal{F}_{\max}(\Sigma', \mathcal{C}(\Sigma)')$, i.e., the weighted maximum flow time of the dominating state is at least as good as that of the dominated state.

Dominated states do not have to be considered for the next stage of BDP since they will never lead to any improvement. Otherwise, the BDP can run as described in Sect. 3.1.2.

Example (cont.): Consider the example from Sect. 3.1.2, where a slave problem for processor $i = 2$ with $\Gamma_2 = \{1, 2, 4\}$ is to be solved. Additionally, the following weights for jobs 1, 2, and 4 are given: $w_1 = 1$, $w_2 = 2$, $w_4 = 1$. The resulting DP graph for minimizing the weighted maximum flow time on machine 2 is shown in Fig. 3. The optimal solution is bold, corresponding to job sequence $\langle 1, 2, 4]$.

To adjust our GSPP heuristic, only objective function (7) of the GSPP model has to be substituted by (12).

$$\text{Minimize } G(V) = \max_{(i,j,t)=m\in M} \{w_j \cdot v_m \cdot (t + p_{ij} - r_j)\} \tag{12}$$

4.2.2 Minimizing maximum weighted lateness

Similar considerations as in the previous section can be applied for minimization of the *maximum weighted lateness* $wL_{\max}$.

To modify model [Master] for $[R|r_j|wL_{\max}]$, for each job j, we define the set $V(j) = \{j' \in J | d_{j'} \leq d_j\}$ of jobs whose due dates are not later than the due date of job j. Further, we define sets $V_k(j) \subseteq V(j)$, containing the k jobs with the latest due dates from $V(j)$ ($k \in \{1, \ldots, |V(j)|\}$). This way we can replace valid inequalities (4) of [Master] with (13)

$$\min_{j'\in V_k(j)} \{w_{j'}\} \cdot \left(\min_{j'\in V_k(j)} \{r_{j'}\} + \sum_{j'\in V_k(j)} y_{ij'} \cdot p_{ij'} - d_j \right) \leq z \quad \forall i \in P;\ j \in J;\ k \in \{1, \ldots, |V(j)|\}, \tag{13}$$

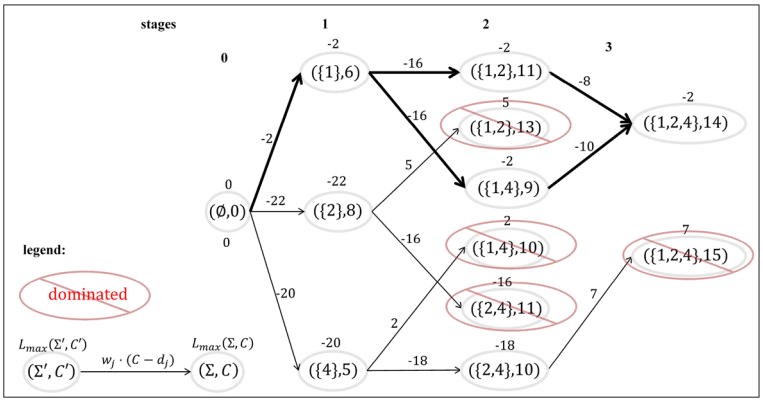


Fig. 4 Dynamic programming graph for the slave problem in the example minimizing $L_{\max}$

which determines the lower bound on the weighted maximum lateness. The idea behind inequalities (13) is to relax the weights and the release dates, so that the relaxed slave problem per processor becomes $[1||L_{\max}]$ which can be easily solved to optimality by the EDD rule (see Sect. 4.1). Thus, the optimal objective value of the relaxed slave problem $[1||L_{\max}]$, weighted with the smallest weight $\min_{j' \in V_k(j)}\{w_{j'}\}$, is a valid lower bound for the objective function of $[1|r_j|wL_{\max}]$ which is what inequalities (13) mimic.

To solve the corresponding slave problem $[1|r_j|wL_{\max}]$, BDP has to be adapted similarly as for $wF_{\max}$ objective with the following differences:

- The (partial) objective value of a state $(\Sigma, \mathcal{C}(\Sigma))$ is calculated as

$$\mathcal{L}_{\max}(\Sigma, \mathcal{C}(\Sigma)) = \min_{(\Sigma', \mathcal{C}(\Sigma)') \in \Xi(\Sigma, \mathcal{C}(\Sigma))} \{\max\{\mathcal{L}_{\max}(\Sigma', \mathcal{C}(\Sigma)'); w_j \cdot (\mathcal{C}(\Sigma) - d_j)\}\},$$

  where $j = \Sigma \backslash \Sigma'$.
- The lower bound of state $(\Sigma, \mathcal{C}(\Sigma))$ is computed as follows:

$$LB(\Sigma, \mathcal{C}(\Sigma)) = \max\left\{\mathcal{L}_{\max}(\Sigma, \mathcal{C}(\Sigma)); \max_{j \in \Gamma_i \backslash \Sigma; k \in \{1,\ldots,|V(j)|\}} \left\{\min_{j' \in V_k(j) \cap (\Gamma_i \backslash \Sigma)} \{w_{j'}\} \cdot \left(\max\left\{\mathcal{C}(\Sigma); \min_{j' \in V_k(j) \cap (\Gamma_i \backslash \Sigma)} \{r_{j'}\}\right\} + \sum_{j' \in V_k(j) \cap (\Gamma_i \backslash \Sigma)} p_{ij'} - d_j\right)\right\}\right\},$$

- While generating the states for BDP, criteria 1 and 2 from Sect. 3.1.2 have to be ignored because the deadlines are no longer strict.

Example (cont.): The dynamic programming graph for the example slave problem from Sect. 3.1.2, minimizing the maximum weighted lateness is depicted in Fig. 4.

Note that if the problem does not consider any weights, then the corresponding slave problem $[1|r_j|L_{\max}]$ is a textbook single-machine scheduling problem and can

be solved by an exact algorithm from the literature; for an overview, we refer to Leung (2004, Chap. 10) and Pinedo (2016, Chap. 3.2).

To adjust the GSPP model for the new objective, only the objective function (7) has to be substituted by new objective function

$$\text{Minimize } G(V) = \max_{(i,j,t)=m\in M} \{w_j \cdot v_m \cdot (t + p_{ij} - d_j)\}.$$

However, note that GSPP formulation with lateness objective can only be applied if the planning horizon is bounded from above by another restriction, e.g., end availability time of the machines or an upper bound on the schedule length.

5 Computational study

In this section, we compare the computational performance of the three presented solution approaches, namely a default solver solving the original mixed integer program (MIP-OP), branch and Benders cut (B&BC), and heuristic column selection based on a generalized set partitioning formulation (GSPP). A mixed-integer linear programming model [MIP-OP] for $[R|r_j, \overline{d}_j|C_{\max}]$, based on the model formulation for the classic discrete berth allocation problem proposed by Imai et al. (2001) and Monaco and Sammarra (2007), is presented in "Appendix." We use it as a benchmark. Experiments are performed on newly generated random problem instances as well as the benchmark instances from the literature for both applications (i.e., berth allocation and truck scheduling). We first describe how new problem instances have been generated and then describe the instances from the literature. Afterwards, we present the computational results of our solution approaches on both instance sets.

5.1 Benchmark instances and computational environment

To observe the computational performance of our algorithms, we test them on different kinds of problem instances. First, we generate new random instances for the basic problem $[R|r_j, \overline{d}_j|C_{\max}]$ according to the instance generation scheme proposed by Hall and Posner (2001). Specifically, the processing time p_{ij} of each job $j \in J$ on each machine $i \in P$ is randomly drawn from the following normal distribution: $p_{ij} \sim N(30, 6)$ (truncated below to positive lower bound). The release dates r_j are generated with the following scheme: $r_1 = 0$ and $r_j = r_{j-1} + X_j$, $\forall j \in 2, \ldots, n$, where $X_j \sim exp(\lambda)$ is an exponentially distributed random number with mean $\lambda = \frac{30}{|P|}$. This way, the jobs' arrival represents a Poisson process whose rate matches the total processors' job capacity. Finally, we draw the deadline of job j with the following equation: $\overline{d}_j = r_j + k \cdot E[p_{ij}]$, where $E[p_{ij}] = \sum_{i\in P} \frac{p_{ij}}{|P|}$ and $k \geq 1$. Thus, by varying parameter k, we receive instances that either have tight time windows (low k) or have more flexibility (high k). All time related parameters are rounded to the next integer value. Note that this instance generation scheme does not prevent infeasible instances from being generated. However, instances with a higher value of k are more likely to have feasible solutions.

Table 2 Summary of benchmark instance sets

Proposed by	Dataset name	# Instances	n	m	Additional characteristics
This paper (based on Hall and Posner (2001))	S	90	{20}	{4, 5, 6}	$k \in \{2, 3, 4\}$
	M	10	{50}	{10}	$k = 2.5$
	L	90	{80}	{15, 20, 25}	$k \in \{2, 3, 4\}$
Cordeau et al. (2005)	I2	50	{25, 35}	{5, 7, 10}	–
	I3	30	{60}	{13}	–
Tadumadze et al. (2019)	ITWS-DC-M	30	{100}	{25}	$\Omega_{\max} \in \{1, 2, 3\}$

To observe the impact of the problem size on the computational performance of our algorithms, we generate different sized sets of problem instances. Specifically, we generate small instances consisting of $n = 20$ jobs and $m \in \{4, 6, 8\}$ machines (dubbed S) and larger instances with $n = 80$ jobs and $m \in \{15, 20, 25\}$ machines (dubbed L). Moreover, we vary parameter $k \in \{2, 3, 4\}$. Apart from this, for the parameter tuning tests we use medium-sized instances with $n = 50$ jobs, $m = 10$ machines and $k = 2.5$ (dubbed M).

Apart from our new problem instances, we test the performance of our approaches on extant problem instances from the literature for both aforementioned applications: berth allocation and truck scheduling.

In the context of berth allocation, we use the problem instances from sets "*I*2" and "*I*3" originally proposed by Cordeau et al. (2005). These instances were generated based on statistical traffic and berth allocation data at the Port of Gioia Tauro. *I*2 contains 50 instances with the following five instance sizes: 25 ships (i.e., jobs) with 5, 7 and 10 berths (i.e., processors); 35 ships with 7 and 10 berths. *I*3 consists of 30 larger instances with 60 ships and 13 berths.

As benchmark instances from the truck scheduling context, we use the problem instances generated by Tadumadze et al. (2019) that can be downloaded using the following DOI: https://doi.org/10.5281/zenodo.1487845. Specifically, we use the problem instances from the set "ITWS-DC-M" that contains 30 larger problem instances with 100 trucks (i.e., jobs) and 25 dock doors (i.e., processors). Furthermore, these instances vary in the relative width of time windows of the trucks, which is controlled by the parameter $\Omega_{\max} \in \{1, 2, 3\}$, where instances with lower $\Omega_{\max}$ tend to have tighter time windows (similar to parameter k described above). Table 2 summarizes all the instances used.

Note that there is, to the best of our knowledge, no dataset in the literature for $[R|r_j, \overline{d}_j|C_{\max}]$ and its variants discussed in this paper. The literature datasets of Cordeau et al. (2005) and Tadumadze et al. (2019) have a feasible solution space that is covered by our approaches, but the original papers consider a different objective function—minimization of total (weighted) flow time $\sum w_j F_j$, which differs from all minmax objectives considered in this work. The numerical results can therefore not

be directly compared. However, the performance of our algorithms on these datasets should nonetheless give a good idea of the applicability of our approaches.

We have implemented our solution procedures in C# 6.0 and applied off-the-shelf solver IBM ILOG CPLEX Optimizer V12.8.0 for solving our MILP models, including the master model of our B&BC approach. All tests have been executed on an x64 PC with an Intel Core i7-8700K 3.70 GHz CPU and 64 GB RAM. The time limit for solving the MILP models is set to 300 CPU seconds. Note that we also experimented with larger time limits in preliminary tests but found that the best upper bounds are usually found within the first five minutes of computation. Moreover, to make a fair comparison between alternative solution approaches, we execute all solution methods (including the default solver) on a single thread. The problem instances as well as the detailed computational results for every instance are available from the following DOI: https://doi.org/10.5281/zenodo.3696775.

5.2 Parameter tuning

We calibrate both of the proposed approaches in a pretest. Specifically, in the case of B&BC we compare two ways of adding cuts to the master model: either as lazy constraints, injecting them into the branch & bound tree as it grows, or as regular constraints, such that the master model is resolved from scratch every time a new cut is added. Similarly, the performance of our heuristic column selection approach strongly depends on the value of the parameter μ. A low value of μ leads to a high number of generated columns which, on the one hand, can be beneficial for smaller size problem instances. On the other hand, it can be counterproductive for larger instances, where solving the resulting large GSPP model within a given time limit may lead to poor solutions. The proper value of μ clearly depends on the instance size; thus, we derive the value of μ from the number of jobs $|J|$ with the equation $\mu = \lceil |J| \cdot \delta \rceil$ and the predefined parameter δ. Note that a high value of δ leads to compact GSPP models with a smaller search space.

For our pretest, we use 10 medium-sized instances with $|J| = 50$, $|P| = 10$, and $k = 2.5$. We solve them with regard to makespan as an objective function applying both versions of B&BC (i.e., applying either lazy or regular cuts) and the heuristic GSPP approach, applying five different values of $\delta \in \{0.02, 0.1, 0.2, 0.3, 0.4\}$.

Table 3 shows the results for the parameter tuning test with regard to $C_{\max}$ (i.e., makespan) as an objective function. Column "$C^*_{\max}$" denotes the best known makespan for that instance, found by B&BC, either applying lazy or regular cuts, where an asterisk (*) after the value indicates that this objective value is optimal. For each approach (i.e., B&BC with both kinds of cuts and GSPP heuristics for each value of δ), we report the average relative gap (in %) of the objective value val$'$ found by that approach to the best known objective value for that instance val*, calculated as $\text{gap} = \frac{\text{val}' - \text{val}^*}{\text{val}^*} \cdot 100$ and the required computational time (in CPU seconds).

First of all, the results tend to be better when adding cuts to [Master] as lazy constraints. For each of the 10 instances, B&BC with lazy cuts obtains an optimal solution within the given time limit. On the other hand, B&BC with regular cuts struggles to solve 2 out of 10 instances to proven optimality within 300 seconds.

Table 3 Calibration of B&BC (type of cuts) and GSPP heuristic (value of δ)

Instance	C^*_{max}	B&BC		GSPP				
		Lazy cuts Gap/time	Regular cuts Gap/time	$\delta = 0.02$ Gap/time	$\delta = 0.1$ Gap/time	$\delta = 0.2$ Gap/time	$\delta = 0.3$ Gap/time	$\delta = 0.4$ Gap/time
50 × 10−01	175*	0.0/0.6	0.0/0.1	9.1/300.8	0.0/11.4	0.0/1.9	0.0/0.2	0.0/0.2
50 × 10−02	230*	0.0/0.0	0.0/0.1	2.6/300.8	0.0/6.2	0.0/3.9	0.0/0.2	0.0/0.1
50 × 10−03	153*	0.0/0.5	0.0/22.7	23.5/300.7	1.3/150.5	3.3/6.7	4.6/4.4	7.8/2.7
50 × 10−04	201*	0.0/0.4	0.0/0.1	0.0/250.7	0.0/16.4	0.0/4.1	0.0/0.1	0.0/0.1
50 × 10−05	189*	0.0/1.3	0.0/300.0	–/300.7	1.1/52.2	3.2/8.1	2.7/2.4	3.2/2.0
50 × 10−06	153*	0.0/0.1	0.0/0.1	–/300.7	0.0/8.0	0.0/0.4	0.7/2.3	3.3/1.5
50 × 10−07	163*	0.0/1.0	0.0/141.4	–/300.7	1.2/15.4	1.8/6.8	1.8/6.3	3.1/2.4
50 × 10−08	170*	0.0/114.0	1.8/300.0	–/300.7	1.2/46.4	3.5/5.1	4.7/5.0	3.5/2.1
50 × 10−09	200*	0.0/0.0	0.0/0.1	10.0/300.8	0.0/19.0	0.0/0.4	0.0/0.2	0.0/0.2
50 × 10−10	176*	0.0/0.2	0.0/1.8	26.7/300.7	0.0/128.7	0.0/0.3	2.8/1.8	2.8/0.8
Mean		0.0/11.8	0.2/76.6	12.0/295.7	0.5/45.4	1.2/3.8	1.7/2.3	2.4/1.2

* Indicates that the value is optimal
– Indicates that in the time limit no feasible solution has been found

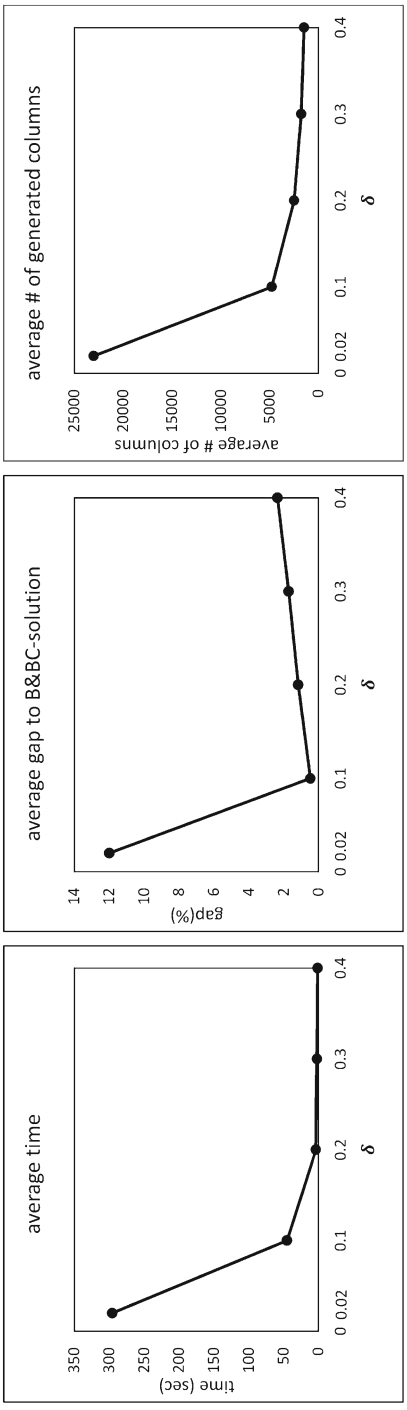


Fig. 5 Influence of δ on the solution quality of the GSPP heuristic

This is likely due to the computation time that is expended on solving the master model from scratch in every iteration. This leads to the solver wasting a lot of time re-exploring regions of the search space it already explored in previous iterations. Note, however, that regular cuts are sometimes marginally superior to lazy constraints, probably because the presolver is more aggressive in the absence of lazy constraints and the root node relaxation is stronger in later iterations, when multiple cuts have already been added. Nonetheless, in light of the performance stability of lazy cuts, adding constraints lazily seems advisable.

Figure 5 summarizes the average results in terms of runtime and relative gap to the optimal objective value (obtained by B&BC) for each value of parameter δ used to select columns for the GSPP heuristic. The leftmost and rightmost graphics of Fig. 5 present an apparently exponential decrease in runtime and number of generated columns when the value of δ rises. For $\delta = 0.02$, only one out of 10 instances could be solved in less than 300 seconds (note that $\delta = 0.02$ leads to $\mu = 1$, and the resulting GSPP, containing all columns, is equivalent to the original problem). For higher values of δ, all resulting GSPP model instances could be solved within the time limit. However, raising the value of δ makes it less probable that the GSPP contains potentially good columns, possibly lowering the solution quality. From our results, $\delta = 0.1$ turns out to be the most promising compromise of solution quality and runtime. For $\delta = 0.1$, in 6 out of 10 cases, GSPP obtains the optimal solution for the corresponding original problem while maintaining acceptable computational times. Also, for the remaining 4 instances, GSPP finds near-optimal solutions (with a maximal optimality gap of less than 1.4%) in reasonable computational times. Therefore, for the next computational experiments we apply $\delta = 0.1$ when using our GSPP heuristic and we add cuts as lazy constraints to the master model of B&BC. Note that setting $\delta = 0.1$ does not guarantee that a feasible solution is in the GSPP search space. If no feasible solution can be found, δ must be lowered to expand the number of columns in the model. However, in our computational experiments, this proves to be unnecessary; we are able to solve all instances to feasibility with $\delta = 0.1$.

5.3 Computational performance on new instances

In this section, we compare the computational performance of the three proposed approaches (B&BC, GSPP and MIP-OP) solving $[R|r_j, \overline{d}_j|C_{\max}]$. For this, we use the problem instances from two different sized datasets and solve them with all three solution approaches. The first dataset S contains the problem instances with $n = 20$ jobs and $m \in \{4, 6, 8\}$ processors while the second dataset L consists of larger instances considering $n = 80$ jobs and $m = \{15, 20, 25\}$ machines. Further, we observe the impact of the time window tightness on the performance of our approaches by varying parameter $k \in \{2, 3, 4\}$. For each parameter constellation, we have 10 random instances, so that in total 180 new problem instances have been generated.

Table 4 summarizes the aggregated computational results for the newly generated problem instances. The first two columns describe the instance characteristics: the number of jobs n and machines m and the value of parameter k. For each approach, in columns "Gap (%)" and "Time(s)" we report the average relative gap (in %) of

Table 4 Comparison between B&BC, GSPP and MIP-OP on newly generated instances

Instance		B&BC				GSPP					MIP-OP		
Size	k	Opt	Gap (%)	Time (s)	Cuts	Best	Opt	Gap (%)	Time (s)	Columns	Opt	Gap (%)	Time (s)
Dataset$S$													
20 × 4	2	1	0	0.2	314.9	1.0	1.0	0.0	0.3	1265.3	0.3	0.8	267.9
20 × 4	3	1	0	0.1	33.6	0.9	1.0	0.0	5.4	2465.2	0.3	0.2	226.2
20 × 4	4	1	0	9.3	851.6	0.6	1.0	0.3	13.7	3689.6	0.3	0.6	225.3
20 × 6	2	0.9	0	30.3	256.4	0.8	1.0	0.2	0.6	1882.7	0.2	0.9	276.3
20 × 6	3	0.9	0	30.5	246.6	0.8	1.0	0.1	9.9	3691.8	0.1	0.1	277.4
20 × 6	4	0.9	0	30.1	918.7	0.7	1.0	0.2	7.8	5489.2	0.4	0.3	215.9
20 × 8	2	1	0	0.7	13.6	0.8	1.0	0.2	2.9	2524.8	0.4	0.4	232.6
20 × 8	3	1	0	2.1	14.6	0.9	1.0	0.2	10.3	4925.7	0.4	0.0	203.6
20 × 8	4	1	0	3.6	53.1	1.0	1.0	0.0	9.7	7319.2	0.4	0.2	209.5
Mean (S)		0.97	0.00	11.88	300.34	0.83	1.00	0.15	6.73	3694.83	0.31	0.39	237.18
Dataset$L$													
80 × 15	2	0.5	0.6	184.9	6620.4	0.8	1.0	0.2	61.6	5122.2	–	–	–
80 × 15	3	0.8	0	76.7	31.4	0.2	0.3	8.5	269.5	9642.5	–	–	–
80 × 15	4	0.6	0	121.0	38.8	0.0	0.0	21.0	300.6	14188.2	–	–	–
80 × 20	2	0.6	0	122.4	316.3	0.8	1.0	0.2	111.1	6820.6	–	–	–
80 × 20	3	0.8	0	89.0	74.0	0.3	0.4	8.8	262.0	12877.1	–	–	–
80 × 20	4	0.8	0	71.3	33.8	0.2	0.2	22.3	270.7	18926.9	–	–	–
80 × 25	2	0.7	0.7	121.1	303.2	0.5	0.5	7.4	219.6	8524.9	–	–	–
80 × 25	3	0.6	0.2	121.6	39.1	0.6	0.6	5.3	227.8	16082.4	–	–	–
80 × 25	4	0.6	0	123.2	43.0	0.0	0.0	36.0	300.8	23684.4	–	–	–
Mean (L)		0.67	0.17	114.58	833.33	0.38	0.44	12.18	224.86	12874.36	–	–	–

Each entry contains averaged results over 10 instances with the same parameter constellation

the best found objective value to the best known objective value of that instance, calculated as described in Sect. 5.2, and the average computational runtime (in CPU seconds), averaged per 10 instances of the same size. Furthermore, for each approach we report the share of the instances whose corresponding (M)ILP models are solved to optimality within the given time limit. Note, that in case of our GSPP heuristic solving the corresponding ILP does not necessarily mean that the found solution is optimal. Therefore, for GSPP, we additionally report the share of instances whose objective value matches the best known objective value (column "Best"). Moreover, the average number of generated cuts for B&BC and columns for GSPP are reported in columns "Cuts" and "Columns," respectively. For the large instances from dataset L the solver runs out of memory even before generating the corresponding MIP-OP model. Hence, for those instances we only report the computational performance of the B&BC and GSPP approaches.

The first remarkable result is that B&BC outperforms its competitors, solving in total 147 out of 180 instances to optimality within 300 seconds. Only for 5 large instances, the solution obtained by B&BC is improved upon by the GSPP heuristic, which found the best known solution for 108 out of 180 instances. It is noteworthy that the heuristic column selection approach tends to be more successful for problem instances with tight time windows (i.e., low k) and fewer processors, leading to fewer columns (i.e., potential starting times of jobs). The default solver, on the other hand, struggles with solving instances to proven optimality within the time limit using model [MIP-OP]. Only 28 out of 90 small instances from dataset S can be solved to optimality within a time limit and CPLEX runs out of memory even before generating the corresponding MIP-OP models for large instances from dataset L.

5.4 Benchmark tests on berth allocation problem instances from the literature

To observe the performance of our approaches on berth allocation problem instances, we use datasets $I2$ and $I3$ originally proposed by Cordeau et al. (2005), containing in total 80 realistic BAP instances.

Note that Cordeau et al. (2005), apart from the time window restrictions for each vessel, additionally consider the availability times of berths (i.e., start s_i and end e_i of availability for each berth $i \in P$). Thus, to solve these instances with our approaches we modify them to include the machine availability time restrictions as described in Sect. 4.1. Moreover, since minimizing the weighted flow time is a common objective in the berth allocation literature, apart from our baseline objective (i.e., makespan) we solve the instances with regard to maximum weighted flow time. Finally, we solve the instances with regard to maximum weighted lateness, which is considered an appropriate objective in many berth allocation scenarios (e.g., Liu et al. 2006; Chen et al. 2012).

Tables 5, 6 and 7 summarize the computational results of our comparison on the BAP instances with regard to objective $C_{\max}$, $wF_{\max}$ and $wL_{\max}$, respectively. Each entry contains averaged results over all instances of the same size (i.e., for dataset $I2$, each entry contains results averaged over 10 instances and for dataset $I3$, over 30 instances). The columns of Tables 5, 6 and 7 have the same meaning as in Table 4

Table 5 Comparison between B&BC, GSPP and MIP-OP on BAP minimizing makespan

Instance size	$C^*_{\max}$	B&BC				GSPP					MIP-OP		
		Opt	Gap (%)	Time (s)	Cuts	Best	Opt	Gap (%)	Time (s)	Columns	Opt	Gap (%)	Time (s)
25 × 5	173.5	0.8	0.0	61.2	60.3	0.8	1.0	0.2	32.5	8,118.5	0.2	1.0	250.9
25 × 7	163.7	1.0	0.0	0.0	22.2	1.0	1.0	0.0	20.5	11,323.7	0.6	0.1	188.2
25 × 10	171.5	1.0	0.0	0.1	23.6	1.0	1.0	0.0	23.5	14,973.5	0.3	0.1	255.8
35 × 7	175.6	0.7	0.0	95.4	57.4	0.4	0.8	0.7	152.6	11,858.2	0.0	5.2	300.7
35 × 10	173.7	1.0	0.0	0.2	47.8	0.8	0.9	4.2	95.6	15,908.2	0.0	2.6	301.0
60 × 13	169.5	1.0	0.0	0.1	33.6	1.0	1.0	0.0	47.8	14,940.9	0.0	9.4	310.9

Each entry contains results averaged over all instances of the same size

from Sect. 5.3. Furthermore, we report the objective values of the best found solutions, averaged per parameters constellation (see columns "$C^*_{\max}$," "$wF^*_{\max}$" and "$wL^*_{\max}$" of Tables 5, 6 and 7, respectively).

The results indicate that B&BC remains a successful solution approach for BAP with regard to all considered performance criteria. It solves in total 77, 78 and 78 out of 80 instances to proven optimality within 300 seconds for objectives $C_{\max}$, $wF_{\max}$ and $wL_{\max}$, respectively. Also, the GSPP heuristic performs quite well, providing optimal solutions for 70, 64 and 66 out of 80 instances for $C_{\max}$, $wF_{\max}$ and $wL_{\max}$, while the gaps for non-optimal solutions are almost always negligible. The default solver, on the other hand, struggles with solving instances to proven optimality within the time limit using the MIP-OP formulation. CPLEX manages to solve only 11 out of 80 instances within the time limit while minimizing $C_{\max}$. The effort of MIP-OP rises for the alternative objectives (i.e., minimization $wF_{\max}$ and $wL_{\max}$). CPLEX only solves one out of 80 instances within the time limit when $wL_{\max}$ is minimized, and for $wL_{\max}$, none of 80 instances are solved within 300 CPU seconds.

5.5 Benchmark tests on truck scheduling problem instances from the literature

Our benchmark set of problem instances from the truck scheduling context is originally proposed by Tadumadze et al. (2019) for what they call the "integrated truck and workforce scheduling problem" (ITWS). The set contains 30 very large problem instances considering 100 trucks and 25 doors. These instances are characterized by tighter time windows than our newly generated instances, which are controlled by the parameter $\Omega_{\max}$. More specifically, the expected relative width of the trucks' time windows for an ITWS instance with $\Omega_{\max} = 1$ is comparable with the expected relative width of the time windows of an problem instance generated by our instance generator using $k = 0.5$. However, the instances of Tadumadze et al. (2019) are generated in such way that each problem instance has at least one feasible solution with regard to the trucks' time windows. Note that the formal definition of the ITWS for the distribution center context (ITWS-DC) from Tadumadze et al. (2019) differs from our problem as apart from truck scheduling it simultaneously solves the workforce scheduling problem. The problem of scheduling jobs with time windows on unrelated machines is equivalent to the case when ITWS-DC is solved for a given dock-specific workforce assignment as described in the original paper.

Tables 8 and 9 present the computational results of our approaches on truck scheduling instances. These instances are too large to be solved by a default solver. MIP-OP runs out of the memory even before generating the corresponding MILP model. Thus, we only solve each instance with B&BC and GSPP heuristics. Moreover, not every instance could be solved to feasibility, which is why we additionally report the share of instances that could be solved to feasibility within the time limit (see column "Feas").

As can be seen from the results, B&BC tends to be less successful at solving the large truck scheduling instances with tighter time windows from Tadumadze et al. (2019). The effort of B&BC is especially high for problem instances with tight time windows (i.e., low $\Omega_{\max}$). Recall that [Master] is a relaxed version of the problem which ignores the time window-related constraints. If the time windows of the jobs

Table 6 Comparison between B&BC, GSPP and MIP-OP on BAP minimizing maximum weighted flow time

Instance size	$wF^*_{\max}$	B&BC				GSPP					MIP-OP		
		Opt	Gap (%)	Time (s)	Cuts	Best	Opt	Gap (%)	Time (s)	Columns	Opt	Gap (%)	Time (s)
25 × 5	58.2	1.0	0.0	5.6	47.3	0.2	1.0	2.5	23.1	8,118.5	0.0	4.5	300.2
25 × 7	48.9	1.0	0.0	0.3	24.1	1.0	1.0	0.0	24.0	11,323.7	0.0	4.3	300.2
25 × 10	51.3	1.0	0.0	0.3	20.3	1.0	1.0	0.0	16.3	14,973.5	0.0	1.0	300.3
35 × 7	57.1	0.9	0.0	55.5	50.5	0.4	1.0	1.4	61.7	11,858.2	0.0	14.8	300.7
35 × 10	61.1	0.9	0.0	31.5	53.7	0.8	1.0	0.4	72.2	15,908.2	0.0	19.6	301.0
60 × 13	43.1	1.0	0.0	0.5	11.6	1.0	1.0	0.0	40.7	14,940.9	0.0	219.5	310.7

Each entry contains results averaged over all instances of the same size

Table 7 Comparison between B&BC, GSPP and MIP-OP on BAP minimizing maximum weighted lateness

Instance size	$wL^*_{\max}$	B&BC				GSPP					MIP-OP		
		Opt	Gap (%)	Time (s)	Cuts	Best	Opt	Gap (%)	Time (s)	Columns	Opt	Gap (%)	Time (s)
25 × 5	− 241.8	1.0	0.0	6.0	44.1	0.2	1.0	− 0.6	27.7	8,118.5	0.0	− 1.0	300.0
25 × 7	− 251.1	1.0	0.0	0.9	26.9	1.0	1.0	0.0	14.7	11,323.7	0.0	− 0.1	300.0
25 × 10	− 248.7	1.0	0.0	1.0	81.1	1.0	1.0	0.0	15.8	14,973.5	0.1	− 0.2	271.4
35 × 7	− 242.9	0.9	0.0	60.9	46.3	0.4	1.0	− 0.3	75.1	11,858.2	0.0	− 2.1	300.1
35 × 10	− 238.9	0.9	0.0	32.6	52.1	1.0	1.0	0.0	37.3	15,908.2	0.0	− 2.3	300.1
60 × 13	− 256.9	1.0	0.0	0.9	11.0	1.0	1.0	0.0	23.2	14,940.9	0.0	− 9.2	300.6

Each entry contains results averaged over all instances of the same size

Table 8 Comparison between B&BC and GSPP on ITWS minimizing makespan

Instance			B&BC					GSPP					
Size	Ω_{max}	C^*_{max}	Feas	Opt	Gap (%)	Time (s)	Cuts	Feas	Best	Opt	Gap (%)	Time (s)	Columns
100 × 25	1	72.7	0.7	0.7	0.0	114.8	15897.0	1.0	1.0	1.0	0.0	7.8	5677.3
100 × 25	2	69.2	0.8	0.3	0.6	217.5	7806.5	1.0	0.6	0.8	2.5	131.5	8889.0
100 × 25	3	69.3	0.8	0.3	1.0	215.8	21499.4	1.0	0.8	0.8	4.3	113.9	10350.7

Each entry contains averaged results over all 10 instances with the same parameter constellation

Table 9 Comparison between B&BC and GSPP on ITWS minimizing maximum flow time

Instance			B&BC					GSPP					
Size	$\Omega_{\max}$	$F^*_{\max}$	Feas	Opt	Gap (%)	Time (s)	Cuts	Feas	Best	Opt	Gap (%)	Time (s)	Columns
100×25	1	31.0	0.2	0.0	27.8	301.0	292.2	1.0	1.0	1.0	0.0	24.7	5677.3
100×25	2	37.4	0.8	0.0	20.3	300.9	131.8	1.0	0.7	0.6	8.7	167.8	8889.0
100×25	3	41.6	0.6	0.0	14.7	300.9	139.4	1.0	0.9	0.5	2.3	222.4	10350.7

Each entry contains averaged results over 10 instances with the same parameter constellation

are too tight, the relaxation is bound to be less tight. As a result, the number of cuts added to [Master] rises, which negatively affects the performance of B&BC. While minimizing the makespan, B&BC obtains a feasible solution for 23 out of 30 instances out of which 13 are optimal. The effort is greater when the maximum weighted flow time is minimized. Here, B&BC manages to find a feasible solution for 16 out of 30 instances, none of which is proven to be optimal. On the other hand, GSPP exhibits exactly the opposite behavior. This is due to there being fewer columns (i.e., fewer potential starting times of jobs) in problems with tighter time windows. GSPP manages to find a feasible solution for every instance, which in most cases matches or improves the solutions found by B&BC. Specifically, while minimizing the makespan, GSPP manages to match or improve solutions obtained by B&BC for 24 out of 30 instances in shorter computational times. For the alternative objective (i.e., $F_{\max}$), GSPP shows even better performance matching or improving the B&BC solutions for 26 out of 30 instances. This indicates that B&BC and GSPP complement each other.

6 Conclusion

In this paper, we propose a novel logic-based branch & Benders cut algorithm for a family of unrelated parallel machines scheduling problems with time windows and minmax objective. We also devise a heuristic column selection approach based on a generalized set partitioning problem formulation. We show that the proposed approaches can be easily modified for relevant generalizations. We compare our methods to an adaptation of an existing mixed integer linear program solved by a commercial solver. We test all algorithms on newly generated problem instances with varying time window width and realistic benchmark instances from the literature.

The exact B&BC algorithm performs quite well, solving most instances, both from the literature and generated, to optimality within a few minutes, clearly outperforming a default solver solving the non-decomposed MILP model. The only cases where B&BC exhibits non-negligible optimality gaps are very large problem instances with tight time windows. For those cases, our heuristic column selection method is faster and frequently delivers better results.

Future research could aim to adjust the proposed solution method for alternative performance criteria, like total weighted flow time and other minsum objectives. Moreover, alternative heuristic approaches can also be tested and compared to our GSPP approach.

Appendix

This appendix contains a mixed-integer linear programming model for $[R|r_j, \overline{d}_j|C_{\max}]$. The model is based on the original discrete berth allocation model

proposed by Imai et al. (2001) and Monaco and Sammarra (2007), which is modified to account for minmax objectives. Table 10 summarizes the notation.

$$\text{[MIP-OP] Minimize } C(X,U) = \max_{i \in P} \left\{ \sum_{k \in K} \left(u_{ik} + \sum_{j \in J} p_{ij} \cdot x_{ijk} \right) \right\} \tag{14}$$

subject to

$$\sum_{i \in P} \sum_{k \in K} x_{ijk} = 1 \qquad \forall j \in J \tag{15}$$

$$\sum_{j \in J} x_{ijk} \leq 1 \qquad \forall i \in P, k \in K \tag{16}$$

$$\sum_{j \in J} r_j \cdot x_{ijk} - \sum_{\substack{l \in K: \\ l < k}} \left(u_{il} + \sum_{j \in J} p_{ij} \cdot x_{ijl} \right) \leq u_{ik} \qquad \forall i \in P, k \in K \tag{17}$$

$$\sum_{\substack{l \in K: \\ l \leq k}} \left(u_{il} + \sum_{j' \in J} p_{ij'} \cdot x_{ij'l} \right) \leq \overline{d}_j + \mathcal{M} \cdot (1 - x_{ijk}) \quad \forall i \in P, j \in J, k \in K \tag{18}$$

$$u_{ik} \geq 0 \qquad \forall i \in P, k \in K \tag{19}$$

$$x_{ijk} \in \{0; 1\} \qquad \forall j \in J, i \in P, k \in K \tag{20}$$

Objective function (14) minimizes the makespan of the machine whose completion time after processing all jobs is highest. The makespan for each machine is derived

Table 10 Notation for the MILP model

J	Set of jobs (index j)
P	Set of processors (index i)
K	Set of service positions (index k, $K = \{1, \ldots, n\}$)
$\mathcal{M}$	Big integer
p_{ij}	Processing time of job j on processor i
r_j	Release date of job j
$\overline{d}_j / d_j$	Deadline/due date of job j
x_{ijk}	Binary variable: 1, if job j is the kth job to be processed on processor i; 0, otherwise
X	Set of x variables: $X = \{x_{ijk} \mid i \in P, j \in J, k \in K\}$
u_{ik}	Continuous variable: time interval that processor i stays idle before executing the kth job and after executing the $(k-1)$th job
U	Set of u variables: $U = \{u_{ik} \mid i \in P, k \in K\}$

by adding up the total processing time of the jobs and the total idle time between jobs. Constraints (15) ensure that every job is assigned to exactly one processor at exactly one service position. Constraints (16) make it impossible for multiple jobs to be assigned the same service position on the same machine. Inequalities (17) define the idle times between jobs: idle time u_{ik} (i.e., the amount of time that machine i stays idle before executing the kth job) must be no less than the duration between the completion time of the $(k-1)$th job on machine i and the release date of the kth job on the same machine (i.e., the earliest possible processing start time of its successive job). Note that the summation on the left-hand side of constraints (17) and (18) indicates the completion time of the $(k-1)$th job on machine i. Constraints (18) make sure that the deadlines are not violated. Finally, (19) and (20) define the domain of the variables.

To extend MIP-OP to account for machine availability restrictions, constraints (21) and (22) are added to the model.

$$u_{i1} \geq s_i \qquad \forall i \in P \tag{21}$$

$$\sum_{k \in K} \left(u_{ik} + \sum_{j \in J} p_{ij} \cdot x_{ijk} \right) \leq e_i \qquad \forall i \in P \tag{22}$$

Constraints (21) guarantee that on each machine $i \in P$, processing of its first job cannot be started before its availability time s_i. Constraints (22) take care that on each machine i none of the jobs are completed after its availability time e_i.

Finally, to consider objectives maximum weighted flow time and lateness, we consider the following models.

To minimize the maximum weighted flow time,

$$\text{minimize } F(X, U, F^{\text{flow}}) = F^{\text{flow}},$$

subject to (15)–(20) and

$$F^{\text{flow}} \geq w_j \left(\sum_{\substack{k' \in K : \\ k' \leq k}} \left(u_{ik'} + \sum_{j' \in J} p_{ij'} \cdot x_{ij'k'} \right) - r_j \right) - \mathcal{M} \cdot (1 - x_{ijk}), \forall i \in P, j \in J, k \in K.$$

To minimize the maximum weighted lateness,

$$\text{minimize } L(X, U, F^{\text{late}}) = F^{\text{late}},$$

subject to (15)–(17), (19), (20), and

$$F^{\text{late}} \geq w_j \left(\sum_{\substack{k' \in K: \\ k' \leq k}} \left(u_{ik'} + \sum_{j' \in J} p_{ij'} \cdot x_{ij'k'} \right) - d_j \right)$$
$$- \mathcal{M} \cdot (1 - x_{ijk}), \forall i \in P, j \in J, k \in K$$

References

Benders JF (1962) Partitioning procedures for solving mixed-variables programming problems. Numer Math 4(1):238–252

Bierwirth C, Meisel F (2010) A survey of berth allocation and quay crane scheduling problems in container terminals. Eur J Oper Res 202(3):615–627

Bierwirth C, Meisel F (2015) A follow-up survey of berth allocation and quay crane scheduling problems in container terminals. Eur J Oper Res 244(3):675–689

Blazewicz J, Cheng TE, Machowiak M, Oguz C (2011) Berth and quay crane allocation: a moldable task scheduling model. J Oper Res Soc 62(7):1189–1197

Boysen N (2010) Truck scheduling at zero-inventory cross docking terminals. Comput Oper Res 37(1):32–41

Boysen N, Fedtke S, Weidinger F (2017) Truck scheduling in the postal service industry. Transp Sci 51(2):723–736

Boysen N, Fliedner M (2010) Cross dock scheduling: classification, literature review and research agenda. Omega 38(6):413–422

Buhrkal K, Zuglian S, Ropke S, Larsen J, Lusby R (2011) Models for the discrete berth allocation problem: a computational comparison. Transp Res Part E Logist Transp Rev 47(4):461–473

Carlier J (1982) The one-machine sequencing problem. Eur J Oper Res 11(1):42–47 Third EURO IV Special Issue

Chen JH, Lee D-H, Cao JX (2012) A combinatorial benders–cuts algorithm for the quayside operation problem at container terminals. Transp Res Part E Logist Transp Rev 48(1):266–275

Codato G, Fischetti M (2006) Combinatorial benders' cuts for mixed-integer linear programming. Oper Res 54(4):756–766

Cordeau J-F, Laporte G, Legato P, Moccia L (2005) Models and tabu search heuristics for the berth-allocation problem. Transp Sci 39(4):526–538

Ebenlendr T, Krčál M, Sgall J (2014) Graph balancing: a special case of scheduling unrelated parallel machines. Algorithmica 68(1):62–80

Emde S (2017) Optimally scheduling interfering and non-interfering cranes. Naval Res Logist (NRL) 64(6):476–489

Emde S, Boysen N, Briskorn D (2014) The berth allocation problem with mobile quay walls: problem definition, solution procedures, and extensions. J Sched 17(3):289–303

Fanjul-Peyro L, Ruiz R (2010) Iterated greedy local search methods for unrelated parallel machine scheduling. Eur J Oper Res 207(1):55–69

Fanjul-Peyro L, Ruiz R (2011) Size-reduction heuristics for the unrelated parallel machines scheduling problem. Comput Oper Res 38(1):301–309

Garey MR, Johnson DS (1979) Computers and intractability: a guide to NP-completeness. WH Freeman and Company, San Francisco

Gedik R, Rainwater C, Nachtmann HH, Pohl EA (2016) Analysis of a parallel machine scheduling problem with sequence dependent setup times and job availability intervals. Eur J Oper Res 251(2):640–650

Gharbi A, Haouari M (2002) Minimizing makespan on parallel machines subject to release dates and delivery times. J Sched 5(4):329–355

Ghirardi M, Potts C (2005) Makespan minimization for scheduling unrelated parallel machines: a recovering beam search approach. Eur J Oper Res 165(2):457–467

Graham RL, Lawler EL, Lenstra JK, Kan AR (1979) Optimization and approximation in deterministic sequencing and scheduling: a survey. Ann Discrete Math 5:287–326
Hall NG, Posner ME (2001) Generating experimental data for computational testing with machine scheduling applications. Oper Res 49(6):854–865
Held M, Karp RM (1962) A dynamic programming approach to sequencing problems. J Soc Ind Appl Math 10(1):196–210
Hooker J (2011) Logic-based methods for optimization: combining optimization and constraint satisfaction, vol 2. Wiley, Hoboken
Hooker JN (2007) Planning and scheduling by logic-based benders decomposition. Oper Res 55(3):588–602
Imai A, Nishimura E, Papadimitriou S (2001) The dynamic berth allocation problem for a container port. Transp Res Part B Methodol 35(4):401–417
Jain V, Grossmann IE (2001) Algorithms for hybrid milp/cp models for a class of optimization problems. INFORMS J Comput 13(4):258–276
Knop D, Koutecky M (2017) Scheduling meets n-fold integer programming. In: Proceedings of the 13th workshop on models and algorithms for planning and scheduling problems (MAPSP)
Konur D, Golias MM (2017) Loading time flexibility in cross-docking systems. Procedia Comput Sci 114:491–498
Lalla-Ruiz E, Expósito-Izquierdo C, Melián-Batista B, Moreno-Vega JM (2016) A set-partitioning-based model for the berth allocation problem under time-dependent limitations. Eur J Oper Res 250(3):1001–1012
Lancia G (2000) Scheduling jobs with release dates and tails on two unrelated parallel machines to minimize the makespan. Eur J Oper Res 120(2):277–288
Lawler EL (1973) Optimal sequencing of a single machine subject to precedence constraints. Manag Sci 19(5):544–546
Lee D-H, Qiu Wang H (2010) Integrated discrete berth allocation and quay crane scheduling in port container terminals. Eng Optim 42(8):747–761
Lenstra J, Rinnooy Kan A, Brucker P (1977) Complexity of machine scheduling problems. Ann Discrete Math 1:343–362
Lenstra JK, Shmoys DB, Tardos É (1990) Approximation algorithms for scheduling unrelated parallel machines. Math Program 46(1–3):259–271
Leung JY (2004) Handbook of scheduling: algorithms, models, and performance analysis. CRC Press, Boca Raton
Leung JY-T, Li C-L (2008) Scheduling with processing set restrictions: a survey. Int J Prod Econ 116(2):251–262
Li C-L, Cai X, Lee C-Y (1998) Scheduling with multiple-job-on-one-processor pattern. IIE Trans 30(5):433–445
Lin Y-K, Pfund ME, Fowler JW (2011) Heuristics for minimizing regular performance measures in unrelated parallel machine scheduling problems. Comput Oper Res 38(6):901–916
Liu J, Wan Y-W, Wang L (2006) Quay crane scheduling at container terminals to minimize the maximum relative tardiness of vessel departures. Naval Res Logist (NRL) 53(1):60–74
Mokotoff E, Chrétienne P (2002) A cutting plane algorithm for the unrelated parallel machine scheduling problem. Eur J Oper Res 141(3):515–525
Monaco MF, Sammarra M (2007) The berth allocation problem: a strong formulation solved by a lagrangean approach. Transp Sci 41(2):265–280
Pinedo M (2016) Scheduling: theory, algorithms, and systems, 5th edn. Springer, Berlin
Rahmaniani R, Crainic TG, Gendreau M, Rei W (2017) The benders decomposition algorithm: a literature review. Eur J Oper Res 259(3):801–817
Sanlaville E, Schmidt G (1998) Machine scheduling with availability constraints. Acta Inform 35(9):795–811
Sels V, Coelho J, Dias AM, Vanhoucke M (2015) Hybrid tabu search and a truncated branch-and-bound for the unrelated parallel machine scheduling problem. Comput Oper Res 53:107–117
Shchepin EV, Vakhania N (2005) An optimal rounding gives a better approximation for scheduling unrelated machines. Oper Res Lett 33(2):127–133
Steenken D, Voß S, Stahlbock R (2004) Container terminal operation and operations research-a classification and literature review. OR Spectr 26(1):3–49
Tadumadze G, Boysen N, Emde S, Weidinger F (2019) Integrated truck and workforce scheduling to accelerate the unloading of trucks. Eur J Oper Res 278(1):343–362

Tong CJ, Lau HC, Lim A (1999) Ant colony optimization for the ship berthing problem. In: Annual Asian computing science conference, Springer, pp 359–370

Tran TT, Araujo A, Beck JC (2016) Decomposition methods for the parallel machine scheduling problem with setups. INFORMS J Comput 28(1):83–95

UNCTAD (2016) Review of maritime transport 2016. Technical report, United Nations Conference on Trade and Development

Vacca I, Bierlaire M, Salani M (2007) Optimization at container terminals: status, trends and perspectives. In: 7th Swiss transport research conference

Xu D, Li C-L, Leung JY-T (2012) Berth allocation with time-dependent physical limitations on vessels. Eur J Oper Res 216(1):47–56