

Emde, Simon; Lysgaard, Jens

Article — Published Version

Branch-cut-and-price for open-shop scheduling problems with any regular minsum objective

Journal of Scheduling

Suggested Citation: Emde, Simon; Lysgaard, Jens (2026) : Branch-cut-and-price for open-shop scheduling problems with any regular minsum objective, Journal of Scheduling, ISSN 1099-1425, Springer US, New York, Vol. 29, Iss. 4, pp. 279-298, <https://doi.org/10.1007/s10951-026-00874-3>

This Version is available at:

<https://hdl.handle.net/10419/342619>

Standard-Nutzungsbedingungen:

Die Dokumente auf EconStor dürfen zu eigenen wissenschaftlichen Zwecken und zum Privatgebrauch gespeichert und kopiert werden.

Sie dürfen die Dokumente nicht für öffentliche oder kommerzielle Zwecke vervielfältigen, öffentlich ausstellen, öffentlich zugänglich machen, vertreiben oder anderweitig nutzen.

Sofern die Verfasser die Dokumente unter Open-Content-Lizenzen (insbesondere CC-Lizenzen) zur Verfügung gestellt haben sollten, gelten abweichend von diesen Nutzungsbedingungen die in der dort genannten Lizenz gewährten Nutzungsrechte.

Terms of use:

Documents in EconStor may be saved and copied for your personal and scholarly purposes.

You are not to copy documents for public or commercial purposes, to exhibit the documents publicly, to make them publicly available on the internet, or to distribute or otherwise use the documents in public.

If the documents have been made available under an Open Content Licence (especially Creative Commons Licences), you may exercise further usage rights as specified in the indicated licence.



<http://creativecommons.org/licenses/by/4.0/>



Branch-cut-and-price for open-shop scheduling problems with any regular minsum objective

Simon Emde¹ · Jens Lysgaard²

Received: 13 August 2021 / Accepted: 6 February 2026 / Published online: 22 May 2026
 © The Author(s) 2026

Abstract

This paper deals with a general open-shop scheduling problem where jobs have release and due dates, transferring between machines incurs a machine–pair-dependent delay, and any regular objective defined over job completion times is minimized. Such problems are widely encountered in practice, e.g., in industrial testing and maintenance processes and just-in-time logistics. Despite its relevance, exact solution methods from the literature have so far focused on the makespan objective or very specific special cases. We hence present the first exact method for the general open-shop scheduling problem with any regular minsum objective with respect to job completion times. Moreover, we demonstrate how branch-cut-and-price methods can be applied to the open shop. Finally, we also adapt the famous subset-row inequalities, originally proposed for the vehicle routing problem, to the open shop. Our computational study on both newly generated instances as well as those from the literature demonstrates good performance on a broad set of different objective functions, finding tight bounds even for the largest problems.

Keywords Open-shop scheduling · Branch-cut-and-price · Subset-row inequalities · Minsum objective

1 Introduction

In the open-shop scheduling problem, a given set of jobs must be processed on each of a given (sub-)set of machines for given amounts of time. The order in which jobs are processed is arbitrary, but at any point in time, each job must be processed on at most one machine, and each machine processes at most one job. This basic problem has a number of applications, e.g., in industrial testing and maintenance processes (Liaw, 2003), in garages for vehicle repairs (Gonzalez & Sahni, 1976), in medical centers to schedule surgeries and exams (Bai et al., 2016; Fei et al., 2009), and in satellite communications (Dell’Amico & Martello, 1996), just to name a few.

In this context, we tackle the following problem. Given are a set J of n jobs and a set M of m machines. Every job j is associated with a given release date r_j , a soft due date d_j , and a priority weight w_j . Each job j must be processed on a given subset of machines $M_j \subseteq M$, and processing a job $j \in J$ on a machine $i \in M_j$ takes p_{ij} time units of processing time. We refer to a job-machine pair (i, j) as an *operation*. For notational convenience, we also use binary parameter $\rho_{ij} \in \{0, 1\}$ to indicate if job $j \in J$ requires processing on machine $i \in M$, i.e., $M_j = \{i \in M \mid \rho_{ij} = 1\}$. It is possible that some jobs need not be processed on some machines, i.e., M_j may be a true subset of M for some or all $j \in J$. Jobs can run through the machines in arbitrary order; however, transferring a job from one machine to another entails machine–pair-dependent transportation delays $t_{ii'}$, for which we assume the triangle inequality holds. Every job starts at dummy machine 0, to which it must also eventually return, which entails no processing time (i.e., $p_{0j} = 0, \forall j \in J$) but possibly transportation delays t_{0i} (t_{i0}) to (from) machine $i \in M$.

We consider any regular objective function of the sum of job completion times, including nonlinear ones. In machine scheduling notation as introduced by Graham et al. (1979), this can be seen as a generalization of the problem expressed

✉ Simon Emde
simon.emde@uni-jena.de

Jens Lysgaard
lys@econ.au.dk

¹ Chair of Business Intelligence, Friedrich-Schiller-Universität Jena, Carl-Zeiß-Str. 3, 07743 Jena, Germany

² CORAL - Cluster for Operations Research, Analytics, and Logistics, Department of Economics and Business Economics, Aarhus University, Fuglesangs Allé 4, DK-8210 Aarhus V, Denmark

by the tuple $[O|t_{kl}, r_i|\sum f(C_j)]$, where $\sum f(C_j)$ denotes any regular minsum objective depending upon the completion time of the jobs, which may (e.g., total weighted tardiness) or may not (e.g., total completion time) be related to due dates d_j and weights w_j . In our computational study, we specifically consider the total number of late jobs ($\sum U_j$), i.e., jobs that complete processing past their due date, the sum of quadratic completion times ($\sum C_j^2$), and the total weighted tardiness ($\sum w_j T_j$), i.e., the total weighted time by which completion of the jobs exceeds their respective due date. These examples cover a fairly wide spectrum of regular minsum objectives, ranging from job count ($\sum U_j$), over nonlinear ($\sum C_j^2$), to weighted-sum ($\sum w_j T_j$). Note, however, that our problem definition is more general than the tuple suggests, as we allow jobs to be processed on only a subset of machines (as defined by the sets M_j). This is not equivalent to simply setting the processing time to zero on unused machines, due to transportation delays. We refer to this problem as the minsum open-shop scheduling problem with transportation delays, or MOSP-TD.

This problem definition is quite general and covers a lot of open-shop scheduling problems encountered in practice. It is particularly relevant in just-in-time logistics, where trucks (i.e., jobs) have to visit multiple facilities (i.e., machines) to offload (parts of) their cargo. Driving between facilities takes time (i.e., transportation delays), as does the unloading of the trucks at the facilities (i.e., processing time). Not all trucks may have to visit all facilities, and very commonly trucks have time windows (i.e., release and due dates). These practical cases are further discussed in Wirth and Emde (2018); Mejía and Yuraszcek (2020).

We contribute to the literature by proposing a flexible branch-cut-and-price algorithm capable of solving a broad spectrum of open-shop scheduling problems. To the best of our knowledge, branch-and-price has not yet been applied to open-shop scheduling problems. We show how Dantzig–Wolfe decomposition is applied to a time-indexed formulation and adapt subset-row inequalities from the vehicle routing literature to the open-shop scheduling problem. We demonstrate the good computational performance on diverse instance sets from the literature as well as newly generated data.

The remainder of this paper is organized as follows: In Sect. 2, we review the literature on related open-shop and truck scheduling problems. We formally define the problem and present mixed-integer programming (MIP) models in Sect. 3. Section 4 introduces our branch-cut-and-price scheme, which we test on benchmark instances from the literature and on newly generated instances in Sect. 5. Finally, Sect. 6 concludes the paper.

2 Literature review

Open-shop scheduling has received attention from academia since at least the 1970s. The literature and complexity results are recently surveyed by Anand and Panneerselvam (2015); Pinedo (2016); Ahmadian et al. (2021) and Strusevich (2022). Despite the indisputable practical relevance of open-shops, only few exact algorithms have been proposed compared to other machine environments like flow shops or job shops. This may be due to the fact that open-shop scheduling problems tend to be quite hard to analyze, let alone solve [Chapter 8.6 (Pinedo, 2016)].

One of the first exact methods is the branch-and-bound method proposed by Brucker et al. (1997) based on a disjunctive-graph formulation of the classic $[O||C^{\max}]$ problem with a makespan objective. The algorithm is improved by Guéret et al. (2000) by including a backtracking mechanism and by Dorndorf et al. (2001) by integrating constraint-propagation techniques to reduce the search space. More recently, Laborie (2005); Tamura et al. (2009); Malapert et al. (2012) propose constraint-programming approaches, and Ozolins (2021) use a method based on dynamic programming. Out of these exact methods, the constraint-programming approach by Malapert et al. (2012) generally yields the fastest results, solving instances with up to 20 machines and jobs. Note that all of these approaches aim at the classic open-shop with a makespan objective. To the best of our knowledge, a branch-and-price approach for the open-shop scheduling problem has not yet been proposed.

While branch-and-price has not been applied to the open-shop yet, it has seen widespread application to other types of machine environments like parallel (Kowalczyk & Leus, 2018) or single (Wang et al., 2018) machine workshops. The typical column generation scheme for parallel machine scheduling problems consists of pricing out schedules for individual machines (van Den Akker et al., 1999). However, the solution space of open-shop scheduling problems tends to be much larger than that of other machine environments because jobs consist of multiple operations that can be processed in arbitrary order on different machines, making an effective Dantzig–Wolfe decomposition difficult. Decomposition schemes for other types of machine scheduling problems are thus not easily adapted.

There are only a few published papers dealing with applying Dantzig–Wolfe decomposition to time-indexed scheduling models, where the planning horizon is divided into T discrete periods, similar to the approach we take in this paper. In their seminal paper, Van den Akker et al. (2000) apply a column generation technique to a single-machine scheduling problem, where columns correspond to pseudo-schedules that allow each job to occur not at all, once, or multiple times. The pricing problem is solved by finding a shortest path in a network consisting of $T + 1$ nodes, one for each time period.

This approach is further refined by Bigras et al. (2008). As Van den Akker et al. (2000) remark, an important advantage of this time-indexed formulation and decomposition is that it is relatively easy to consider different objective functions and side constraints.

Considering more than a single machine, Lancia et al. (2011) propose a column generation scheme for job shops using a similar decomposition based on a time-indexed model. Columns are priced out by a dynamic programming approach. Pessoa et al. (2010) extend the time-indexed model by adding another index for arcs, denoting the processing sequence of the jobs. While this increases the number of variables even further, it permits the generation of additional cuts such as extended capacity cuts. The authors embed this and several other algorithmic enhancements in a branch-cut-and-price framework to solve a parallel machine scheduling problem with a total weighted tardiness objective. The algorithm is able to solve instances with up to 100 jobs and 4 machines. This algorithm is further improved by Oliveira and Pessoa (2020). Recently, Bulhões et al. (2020) propose a branch-cut-and-price framework for parallel machine scheduling problems with time windows, setup times, and any minsum objective. Similar to Pessoa et al. (2010) and Oliveira and Pessoa (2020), the authors employ an arc-time-indexed formulation and solve single-machine pricing problems. Note that such a decomposition does not easily translate to an open-shop scheduling environment because jobs must be processed on more than one machine. More recently, Naderi et al. (2023) compare the performance of mixed-integer programming and constraint programming across a variety of shop scheduling problems including open-shop problems.

The vast majority of papers dealing with open-shop scheduling problems and, indeed, machine scheduling problems in general, consider minmax objectives, especially the makespan. Specifically, all of the exact algorithms reviewed above are either for minmax objectives or for different machine environments (single or parallel machine or else flow or job shops). Looking at the literature from a modeling perspective, Anand and Panneerselvam (2015) list several papers up to 2015 that consider open-shop scheduling problems with minsum objectives. Only a few publications propose an exact solution method beyond a default solver. Liu and Bulfin (1988) and Cheng and Shakhlevich (2005) develop polynomial-time algorithms for the unit execution time open shop with non-decreasing objective functions. The non-preemptive problem variants are special cases of the $[O|t_{kl}, r_i|\sum f(C_j)]$ problem we solve, but given that they are solvable in polynomial time under the particular assumption of unit processing times of all operations, there is no need for enumerative algorithms. The same also goes for Błażewicz et al. (2004), who propose polynomial-time methods for the special cases $[O|pmtn, r_i|Y_w]$ and $[O2|d_i =$

$d|Y]$, where Y (Y_w) denotes the total (weighted) work done after the due date without considering the quantity of the delay. The latter problem $[O2|d_i = d|Y]$ is a special case of $[O|t_{kl}, r_i|\sum f(C_j)]$, but, again, is not interesting for us in relation to this paper because this particular case can be solved efficiently without enumerative methods.

There are several papers that consider special cases of $[O|t_{kl}, r_i|\sum f(C_j)]$ other than polynomially solvable corner cases. Bai et al. (2017b) solve $[O||\sum w_j C_j]$, i.e., an open-shop scheduling problem with a total weighted completion time objective. They propose a “weighted shortest processing time block heuristic” and a hybrid discrete differential evolution algorithm, with which they solve instances with up to 200 jobs and 10 machines, or up to 80 jobs and 100 machines. These solution methods build on Tang and Bai (2010), who solve the unweighted version $[O||\sum C_j]$ with a constructive heuristic they dub “Shortest Processing Time Block”. The same problem $[O||\sum C_j]$ is also solved by Bräsel et al. (2008), who propose several constructive heuristics, which they test on instances with up to 50 jobs and 50 machines. Andresen et al. (2008) extend the previous paper with two metaheuristics (simulated annealing and genetic algorithm). Naderi et al. (2011) present models and metaheuristics (genetic algorithm and variable neighborhood search) for $[O||\sum T_j]$, solving instances with up to 20 jobs and machines.

Considering more complex scheduling environments, Wirth and Emde (2018) model a real-world logistics problem as an open-shop scheduling problem with release and due dates and transportation delays. They consider a total weighted tardiness objective in order to minimize the delays when receiving parts on factory premises in a just-in-time context. The article describes heuristic solution methods based on greedy adaptive randomized search (GRASP) and simulated annealing. Publications that discuss special cases of $[O|t_{kl}, r_i|\sum f(C_j)]$ are summarized in Table 1.

Another real-world problem modeled as an open-shop problem is considered recently by Zhuang et al. (2021), where the planning of multi-robot systems in an order picking warehouse is described as an open-shop problem with sequence-dependent setup and transportation times, with the objective to minimize makespan. Zhuang et al. (2021) provide a mixed-integer linear programming model for small instances and a hybrid artificial bee colony heuristic for larger instances. Mejía and Yuraszeck (2020) is also similar to the scheduling environment we discuss in this paper, seeing that there are transportation delays in order to manage the transportation of goods from suppliers to company warehouses in a speedy manner. However, all of these papers deal with a makespan objective, which is not a minsum objective and hence not a special case of $[O|t_{kl}, r_i|\sum f(C_j)]$.

Recently, Xiao et al. (2022) develop a heuristic based on column generation to solve a generalization of the open-shop

Table 1 Publications dealing with special cases of $[O|t_{kl}, r_i| \sum f(C_j)]$

Reference	Problem	Algorithm	Max $n \times m$
Bräsel et al. (2008)	$[O \sum C_j]$	Constructive heuristics	50×50
Andresen et al. (2008)	$[O \sum C_j]$	Metaheuristics	50×50
Tang and Bai (2010)	$[O \sum C_j]$	Constructive heuristic	1000×10
Naderi et al. (2011)	$[O \sum T_j]$	Metaheuristics	20×20
Bai et al. (2017b)	$[O \sum w_j C_j]$	Metaheuristic	200×10 and 80×100
Wirth and Emde (2018)	$[O t_{kl}, r_i \sum w_j T_j]$	Metaheuristics	120×40

Table 2 Notation for the MIP model adapted from the literature

Sets	
J	Set of jobs, $J = \{1, \dots, n\}$
M	Set of machines, $M = \{1, \dots, m\}$
Parameters	
T	Number of time periods in the planning horizon
r_j	Release date of job $j \in J$, $r_j \in \mathbb{N}_0$
d_j	Due date of job $j \in J$, $d_j \in \mathbb{N}$, $d_j > r_j$
w_j	Weight of job j in the objective function, $w_j \in \mathbb{N}_0$
p_{ij}	Processing time of job $j \in J$ on machine $i \in M$, $p_{ij} \in \mathbb{N}_0$
ρ_{ij}	1, if job $j \in J$ requires processing on machine $i \in M$; 0, otherwise
$t_{ii'}$	Transportation delay from machine i to machine i' , $t_{ii'} \in \mathbb{N}_0$
t_{0i} (t_{i0})	Transportation delay from (to) initial (final) machine 0 to (from) machine i , $t_{0i}, t_{i0} \in \mathbb{N}_0$
$\mathcal{M}$	Sufficiently large integer
$f(j, t)$	Contribution to the objective function when job j finishes at time t
Variables	
$x_{ijj'}$	1, if job j precedes job j' on machine i ; 0, otherwise
$\alpha_{jii'}$	1, if machine i precedes machine i' when processing job j ; 0, otherwise
τ_{ij}	Time when job j starts processing on machine i
C_{jt}	1, if job j finishes at time t ; 0, otherwise

scheduling problem with transfer times, reporting low optimality gaps averaging 0.08%. Referring to an application in healthcare services, patients (i.e., jobs) must visit a set of therapists (i.e., machines), where moving between therapists requires travel (i.e., transfer) time. The goal is to minimize the sum of makespans over all patients, i.e., the total difference between start and end times of treatment. While the problem formulation and decomposition bear some similarity to our approach, there are three major differences: First, although the objective in Xiao et al. (2022) is a *minsum* objective, it is not *regular*, because delaying a completion time does not necessarily worsen the objective value if the corresponding start time is also delayed. Second, the treatment of time windows differs in that Xiao et al. (2022) consider time windows on the therapists (i.e., machines), whereas MOSP-TD considers time windows on the jobs. Third, beyond a default solver, Xiao et al. (2022) propose no exact method, and the pricing problem is solved heuristically via a genetic algorithm, whereas we propose an exact method with guar-

anteed convergence. Consequently, while the heuristic from the literature yields strong upper bounds, it is neither easily adaptable to MOSP-TD due to a disparity in objectives and constraints, nor is our method straightforwardly applicable to the problem of Xiao et al. (2022), due to the irregularity of their objective.

The problem definition in Wirth and Emde (2018) comes closest to what we address in this paper, integrating a minsum objective, transportation delays, and time windows, based on a real-world scheduling problem on factory premises. Since their problem constitutes a special case of MOSP-TD, we reuse their instances as part of our computational tests (Sect. 5.2).

3 Mixed-integer programming models

In this section, we formalize MOSP-TD, first, by adapting the MIP model from Wirth and Emde (2018) and then as

a generalized set-partitioning model, which forms the basis of a column generation approach. Note that we assume for the following models and procedures that all parameters are integer. This can always be imposed by scaling real-valued data to arbitrary precision.

3.1 Compact MIP model

Using the notation in Table 2, we adapt the model by Wirth and Emde (2018) to account for arbitrary regular minsum objectives as follows.

$$\text{Minimize } \mathcal{F}(\mathbf{x}, \boldsymbol{\tau}, \boldsymbol{\alpha}, \mathbf{C}) = \sum_{j \in J} \sum_{t=r_j}^T f(j, t) \cdot C_{jt} \quad (1)$$

subject to

$$\sum_{t=r_j}^T C_{jt} = 1 \quad \forall j \in J \quad (2)$$

$$\begin{aligned} \sum_{t=r_j}^T t \cdot C_{jt} - \tau_{ij} &\geq p_{ij} + t_{i0} & \forall i \in M_j; j \in J & \quad (3) \\ \tau_{i'j} - \tau_{ij} + (1 - \alpha_{jii'}) \cdot \mathcal{M} &\geq p_{ij} + t_{ii'} & \forall j \in J; i, i' \in M_j; i \neq i' & \quad (4) \\ \alpha_{jii'} + \alpha_{ji'i} &= 1 & \forall j \in J; i, i' \in M_j; i \neq i' & \quad (5) \\ \tau_{ijj'} - \tau_{ij} + (1 - x_{ijj'}) \cdot \mathcal{M} &\geq p_{ij} & \forall j, j' \in J; j \neq j'; i \in M_j \cap M_{j'} & \quad (6) \\ x_{ijj'} + x_{ij'j} &= 1 & \forall j, j' \in J; j \neq j'; i \in M_j \cap M_{j'} & \quad (7) \\ x_{ijj'} &\in \{0; 1\} & \forall j, j' \in J; j \neq j'; i \in M_j \cap M_{j'} & \quad (8) \\ \tau_{ij} &\geq r_j + t_{0i} & \forall j \in J; i \in M_j & \quad (9) \\ \alpha_{jii'} &\in \{0; 1\} & \forall j \in J; i, i' \in M_j; i \neq i' & \quad (10) \end{aligned}$$

Objective function (1) minimizes any objective function related to the total completion time of the jobs. This includes, e.g., the total weighted tardiness objective of Wirth and Emde (2018) if $f(j, t) = w_j \cdot \max\{0, t - d_j\}$ and also nonlinear objectives like the total k -power completion time (Bai et al., 2017a) if $f(j, t) = t^k$. Constraints (2) and (3) set the completion time to a single value that cannot be less than the time when the job returns to machine 0. Constraints (4) and (5) enforce non-overlapping processing times on the machines, and (6) and (7) do the same for the jobs. Finally, (8) through (10) define the domain of the variables.

While this MIP model is quite flexible, allowing for many different objective functions, it suffers from a large number of variables as well as complicating big-M constraints, which are notoriously difficult for default solvers to handle [e.g., (Codato & Fischetti, 2006)]. Note that it is possible to formulate a time-indexed MIP model for MOSP-TD without

big-M constraints; however, this requires a much larger number of constraints. In our preliminary experiments, Gurobi was unable to even generate (let alone solve) such a model for larger instances. To alleviate these issues, we propose to reformulate the problem as a generalized set-partitioning model, which we use as the foundation of a column generation approach.

3.2 Set-partitioning model

Using the additional notation in Table 3, we formulate MOSP-TD as a generalized set-partitioning problem (GSP) as follows. The GSP model requires a set of feasible single-job schedules S as an input. Every schedule $s \in S$ determines the start times τ_{ijs} , $\forall i \in M_j$, of the job $j_s \in J$ that schedule s refers to. Then, it is straightforward to calculate binary parameter d_{sit} for every schedule $s \in S$, indicating whether job j_s is processed on machine $i \in M$ at time $t \in T$. Analogously, binary parameter e_{sj} indicates whether schedule s refers to job $j \in J$, i.e., if $j = j_s$.

$$\text{Minimize } G(\boldsymbol{\lambda}) = \sum_{s \in S} c_s \cdot \lambda_s \quad (11)$$

subject to

$$\sum_{\substack{s \in S: \\ e_{sj}=1}} \lambda_s = 1 \quad \forall j \in J \quad (12)$$

$$\sum_{s \in S} d_{sit} \cdot \lambda_s \leq 1 \quad \forall i \in M, t = 1, \dots, T \quad (13)$$

$$\lambda_s \in \{0; 1\} \quad \forall s \in S \quad (14)$$

Variable λ_s indicates whether schedule s is used. A schedule, in turn, defines the start times of one specific job $j_s \in J$ on all machines in M_j on which job j_s requires processing. Objective function (11) minimizes the total cost of all selected schedules, whereas Constraints (12) enforce that

Table 3 Additional notation for the generalized set partitioning model

Sets	
S	Set of single-job schedules
Parameters	
c_s	Objective contribution of schedule $s \in S$
d_{sit}	1, if schedule $s \in S$ is processed on machine i at time t ; 0, otherwise
e_{sj}	1, if schedule $s \in S$ is for job $j \in J$; 0, otherwise
Variables	
λ_s	1, if schedule $s \in S$ is selected; 0, otherwise

exactly one schedule must be chosen for each job and Constraints (13) ensure that no two jobs occupy any machine at the same time. Finally, a schedule is either selected or not, according to Constraints (14).

The model has a neater structure than the MIP model from Sect. 3.1, without any complicated (big-M) constraints. However, the number of variables is even larger, as there can be an astronomically large number of schedules in set S . Moreover, the model is still time-indexed because of Constraints (13). We deal with this in Sect. 4.

3.3 Example

To illustrate MOSP-TD, consider an example problem with $n = 3$ jobs, $m = 2$ machines and a planning horizon of $T = 14$, where every job must be processed on every machine, i.e., $\rho_{ij} = 1, \forall i \in M, j \in J$, and machine $i = 0$ is the dummy machine where every job starts and ends. The job characteristics and machine transfer times are given in Table 4.

Assuming that the objective is to minimize the total weighted tardiness, i.e., $f(C_j) = w_j \cdot \max\{C_j - d_j; 0\}$, an optimal solution with objective value 13 (jobs 2 and 3 finish seven and six time units past their due date, respectively) is depicted as a Gantt chart in Fig. 1. In the GSP model, this solution corresponds to three schedules $s_1 = \{(1, 3), (2, 6)\}$ for job $j_{s_1} = 1$, $s_2 = \{(1, 8), (2, 4)\}$ for job $j_{s_2} = 2$, and $s_3 = \{(1, 5), (2, 9)\}$ for job $j_{s_3} = 3$ being selected, where a tuple (i, t) indicates that job j_s starts on machine i at time t .

4 Branch-cut-and-price

Due to the potentially enormous size of schedule set S , it is more practical to only consider a subset $\bar{S} \subset S$ of schedules and generate additional schedules from set $S \setminus \bar{S}$ iteratively in a delayed column generation framework. For this, we consider the LP-relaxation of the set-partitioning model from Sect. 3.2 with only a subset $\bar{S}$ of columns. We refer to this model as the *restricted master problem* (RMP). We discuss the ensuing pricing problem in Sect. 4.1, and we

Table 4 Example problem data

j	1	2	3
(a) Job characteristics			
p_{1j}	1	2	2
p_{2j}	2	2	1
r_j	2	1	4
d_j	12	4	7
w_j	1	1	1
(b) Machine transfer delays			
i	0	1	2
t_{0i}	0	1	3
t_{1i}	1	0	2
t_{2i}	3	2	0

adapt the famous subset-row inequalities originally proposed by Jepsen et al. (2008) for the vehicle routing problem to MOSP-TD in Sect. 4.2. We present a simple primal heuristic for generating initial columns in Sect. 4.3. Finally, solutions to the RMP may not be integral. To enforce integrality, we embed the column generation scheme into a branch-and-bound framework to create a branch-cut-and-price method in Sect. 4.4..

4.1 Pricing problem

Since the RMP does not consider all possible schedules but only a subset $\bar{S}$, additional columns must be priced out. Let $\pi_j \in \mathbb{R}$ and $\mu_{it} \leq 0$ be the dual variables associated with Constraints (12) and (13), respectively. Then, the reduced cost of variable λ_s is:

$$\tilde{c}_s = c_s - \pi_{j_s} - \sum_{i \in M_j} \sum_{t=1}^T d_{sit} \cdot \mu_{it}.$$

The pricing problem consists of finding a column in set $S \setminus \bar{S}$ with negative reduced cost or proving that no such column exists. For a given job $j \in J$, we can interpret finding the best starting times on the machines as a shortest path problem through a network $N(j)$, which is schematically

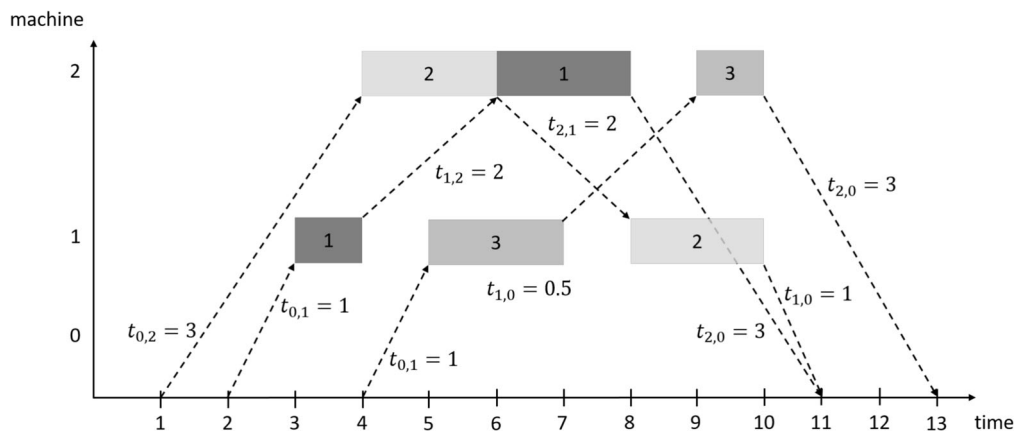


Fig. 1 Gantt chart of the optimal solution for the min total tardiness objective in the example

depicted in Fig. 2. The network consists of $O(m \cdot T)$ nodes, where each node (i, t) denotes that job j starts processing on machine $i \in M_j$ at time $t \in \{r_j, \dots, T\}$. Starting from dummy node $(0, r_j)$ and ending at dummy node $(0, T)$, an arc from node (i, t) to some other node (i', t') signifies that job j is processed on machine i' directly after machine i at times t' and t , respectively. Such an arc only exists if

$$t' \geq t + p_{ij} + t_{ii'} \wedge t' + p_{i'j} \leq T$$

holds. The cost of an arc $((i, t), (i', t'))$ equals

$$\hat{c}((i, t), (i', t')) = \begin{cases} f(t + p_{ij} + t_{i0}), & \text{if } (i', t') = (0, T) \\ -\sum_{t''=t'}^{t'+p_{i'j}-1} \mu_{i't''}, & \text{otherwise} \end{cases}.$$

Any path through the network from source $(0, r_j)$ to sink $(0, T)$ that contains exactly one node for each machine $i \in M_j$ constitutes a feasible schedule $s \in S$. Let $\check{c}(s)$ be the total cost of the path associated with schedule s . Then, the reduced cost of schedule s is

$$\tilde{c}_s = \check{c}(s) - \pi_{j_s}.$$

Finding a minimum reduced cost schedule is equivalent to solving an *elementary shortest path problem with resource constraints* (ESPPRC) in this network, which we solve with a label setting algorithm [surveyed, e.g., in Irnich and Desaulniers (2005)] as follows.

A label is associated with a partial path in the network and is defined by the tuple (P_v, Q_v) , where P_v stands for the dual cost of the partial schedule ending at node $v = (i, t)$ and Q_v is the set of already visited machines. Starting from node $(0, r_j)$ with label $(0, \emptyset)$, a label (P_v, Q_v) is extended along arc $((i, t), (i', t')) = (v, v')$ to create a new label $(P_{v'}, Q_{v'})$

using the following resource extension functions (REF).

$$P_{v'} = \text{REF}_{vv'}^C(P_v, Q_v) = P_v + \hat{c}((i, t), (i', t'))$$

$$Q_{v'} = \text{REF}_{vv'}^M(P_v, Q_v) = Q_v \cup \{i'\}.$$

This new label is feasible only if it satisfies the following conditions.

- $t' + p_{i'j} + \sum_{i'' \in M_j \setminus Q_{v'}} p_{i''j} \leq T$, i.e., the remaining operations can potentially still be scheduled before the end of the planning horizon.
- $i' \notin Q_v$, i.e., machine i' is not yet part of the partial schedule.

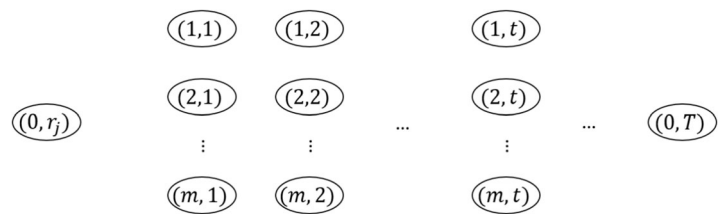
For two different labels (P_v, Q_v) and $(P_{v'}, Q_{v'})$ for the same node v , we apply the following dominance rule. Label (P_v, Q_v) dominates label $(P_{v'}, Q_{v'})$ if $P_v \leq P_{v'}$ and $Q_v \supseteq Q_{v'}$. All dominated labels are discarded, except if the dominance relationship is not strict, i.e., both $P_v = P_{v'}$ and $Q_v = Q_{v'}$, then one label is retained.

Lemma 1 *The dominance rule is valid.*

Proof For the rule to be valid, it must hold that for every possible extension of $(P_{v'}, Q_{v'})$, there exists an extension of (P_v, Q_v) that yields a solution that is at least as good.

Since $Q_v \supseteq Q_{v'}$, in the best case, the same set of operations must still be scheduled from both labels, starting from the same node $v = (i, t)$. Likewise, since $P_v \leq P_{v'}$, at best, both labels are associated with the same cost $P_v = P_{v'}$. Extending the labels further cannot lead to lower cost for label $(P_{v'}, Q_{v'})$, because arc cost function $\hat{c}((i, t), (i', t'))$ is monotonically increasing. Objective function f is by definition regular, i.e., delaying completion of a job cannot improve the objective value, and dual variable μ_{it} is non-positive because it is associated with inequalities (13). Intuitively speaking, blocking some machine i at some time t can

Fig. 2 Schematic depiction of the nodes in network $N(j)$



incur a dual penalty, but never a reward. Since extending (P'_v, Q'_v) requires the scheduling of at least the same operations $Q_v \supseteq Q'_v$ as label (P_v, Q_v) , it is not possible to choose more favorable starting times or an earlier completion time for the extension of (P'_v, Q'_v) than what extending (P_v, Q_v) can also achieve. $\square$

To further reduce the computational effort, we consider the following points, which follow directly from the monotonicity of arc cost function $\hat{c}((i, t), (i', t'))$ as outlined in the proof of Lemma 1.

- A label (P_v, Q_v) can be discarded if it cannot lead to negative reduced cost, i.e., if $P_v + f(t + p_{ij} + \sum_{i'' \in M_j \setminus Q_v} p_{i''j} + t_{i0}) \geq \pi_j$. Analogously, if a complete negative reduced cost schedule has already been found, we can replace π_j on the right-hand side with the best known upper bound.
- For a pair of labels (P_v, Q_v) and $(P'_{v'}, Q'_{v'})$ where $t' > t$, $i = i'$, $\sum_{t''=t}^{t+p_{ij}-1} \mu_{it''} \geq \sum_{t''=t'}^{t'+p'_{ij}-1} \mu_{i't''}$, and $Q_v \supseteq Q_{v'}$, label $(P'_{v'}, Q'_{v'})$ can be discarded because delaying an operation on the same machine cannot make sense if there is no reward in terms of lower dual cost.
- The dual cost $\sum_{t''=t}^{t'+p_{ij}-1} \mu_{it''}$ of blocking some machine i from time t to time t' can be memorized for all $i \in M$, $1 \leq t < t' \leq T$ and re-used for all $j \in J$ until the dual prices change, i.e., the RMP is re-solved. Hence, there is no need to sum over μ_{it} every time a label is extended.

The minimum reduced cost schedule can finally be recovered from the label of sink node $(0, T)$. If no such label exists (because all labels along the way have been discarded), there is no schedule for job j with negative reduced cost. Otherwise, the schedule is added to set $\tilde{S}$ of the RMP. Either way, the process is repeated for each job $j \in J$.

Note that for the column generation algorithm to converge, it is not necessary to find the very best schedule for job j . It is sufficient to find *any* schedule as long as it is feasible and has negative reduced cost. Therefore, we prune the network heuristically: for every node (i, t) , we only consider arcs to nodes $(i', t + p_{ij} + t_{i'v} + v)$, where $v \in \{0, 1, 2, 3, 5, 7, 10, 14, 19, \dots\}$ is an offset reducing the number of outgoing arcs per node. We determine v from the set of Padovan numbers (Stewart, 1996), that is, the increase in v is equivalent to the Padovan series, because in good

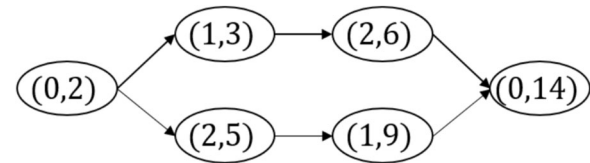


Fig. 3 Paths through network $N(1)$ in the example

schedules, we expect the waiting time between operations to be generally low, whereas for long waiting delays, a lower degree of granularity is sufficient. If no negative reduced cost schedule can be found in the reduced network, then it is still necessary to solve the ESPRC on the original network to prove the non-existence of such a schedule.

Example (cont.): Consider network $N(1)$ for job $j = 1$ from the example in Sect. 3.3. Assuming that all dual prices are $\mu_{it} = 0$, $\forall i \in \{1, 2\}, t = 1, \dots, 14$, Fig. 3 depicts the only two non-dominated paths through the network. In this example, either path is optimal. Path $\langle (0, 2), (1, 3), (2, 6), (0, 14) \rangle$ encodes the schedule for job 1 which is depicted in Fig. 1. Note that if we assume that there is, e.g., a dual price $\mu_{2,7} < 0$ to pay for using machine 2 at time 7, then we must consider one more node in the network, namely $(2, 8)$, which is connected via an arc from node $(1, 3)$. Note that node $(2, 7)$ need not be considered (i.e., has no undiscarded labels) because it offers a reduction neither in dual cost nor in objective value compared to $(2, 6)$. In this case, the bottom path is shorter.

4.2 Valid inequalities

To speed up convergence, we adapt *subset-row inequalities*, originally proposed for the vehicle routing problem with time windows by Jepsen et al. (2008). Since then, they have been applied to many different problems, especially routing problems, with great success [e.g., Baldacci et al. (2011a); Gschwind et al. (2020)]. We restrict ourselves to considering subsets of cardinality 3 of Constraints (13).

Formally, the subset-row inequalities for MOSP-TD are defined as follows.

$$\sum_{s \in S} \lambda_s \cdot \lfloor \beta_s^V / 2 \rfloor \leq 1, \forall V \subset \{(i, t) \mid i \in M, t = 1, \dots, T\}$$

where $|V| = 3$,

where β_s^V is the number of machine-time pairs from set V that schedule s occupies. Since we only consider elementary schedules, $\lfloor \beta_s^V / 2 \rfloor$ is one if schedule s occupies two or three of the time-machine pairs in V , zero otherwise. In plain words, these constraints enforce that at most one schedule may occupy two or three of the machines at the specified times in set V . Note that this must be true for any feasible integer solution; otherwise at least two jobs would use the same machine at overlapping times.

These constraints tighten the LP-relaxation of the set-partitioning model, but their number can be extremely large. We therefore separate these inequalities iteratively in the column generation process by enumerating all subsets of time-machine pairs with cardinality three and checking which inequalities are violated by the current solution. The violated inequalities are then added to the RMP.

Unlike the VRP, where these inequalities originate, we have to contend with machine-time pairs in MOSP-TD, which can be very numerous. Enumerating all possible subsets of three can therefore take a long time. For this reason, we restrict the enumeration of machine-time pairs to a maximum of 15,000 randomly chosen ones and do not separate more than 64 inequalities.

The subset-row inequalities affect the pricing problem. We add another element to the labels to keep track of how many machine-time pairs for each inequality have been occupied by a given (partial) schedule. When the number reaches two, the dual price of the corresponding inequality is added to the transition cost. These labels and the additional dual variables must also be considered in the dominance criteria (a label cannot dominate another label if the former occupies only a subset of the relevant time-machine pairs of the latter) and when comparing the dual cost of starting an operation at specific points in time. (The potential dual cost of the subset-row inequalities must also be considered). Details about how to incorporate SRI into ESPPRC are discussed in Jepsen et al. (2008); Desaulniers et al. (2008).

Example (cont.): Consider three schedules $s_1 = \{(1, 7), (2, 10)\}$, $s_2 = \{(1, 6), (2, 12)\}$, and $s_3 = \{(1, 5), (2, 11)\}$ for jobs $j_{s_1} = 1$, $j_{s_2} = 2$, and $j_{s_3} = 3$, respectively. Clearly, only one of these three schedules can be used in an integer solution because they overlap on machine 1 at times 6 and 7 and on machine 2 at time 11. However, in the LP-relaxed RMP, the variable assignments $\lambda_{s_1} = \lambda_{s_2} = \lambda_{s_3} = 0.5$ (among others) violate no constraint. The subset-row inequality for $V = \{(1, 6), (1, 7), (2, 11)\}$ eliminates this solution.

4.3 Initial columns

To obtain an initial upper bound and set of columns, we use a simple list scheduling-based constructive heuristic, followed by a greedy local search phase. Both of these heuristics phases operate on a sequence of operations $\theta =$

$\langle (j_1, i_1), \dots, (j_o, i_o) \rangle$, where $o = \sum_{j \in J} |M_j|$ is the total number of operations and (j_k, i_k) indicates the k -th operation to be scheduled. A sequence θ is decoded by going through it element by element, starting at the first element $k = 1$ and scheduling job j_k on machine i_k as early as feasible, i.e., either as soon as machine i is available or job j can transfer to that machine, whichever comes last.

An initial sequence is generated purely randomly. A neighborhood is defined by swapping two operations in the incumbent solution. The best neighborhood solution is accepted and becomes the new incumbent. The search ends when no solution in the entire neighborhood improves upon the incumbent. This procedure is similar to the local search method presented in Wirth and Emde (2018), which the authors show to yield decent optimality gaps at short run-times.

While these initial columns are guaranteed to constitute a feasible solution, some or all of these columns may be cut off during branching (see Sect. 4.4). Therefore, to ensure that the RMP has at least one feasible solution in all branches, we include a dummy column s^d that covers all jobs without blocking any machine, i.e., $e_{s^d j} = 1$ and $d_{s^d i t} = 0$, $\forall j \in J, i \in M_j, t = 1, \dots, T$, and has a prohibitively large coefficient in the objective function. If no column with negative reduced cost exists and this column is still in the basis with a positive value, the node can be fathomed.

Example (cont.): Starting from random sequence $\theta = \langle (1, 2), (2, 2), (3, 1), (1, 1), (2, 1), (3, 2) \rangle$, the best neighborhood move is to swap operations $(1, 2)$ and $(1, 1)$, leading to a sequence which decodes to the locally—and in this case globally—optimal solution depicted in Fig. 1.

4.4 Branching

Since variables λ are continuous in the RMP, the objective value of the optimal RMP solution may only be a lower bound because not all variables may have integral values. In this case, branching may be required to ensure integrality.

Let $\bar{\lambda}_s$ be the value of variable λ_s . Let (i, j) , $j \in J$, $i \in M_j$, be the operation of job j on machine i , and let $h_{sit} \in \{0, 1\}$ be a binary parameter, indicating whether schedule s starts an operation on machine i at time t . We define

$$(i^*, j^*) \in \arg \max_{(i, j) \in M \times J: \rho_{ij}=1} \{ \{t = 1, \dots, T \mid \exists s \in S : 0 < \bar{\lambda}_s < 1 \wedge e_{sj} = 1 \wedge h_{sit} = 1\} \} \quad (15)$$

as the operation whose starting time is spread out across the most fractional time periods. In case multiple operations maximize expression (15), ties are broken randomly. Let (i^*, j^*) be undefined if $\bar{\lambda}$ is integral. In this case, no branching is necessary.

Let $\bar{y}_{i^*j^*t} = \sum_{s \in S: e_{sj^*}=1} h_{si^*t} \cdot \bar{\lambda}_s$ be the (fractional) number of times operation (i, j) starts at time t . Let

$$t^* \in \arg \min_{t=1, \dots, T} \left\{ \sum_{t'=1}^t \bar{y}_{i^*j^*t'} > 0.5 \right\} \quad (16)$$

be the point in time such that about half the fractional start times of operation (i^*, j^*) are sooner and about half are later than t^* .

We branch on the starting time of operations. In the zero branch, operation (i^*, j^*) must start sooner than time t^* . In the one branch, (i^*, j^*) must start no earlier than time t^* . In either branch, we can enforce these branching constraints by removing any schedules that violate them from set $\bar{S}$ in the RMP and removing the affected nodes (i^*, t) where $t \geq t^*$ or $t < t^*$, respectively, from the network $N(j^*)$ in the pricing problem.

Example (cont.): Consider the schedules $s_1 = \{(1, 3), (2, 6)\}$, $s_2 = \{(1, 8), (2, 4)\}$, $s_3 = \{(1, 5), (2, 9)\}$, and $s_4 = \{(1, 3), (2, 7)\}$ for jobs $j_{s_1} = j_{s_4} = 1$, $j_{s_2} = 2$, and $j_{s_3} = 3$, respectively, where $\lambda_{s_1} = \lambda_{s_4} = 0.5$ and $\lambda_{s_2} = \lambda_{s_3} = 1$. This schedule also leads to an objective value of 13 and is feasible for the RMP, but it is not integral and so requires branching. The operation (i^*, j^*) that has the most fractional start times is $(2, 1)$. This leads to $t^* = 7$, i.e., in one branch job 1 must start on machine 2 before time 7, and in the other branch job 1 must start on machine 2 no sooner than time 7.

Lemma 2 *Using the above branching rule, BCP converges to an optimal integer solution in a finite number of steps.*

Proof If (i^*, j^*) is defined, i.e., the current solution is not integral, then by Constraints (12) there must be multiple schedules $S' \subset S$ for at least one job $j_s = j^*$ with $0 < \lambda_s < 1$, $\forall s \in S'$, and $\sum_{s \in S'} \lambda_s = 1$. For at least one machine $i^* \in M_{j^*}$, there must be at least two different fractional start times $\tau_{i^*j^*}$ for job j^* on machine i^* defined by S' , because two different schedules in S' for the same job j^* cannot have the exact same start time on every machine, otherwise the schedules would not be different. Determining start time t^* as per Eq. (16) ensures that at least one of the schedules in S' must have a (fractional) start time $\tau_{i^*j^*} < t^*$ and at least one schedule must have a (fractional) start time $\tau_{i^*j^*} \geq t^*$.

Branching on the start time t^* hence ensures that at least two of the fractional variables λ_s , $s \in S'$, that currently start job j^* on machine i^* at different times $\tau_{i^*j^*}$ cannot be part of the same solution anymore. The solutions in child nodes must consequently be different than in the parent node in the BCP tree. At the same time, no solution that is feasible for the original problem is cut off: any feasible (and potentially optimal) start time $\tau_{i^*j^*}$ is still in one of the branches.

Table 5 Instance parameters for instances from Wirth and Emde (2018)

Instance size	Small	Medium	Large
Number of machines m	20	30	40
Number of jobs n	40, 50, 60	60, 75, 90	80, 100, 120
Number of instances	18	18	18

In the worst case, if every single node is branched, the branching constraints in every leaf node will eventually fully determine the integer start time t of every job j on every machine $i \in M_j$. The number of potential branches for every machine–job pair (i^*, j^*) is bounded by the number $O(2^T)$ of potential start times. The optimal solution must necessarily be in one of these leaf nodes, which concludes the proof. $\square$

5 Computational study

In this section, we test our branch-cut-and-price (BCP) scheme on three different benchmark data sets and compare it against the best heuristic from the literature as well as a state-of-the-art default solver. We describe our test bed in Sect. 5.1, investigate the effectiveness of the subset-row inequalities in Sect. 5.2.1 and the algorithmic performance in Sect. 5.2.2.

5.1 Benchmark instances and computational environment

Since the MOSP-TD is a general problem, we use three different benchmark data sets, two from the literature and one newly generated. We describe the test data in more detail in the following.

Wirth and Emde (2018) discuss a special case of the MOSP-TD in the context of routing trucks on factory premises to deliver parts to a production plant. They consider a total weighted tardiness objective; otherwise, their problem definition is identical to MOSP-TD. Consequently, we reuse their original instances without any modification. The instances are divided into three classes: small (up to 20 machines and 60 jobs), medium (up to 30 machines and 90 jobs), and large (up to 40 machines and 120 jobs), where not all jobs need to be processed on all machines. There is a total of 54 instances in the set, which is available via DOI <https://doi.org/10.5281/zenodo.5196633>. The data are summarized in Table 5.

Since the MOSP-TD is an open-shop scheduling problem, it is only natural to use classic open-shop instances from the literature. Specifically, the instance set of Brucker et al. (1997) still counts as one of the most difficult classic benchmarks, even with a makespan objective (Ahmadian et al., 2021; Malapert et al., 2012). The data set contains a total

Table 6 Summary of the data sets used in this study

Symbol	Count	Source	Original problem	Modifications	Max. size ($n \times m$)
R10x10	20	This paper	$[O t_{kl}, r_i \sum C_j^2]$	-	10×10
R20x20	20	This paper	$[O t_{kl}, r_i \sum C_j^2]$	-	20×20
WE_small	18	Wirth and Emde (2018)	$[O t_{kl}, r_i \sum w_j T_j]$	-	60×20
WE_medium	18	Wirth and Emde (2018)	$[O t_{kl}, r_i \sum w_j T_j]$	-	90×30
WE_large	18	Wirth and Emde (2018)	$[O t_{kl}, r_i \sum w_j T_j]$	-	120×40
j*-per*	52	Brucker et al. (1997)	$[O C^{\max}]$	Release and due dates, transportation delays, objective function $\sum U_j$	8×8

of 52 instances, ranging from small ones with 3 jobs and 3 machines up to 8 jobs and 8 machines.

To make the instances more challenging, we enrich them with transportation delays by giving each machine a 2D-coordinate (x_i, y_i) , where x_i and y_i are randomly drawn integers from a uniform distribution between zero and nine. We also add a dummy initial machine 0, which is always at $(x_0, y_0) = (0, 0)$. Delays between machines are then measured via the Manhattan metric, i.e., $t_{ii'} = |x_i - x_{i'}| + |y_i - y_{i'}|$, $\forall i, i' \in M \cup \{0\}$. Let $P_1 = \sum_{i \in M} p_{i1}/m$ be the average processing time of the first job. We set the release date of job j to $r_j = U(0, P_1)$ and the due date to $d_j = r_j + U(m/4 \cdot P_1, 2 \cdot m \cdot P_1)$, where U stands for a uniformly distributed random integer from the interval in the argument. As an objective, we minimize the total number of late jobs, i.e., we solve $[O|r_j, t_{kl}| \sum U_j]$. While the minimization of the number of tardy jobs is widely recognized as an important goal in many scheduling applications [e.g., Ho and Chang (1995)], to the best of our knowledge, disregarding some investigations into special cases with bounded numbers of machines and / or unit processing times, only Ng et al. (2003) have applied a $\sum U_j$ goal in an open-shop scheduling context, presenting some theoretical results. Our branch-cut-and-price algorithm is hence one of the first published exact solution procedures to explicitly solve open-shop scheduling problems with this objective at all.

Finally, we also generate two sets of twenty new instances each, one small one with $n = m = 10$ jobs and machines, and one large one with $n = m = 20$ jobs and machines. We generate the machine transfer delays and release dates the same way as described above. Each job must be processed on between one and six randomly chosen machines, where it requires a randomly chosen processing time (uniformly distributed integer) from the interval $[5, 17]$, the same as in Raut et al. (2008). For this instance set, we consider a nonlinear objective, namely minimizing the sum of quadratic completion times, a special case of the k -power completion time (Bai et al., 2017a).

This gives us a diverse set of instances, totaling $54 + 52 + 40 = 146$ instances, which are summarized in Table 6. For all instances, we set the planning horizon T to the makespan of the initial solution (see Sect. 4.3) plus two times the maximum processing time of any operation. The instance set can be downloaded via DOI <https://doi.org/10.5281/zenodo.5196711>. All tests are run on an x64-PC with an AMD Ryzen 5 3600 CPU with a 3.6-GHz base clock. Our BCP scheme is implemented in C# 8.0, and Gurobi 9.0.2 is the default solver for the RMP and the single-model MIP from Sect. 3.1. All programs are run on a single thread with a one-hour time limit to ensure comparability.

Concerning the choice of benchmark solvers, given that some of the most successful solution methods for the classic open-shop with a makespan objective are based on constraint programming [e.g., Malapert et al. (2012)], we also attempted to solve the instances from Wirth and Emde (2018) with a state-of-the-art CP solver, namely IBM ILOG CP Optimizer. However, the results turned out to be almost universally significantly poorer than those of the MIP solver, both in terms of bounds and runtimes. We therefore only print the results for the MIP solver as a benchmark in the following.

5.2 Computational results

5.2.1 Effect of subset-row inequalities

First, we investigate the effect of the subset-row inequalities (SRI) as elaborated in Sect. 4.2. Table 7 shows the results on the random instances with and without SRI. Column *Root LB* denotes the lower bound in the root node, while *Best LB* is the lower bound after branching. Columns *Root s* and *Total s* stand for the CPU time in seconds for solving the root node and the whole enumeration tree, respectively. *Nodes* and *Columns* list the total number of nodes in the branch-and-bound tree and the total number of generated columns, respectively. Note that we list the best bounds, which are

Table 7 Results on the random R10x10 and R20x20 instances with and without SRI; instances are labeled “R $n \times m_k$ ”, where k is a running index; optimal results are bold and a dash (-) indicates that the time limit was exceeded

Instance	BCP without SRI					BCP with SRI						
	Root s	Root LB	Total s	Best LB	Nodes	Columns	Root s	Root LB	Total s	Best LB	Nodes	Columns
R10x10_0	0	48965	78	50135	677	38454	0	48965	65	50135	571	32620
R10x10_1	0	38474	0	38474	1	77	0	38474	0	38474	1	115
R10x10_2	0	30573	0	30573	1	21	0	30573	0	30573	1	21
R10x10_3	0	55755	3	56382	25	1537	0	55755	3	56382	23	1490
R10x10_4	0	32609	2	32943	39	2118	0	32609	4	32943	47	2754
R10x10_5	0	58577	3	59183	43	1839	0	58577	2	59183	25	1173
R10x10_6	0	55045	1	55235	11	773	0	55045	1	55235	7	440
R10x10_7	0	58378	25	59186	67	8272	0	58378	21	59186	61	7247
R10x10_8	0	55373	33	56471	147	13546	0	55373	32	56471	143	12658
R10x10_9	0	57262	27	58442	267	13028	0	57262	24	58442	241	11,727
R10x10_10	0	55,721	0	55,869	3	208	0	55,721	0	55,869	3	221
R10x10_11	0	39,199	0	39,199	1	178	0	39,199	0	39,199	1	146
R10x10_12	0	31035	0	31249	9	567	0	31035	0	31249	9	567
R10x10_13	0	64306	38	65270	249	13337	0	64306	41	65270	259	12991
R10x10_14	0	42992	2	43189	17	1009	0	42992	1	43189	11	781
R10x10_15	0	72431	237	74658	1485	90014	0	72431	219	74658	1303	82877
R10x10_16	0	30630	0	30630	1	157	0	30630	0	30630	1	112
R10x10_17	0	46870	25	47556	165	11615	0	46870	18	47556	101	7961
R10x10_18	0	35126	0	35126	1	70	0	35126	0	35126	1	57
R10x10_19	0	36132	0	36132	1	149	0	36132	0	36132	1	132
Avg	0.0	47272.4	23.7	47795.1	160.5	9848.5	0.0	47272.4	21.6	47795.1	140.5	8804.5

Table 7 continued

Instance	BCP without SRI					BCP with SRI						
	Root s	Root LB	Total s	Best LB	Nodes	Columns	Root s	Root LB	Total s	Best LB	Nodes	Columns
R20x20_20	0	86076	27	87023	113	11166	0	86076	15	87023	65	6210
R20x20_21	0	108584	74	109683	213	25220	0	108584	37	109683	117	12862
R20x20_22	1	86181	360	87556	943	146752	1	86181	153	87556	413	61366
R20x20_23	4	118112	597	119971	797	152259	3	118112	787	119971	1105	211759
R20x20_24	1	109827	157	110990	279	36451	1	109827	75	110990	145	19179
R20x20_25	0	75036	17	75857	93	7277	0	75036	36	75857	209	15995
R20x20_26	9	100677	-	102601	1126	428496	8	100677	-	102590	1038	408457
R20x20_27	0	93590	13	94222	43	3869	0	93590	35	94222	89	8104
R20x20_28	2	100072	16	100418	19	2507	1	100072	14	100418	21	2454
R20x20_29	0	72773	2	72972	7	715	0	72773	3	72972	7	883
R20x20_30	0	92109	14	92839	41	3787	0	92109	62	92839	231	18483
R20x20_31	0	92138	31	92628	59	7176	1	92138	19	92628	41	5174
R20x20_32	4	118887	2291	121646	2761	484876	3	118887	2235	121646	2561	452506
R20x20_33	11	134984	-	137205	937	332002	8	134984	-	137237	1185	424819
R20x20_34	6	123534	-	125267	922	297052	4	123534	-	125217	1271	392950
R20x20_35	3	95834	3109	97377	3207	352698	1	95834	1936	97377	2497	261436
R20x20_36	4	121080	-	123109	1115	339663	4	121080	-	123097	1461	448811
R20x20_37	0	84082	77	84786	93	10728	2	84082	70	84786	61	6987
R20x20_38	1	98190	87	99605	205	25068	1	98190	173	99605	365	47982
R20x20_39	0	97934	106	99005	353	37537	1	97934	119	99005	417	42253
Avg	2.3	100484.5	1069.0	101737.9	666.3	135265.0	2.0	100484.5	1008.5	101735.8	665.0	142433.5

Table 8 Average performance on the random R10x10 and R20x20 instances

	MIP			BCP					
	CPU s	gap BKS-LB	Optimal	Root s	Root LB	Total s	Best LB	gap BKS-LB	Optimal
Avg. 10×10	691.9	0.9%	90.0%	0.0	47272.4	21.6	47795.1	0.0%	100.0%
Avg. 20×20	3288.3	19.5%	10.0%	2.0	100484.5	1008.5	101735.8	0.7%	80.0%

not necessarily optimal, in case the one-hour time limit is reached.

Overall, the quality of the bounds is very similar because almost all instances can be solved to optimality within a one-hour time limit. However, there is a difference in the required CPU times. In only two small instances is BCP without SRI faster than with SRI. Using SRI, the CPU times as well as the number of explored nodes are significantly lower at the 90% confidence level. For the large instances, the effect is less clear: while the average CPU times are a little lower when using SRI, the effect is not significant. This is not surprising because the large instances, besides having more machines and jobs, also have an inherently longer planning horizon and thus a much greater number of columns and rows. Since we limit the number of SRI cuts to 64, they become a proverbial drop in the bucket. Note that we experimented with larger SRI sets but found that this tends to increase the solution effort of the pricing problem disproportionately and is generally not worthwhile.

We can hence conclude that SRI can help to speed up BCP for the MOSP-TD, although the effect is the greatest in small and medium size instances. We hence use the SRI in the following experiments in the given settings.

5.2.2 Algorithmic performance

As mentioned in Sect. 5.1, we tested BCP on three instance sets: newly generated random ones, those from Wirth and Emde (2018), and those derived from the classic open-shop scheduling instances from Brucker et al. (1997). For all instances, we list the CPU time in seconds for Gurobi solving the single model from Sect. 3.1 as well as BCP. Moreover, we calculate the gap between the best lower bound and the best known upper bound (columns labeled *gap BKS-LB*) as $(f^{\text{BKS}} - f^{\text{LB}})/f^{\text{LB}}$, where f^{BKS} denotes the objective value of the best known feasible solution and f^{LB} stands for the best lower bound of either Gurobi solving the single model or BCP. The time limit is 3600 CPU seconds per instance. For BCP, we also list the solution time (*Root s*) and LB (*Root LB*) before branching. The total runtime is given as *Total s* and the best found lower bound as *Best LB*.

Table 8 summarizes the average results for the new random instances; more detailed results are also in Table 7. Column *Optimal* gives the percentage of instances that were solved

to optimality within the one-hour time limit. BCP can solve all small random instances with $m = n = 10$ machines and jobs to optimality in about 20 s on average, whereas Gurobi solving the MIP takes a little over ten minutes on average for these instances. For the large instances, Gurobi solving the MIP fails to solve all but two of the instances to optimality within one hour, while BCP exhibits an optimality gap of less than 1% on average, solving most instances to optimality.

Table 9 shows the results for the instances from Wirth and Emde (2018). Although BCP struggles with solving the Wirth and Emde (2018) instances to optimality within the time limit, it finds good bounds almost across the board, the largest gap between upper bound (UB) and lower bound (LB) being 20.4% in instance WE_medium_2. On average, the gap is well below 4%. Gurobi working on the MIP model solves more instances to proven optimality but fails to produce tight bounds in those instances where it does not prove optimality. This does not only concern the weak lower bounds but also the best upper bounds: the gap between the best BCP and MIP upper bounds is as large as 15.4% (in instance WE_large_17). Overall, the gap between the BCP upper bound and the best known solution is well below 1%. Note that BCP already finds very tight bounds in the root node: the average gap BKS-LB gap is already down to 4.4% before branching.

Apart from our BCP, no exact algorithm (beyond a default solver) has yet been proposed for this problem. However, Wirth and Emde (2018) introduce several heuristics, the best of which is a greedy adaptive search procedure (GRASP). Unfortunately, all results given in Wirth and Emde (2018) are averaged over groups of three instances, hence—even though we test BCP on the exact same instance data—it is not possible to compare the results instance-by-instance. We can, however, compare the averages, which we do in Table 10. We list the gap of the best found solution by GRASP and BCP, respectively, to the best known lower bound LB^* , calculated as $\frac{UB-LB^*}{LB^*}$. Note that the best known lower bound may be different from the one listed in Table 9, because the column *Best LB* only lists the best bound found by BCP within the given time limit. For any given instance, the best bound may also have been found by solving the single model. For the gaps relating to GRASP, we take the values from Wirth and Emde (2018), who similarly calculate these gaps using the best of several lower bounds as the baseline. Furthermore, following Wirth and Emde (2018), we count an instance as

Table 9 Results on the WE_small, WE_medium, and WE_large instances from Wirth and Emde (2018); optimal results are bold and a dash (-) indicates that the time limit was exceeded

Instance	m	n	MIP		BCP				Nodes			
			BKS	CPU s	gap BKS-LB	Root s	Root LB	Total s		Best LB	Best UB	gap BKS-LB
WE_small_1	20	40	925	8	0.0%	31	923	66	925	925	0.0%	9
WE_small_2	20	40	400	2	0.0%	50	400	50	400	400	0.0%	1
WE_small_3	20	40	573	1	0.0%	43	573	43	573	573	0.0%	1
WE_small_4	20	40	415	2	0.0%	19	415	19	415	415	0.0%	1
WE_small_5	20	40	179	2	0.0%	101	179	101	179	179	0.0%	1
WE_small_6	20	40	1089	7	0.0%	13	998	-	999	1089	9.1%	127
WE_small_7	20	50	2730	33	0.0%	191	2709	-	2709	2730	0.8%	44
WE_small_8	20	50	1962	35	0.0%	168	1867	-	1876	1962	4.6%	97
WE_small_9	20	50	2810	622	0.0%	237	2580	-	2595	2875	8.3%	87
WE_small_10	20	50	1737	1722	0.0%	226	1607	-	1608	1763	8.0%	127
WE_small_11	20	50	3281	-	23.7%	157	3145	-	3186	3321	3.0%	31
WE_small_12	20	50	1292	-	85.9%	90	1253	-	1257	1367	2.8%	45
WE_small_13	20	60	4215	-	28.0%	54	4109	-	4150	4215	1.6%	59
WE_small_14	20	60	3910	-	61.4%	649	3725	-	3738	3910	4.6%	24
WE_small_15	20	60	7998	-	54.4%	202	7804	-	7829	7998	2.2%	69
WE_small_16	20	60	2854	-	28.4%	135	2712	-	2715	2863	5.2%	22
WE_small_17	20	60	4992	-	89.4%	458	4572	-	4579	4992	9.0%	10
WE_small_18	20	60	5719	-	168.1%	751	5118	-	5143	5719	11.2%	11
Avg.			2615.6	1735.2	30.0%	198.6	2482.4	2638.7	2492.9	2627.6	3.9%	42.6
WE_medium_1	30	60	345	5	0.0%	91	345	91	345	345	0.0%	1
WE_medium_2	30	60	425	8	0.0%	52	326	-	353	425	20.4%	284
WE_medium_3	30	60	3463	33	0.0%	168	3376	-	3384	3463	2.3%	142
WE_medium_4	30	60	3070	1615	0.0%	329	2935	-	2977	3070	3.1%	52
WE_medium_5	30	60	698	7	0.0%	145	668	-	668	698	4.5%	251
WE_medium_6	30	60	1129	22	0.0%	372	1100	-	1100	1138	2.7%	152
WE_medium_7	30	75	2824	541	0.0%	142	2757	-	2779	2824	1.6%	69
WE_medium_8	30	75	3523	1580	0.0%	203	3497	-	3522	3523	0.0%	91
WE_medium_9	30	75	3469	160	0.0%	287	3439	-	3451	3469	0.5%	59
WE_medium_10	30	75	2157	336	0.0%	972	2123	-	2123	2158	1.6%	26

Table 9 continued

Instance	m	n	MIP		BCP		Total s	Best LB	Best UB	gap BKS-LB	Nodes
			CPU s	gap BKS-LB	Root s	Root LB					
WE_medium_11	30	75	-	21.2%	900	2704	-	2714	2995	7.4%	8
WE_medium_12	30	75	15	0.0%	617	847	617	847	847	0.0%	1
WE_medium_13	30	90	-	6.9%	189	5864	-	5886	5991	1.8%	26
WE_medium_14	30	90	-	10.8%	629	4886	-	4894	4961	1.0%	23
WE_medium_15	30	90	-	18.2%	1078	4566	-	4573	4657	1.4%	10
WE_medium_16	30	90	-	47.9%	3533	4211	-	4223	4445	5.3%	1
WE_medium_17	30	90	-	125.2%	447	4838	-	4838	5363	6.2%	6
WE_medium_18	30	90	-	15.7%	586	2689	-	2691	2904	5.5%	18
Avg.			1640.4	13.7%	596.7	2842.5	3292.7	2853.5	2959.8	3.6%	67.8
WE_large_1	40	80	15	0.0%	196	2904	196	2904	2904	0.0%	1
WE_large_2	40	80	105	0.0%	416	2426	-	2442	2528	3.5%	66
WE_large_3	40	80	12	0.0%	278	860	-	877	936	6.7%	113
WE_large_4	40	80	15	0.0%	3	60	3	60	60	0.0%	1
WE_large_5	40	80	1618	0.0%	588	4420	-	4434	4685	4.5%	14
WE_large_6	40	80	15	0.0%	194	523	-	531	563	6.0%	405
WE_large_7	40	100	-	8.5%	431	4605	-	4609	4746	3.0%	30
WE_large_8	40	100	641	0.0%	1326	3825	1326	3825	3825	0.0%	1
WE_large_9	40	100	216	0.0%	568	3913	-	3930	3988	1.5%	28
WE_large_10	40	100	-	13.6%	1319	4277	-	4277	4321	0.8%	9
WE_large_11	40	100	185	0.0%	676	1339	-	1339	1467	5.0%	45
WE_large_12	40	100	465	0.0%	1817	511	-	511	600	4.9%	2
WE_large_13	40	120	-	82.6%	634	7648	-	7649	7754	1.4%	19
WE_large_14	40	120	-	32.9%	490	10655	-	10674	10842	1.4%	13
WE_large_15	40	120	-	70.4%	1304	8755	-	8771	8890	1.4%	3
WE_large_16	40	120	-	125.5%	441	5023	-	5035	5340	6.1%	3
WE_large_17	40	120	-	210.2%	494	6590	-	6598	7060	7.0%	4
WE_large_18	40	120	-	81.1%	4505	2521	4505	3008	3008	0.0%	1
Avg.			1783.3	34.7%	871.1	3936.0	3262.5	3970.6	4084.3	3.0%	42.1

Table 10 Comparison of BCP with the best heuristic from Wirth and Emde (2018) (GRASP) on the *WE* instances

	GRASP			BCP		
	CPU s	# optimal	UB-to-LB gap	CPU s	# optimal	UB-to-LB gap
WE_small ($m = 20$)	6.8	9	3.3%	2638.7	8	2.8%
WE_medium ($m = 30$)	12.2	9	4.6%	3292.7	10	2.2%
WE_large ($m = 40$)	27.8	6	10.7%	3262.5	8	2.2%

Table 11 Results on the modified j*-per* instances from Brucker et al. (1997); missing values denote that the best found LB is 0; optimal results are bold and a dash (-) indicates that the time limit was exceeded

Instance	m	n	BKS	MIP		BCP						
				CPU s	gap BKS-LB	Root s	Root LB	Total s	Best LB	Best UB	gap BKS-LB	Nodes
j3-per0-1	3	3	2	7	0.0%	0	2	0	2	2	0.0%	1
j3-per0-2	3	3	3	0	0.0%	0	3	0	3	3	0.0%	1
j3-per10-0	3	3	2	2	0.0%	0	2	0	2	2	0.0%	1
j3-per10-1	3	3	1	0	0.0%	0	1	0	1	1	0.0%	1
j3-per10-2	3	3	1	0	0.0%	0	1	0	1	1	0.0%	1
j3-per20-0	3	3	3	0	0.0%	0	3	0	3	3	0.0%	1
j3-per20-1	3	3	3	12	0.0%	0	3	0	3	3	0.0%	1
j3-per20-2	3	3	2	0	0.0%	0	2	0	2	2	0.0%	1
j4-per0-0	4	4	0	4	0.0%	0	0	0	0	0	0.0%	1
j4-per0-1	4	4	1	6	0.0%	0	1	0	1	1	0.0%	1
j4-per0-2	4	4	2	4	0.0%	0	2	0	2	2	0.0%	1
j4-per10-0	4	4	2	3	0.0%	0	2	0	2	2	0.0%	1
j4-per10-1	4	4	1	4	0.0%	0	1	0	1	1	0.0%	1
j4-per10-2	4	4	1	3	0.0%	0	1	0	1	1	0.0%	1
j4-per20-0	4	4	2	3	0.0%	0	2	0	2	2	0.0%	1
j4-per20-1	4	4	1	4	0.0%	0	1	0	1	1	0.0%	1
j4-per20-2	4	4	2	100	0.0%	0	2	0	2	2	0.0%	1
j5-per0-0	5	5	4	16	0.0%	0	4	0	4	4	0.0%	1
j5-per0-1	5	5	1	10	0.0%	0	1	0	1	1	0.0%	1
j5-per0-2	5	5	2	16	0.0%	0	2	0	2	2	0.0%	1
j5-per10-0	5	5	3	-	50.0%	0	3	0	3	3	0.0%	1
j5-per10-1	5	5	2	10	0.0%	0	2	0	2	2	0.0%	1
j5-per10-2	5	5	2	12	0.0%	0	2	0	2	2	0.0%	1
j5-per20-0	5	5	2	9	0.0%	0	2	0	2	2	0.0%	1
j5-per20-1	5	5	3	11	0.0%	0	3	0	3	3	0.0%	1
j5-per20-2	5	5	4	-	100.0%	0	4	0	4	4	0.0%	1
j6-per0-0	6	6	3	-	50.0%	8	3	-	3	4	0.0%	350
j6-per0-1	6	6	4	60	0.0%	0	4	0	4	4	0.0%	1
j6-per0-2	6	6	4	423	0.0%	0	4	0	4	4	0.0%	1
j6-per10-0	6	6	4	-	100.0%	0	4	0	4	4	0.0%	1
j6-per10-1	6	6	1	25	0.0%	8	1	-	1	2	0.0%	222
j6-per10-2	6	6	2	-		0	2	0	2	2	0.0%	1
j6-per20-0	6	6	1	20	0.0%	0	1	0	1	1	0.0%	1
j6-per20-1	6	6	2	21	0.0%	0	2	0	2	2	0.0%	1
j6-per20-2	6	6	1	16	0.0%	0	1	0	1	1	0.0%	1

Table 11 continued

Instance	m	n	BKS	MIP		BCP						
				CPU s	gap BKS-LB	Root s	Root LB	Total s	Best LB	Best UB	gap BKS-LB	Nodes
j7-per0-0	7	7	6	-	200.0%	0	6	0	6	6	0.0%	1
j7-per0-1	7	7	3	-	0.0%	0	3	0	3	3	0.0%	1
j7-per0-2	7	7	5	-		0	5	0	5	5	0.0%	1
j7-per10-0	7	7	3	53	0.0%	0	3	0	3	3	0.0%	1
j7-per10-1	7	7	3	36	0.0%	0	3	0	3	3	0.0%	1
j7-per10-2	7	7	3	55	0.0%	0	3	0	3	3	0.0%	1
j7-per20-0	7	7	4	113	0.0%	0	4	0	4	4	0.0%	1
j7-per20-1	7	7	4	-	100.0%	0	4	0	4	4	0.0%	1
j7-per20-2	7	7	4	88	0.0%	0	4	0	4	4	0.0%	1
j8-per0-1	8	8	3	-	0.0%	0	3	0	3	3	0.0%	1
j8-per0-2	8	8	6	-	200.0%	1	6	1	6	6	0.0%	1
j8-per10-0	8	8	5	-	25.0%	0	5	0	5	5	0.0%	1
j8-per10-1	8	8	3	98	0.0%	0	3	0	3	3	0.0%	1
j8-per10-2	8	8	2	-		64	2	-	2	3	0.0%	105
j8-per20-0	8	8	3	840	0.0%	26	3	-	3	4	0.0%	282
j8-per20-1	8	8	4	-	100.0%	0	4	0	4	4	0.0%	1
j8-per20-2	8	8	1	58	0.0%	58	1	-	1	2	0.0%	118
Avg.			2.6	1010.6	18.9%	3.2	2.6	348.9	2.6	2.7	0.0%	21.6

solved to optimality if the best found upper bound returned within the time limit is equal to the best lower bound LB^* , even if the lower bound was not found by the same solution method. The optimality criterion we use in the rest of the study (including Table 9) is hence stricter than the one used in Table 10, because we usually only count an instance as solved to optimality if BCP itself manages to close the gap within the time limit. Wirth and Emde (2018) use a PC equipped with an Intel i7-6700K CPU (4.4 GHz) and 64 GB of RAM to measure their CPU times, which we reproduce in Table 10.

The results show that GRASP is significantly faster than BCP, which is not surprising, given that GRASP is a heuristic by design. BCP finds the optimal upper bound slightly more often than GRASP (26 vs. 24 times). More strikingly, BCP produces much smaller gaps overall: the average gap over all *WE* instances is 2.4%, whereas GRASP produces a 6.2% average gap. Moreover, unlike for GRASP, the gap for BCP does not increase when solving larger instances. This indicates, somewhat surprisingly, that BCP scales better than the heuristic procedure, and can deliver decent upper bounds even for large problems and even in cases where it does not find the optimal solution within the time limit.

Our final benchmark set is the data from Brucker et al. (1997), enriched with transfer times, time windows and a minimum number of tardy jobs objective. Table 11 shows that BCP could find the maximal lower bound in every single instance. In the vast majority of instances, it could close

the gap completely; in only five instances is there a difference of one tardy job between the best lower and upper bounds. Apart from those five instances, the optimal solution is found already in the root node. Gurobi solving the single model struggles substantially more, failing to close the gap in 14 instances, yielding an average gap of almost 19% and far longer CPU times. Overall, this suggests that the $\sum U_j$ objective of the *j*-per** instances poses less of a challenge for BCP than the $\sum C_j^2$ objective of the *R* and the $\sum w_j T_j$ objective of the *WE* instances. One reason may be that the $\sum U_j$ objective tends to lead to a search space with large plateaus, i.e., there are many solutions with the same objective value because $\sum U_j$ only counts the number of late jobs but not the amount of lateness unlike, e.g., $\sum w_j T_j$. The MIP (and many heuristics based on neighborhood search) struggle with this, but it seems to aid BCP.

6 Conclusion

We investigate a general open-shop scheduling problem with machine-pair-dependent transfer times, release and due dates, and any arbitrary regular minsum objective. We develop a branch-cut-and-price method, which is based on a time-indexed formulation of the problem. We adapt subset-row cuts, originally from the VRP literature, to the open-shop, and demonstrate in a computational study on instances from the literature as well as newly generated ones

that our BCP is effective in finding tight bounds even for the largest problems. We also demonstrate that the approach is flexible and well-performing across a range of different objectives.

Our results show that BCP is a suitable method for solving even complex open-shop scheduling problems, despite the very large solution space. Moreover, our tests indicate that adapting non-robust cuts to scheduling problems—which has only rarely been tried before (Bulhoes et al., 2020)—may be a viable strategy for boosting the performance of BCP for scheduling problems in general (not just the open-shop). Our adaptation of the famous SRI cuts yielded significant runtime improvements at least in some instances.

Future research may even further generalize our BCP, e.g., by also considering job sequence-dependent setup times [e.g., Mejía and Yuraszeck (2020)]. Due to the flexible nature of our method, this should be relatively straightforward. Moreover, further robust and non-robust cuts may be applicable to open-shop scheduling and could be integrated into our BCP framework. It may also be possible to speed up the solution of the pricing problem by employing, e.g., *ng*-route relaxations (Baldacci et al., 2011b). Beyond BCP, constraint-programming solvers may also serve as a good starting point and/or benchmark. Finally, there is generally speaking a dearth of powerful solution methods for complex (i.e., realistic) open-shop scheduling problems that go beyond the classic makespan objective. For instance, investigating non-regular objective functions [e.g., earliness-tardiness, Hall and Posner (1991)] or general minmax objectives may be a fruitful avenue for further research.

Funding Open Access funding enabled and organized by Projekt DEAL.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by/4.0/>.

References

- Ahmadian, M. M., Khatami, M., Salehipour, A., & Cheng, T. (2021). Four decades of research on the open-shop scheduling problem to minimize the makespan. *European Journal of Operational Research*, 295(2), 399–426.
- Anand, E., & Panneerselvam, R. (2015). Literature review of open shop scheduling problems. *Intelligent Information Management*, 7(1), 33.
- Andresen, M., Bräsel, H., Mörig, M., Tusch, J., Werner, F., & Willenius, P. (2008). Simulated annealing and genetic algorithms for minimizing mean flow time in an open shop. *Mathematical and Computer Modelling*, 48(7–8), 1279–1293.
- Bai, D., Liang, J., Liu, B., Tang, M., & Zhang, Z.-H. (2017). Permutation flow shop scheduling problem to minimize nonlinear objective function with release dates. *Computers & Industrial Engineering*, 112, 336–347.
- Bai, D., Zhang, Z., Zhang, Q., & Tang, M. (2017). Open shop scheduling problem to minimize total weighted completion time. *Engineering Optimization*, 49(1), 98–112.
- Bai, D., Zhang, Z.-H., & Zhang, Q. (2016). Flexible open shop scheduling problem to minimize makespan. *Computers & Operations Research*, 67, 207–215.
- Baldacci, R., Bartolini, E., & Mingozzi, A. (2011). An exact algorithm for the pickup and delivery problem with time windows. *Operations Research*, 59(2), 414–426.
- Baldacci, R., Mingozzi, A., & Roberti, R. (2011). New route relaxation and pricing strategies for the vehicle routing problem. *Operations Research*, 59(5), 1269–1283.
- Bigras, L.-P., Gamache, M., & Savard, G. (2008). Time-indexed formulations and the total weighted tardiness problem. *INFORMS Journal on Computing*, 20(1), 133–142.
- Błażewicz, J., Pesch, E., Sterna, M., & Werner, F. (2004). Open shop scheduling problems with late work criteria. *Discrete Applied Mathematics*, 134(1–3), 1–24.
- Bräsel, H., Herms, A., Mörig, M., Tautenhahn, T., Tusch, J., & Werner, F. (2008). Heuristic constructive algorithms for open shop scheduling to minimize mean flow time. *European Journal of Operational Research*, 189(3), 856–870.
- Brucker, P., Hurink, J., Jurisch, B., & Wöstmann, B. (1997). A branch & bound algorithm for the open-shop problem. *Discrete Applied Mathematics*, 76(1–3), 43–59.
- Bulhoes, T., Sadykov, R., Subramanian, A., & Uchoa, E. (2020). On the exact solution of a large class of parallel machine scheduling problems. *Journal of Scheduling*, 23(4), 411–429.
- Cheng, T. E., & Shakhlevich, N. V. (2005). Minimizing non-decreasing separable objective functions for the unit-time open shop scheduling problem. *European Journal of Operational Research*, 165(2), 444–456.
- Codato, G., & Fischetti, M. (2006). Combinatorial Benders' cuts for mixed-integer linear programming. *Operations Research*, 54(4), 756–766.
- Dell'Amico, M., & Martello, S. (1996). Open shop, satellite communication and a theorem by Egerváry (1931). *Operations Research Letters*, 18(5), 207–211.
- Desaulniers, G., Lessard, F., & Hadjar, A. (2008). Tabu search, partial elementarity, and generalized k-path inequalities for the vehicle routing problem with time windows. *Transportation Science*, 42(3), 387–404.
- Dorndorf, U., Pesch, E., & Phan-Huy, T. (2001). Solving the open shop scheduling problem. *Journal of Scheduling*, 4(3), 157–174.
- Fei, H., Meskens, N., Combes, C., & Chu, C. (2009). The endoscopy scheduling problem: A case study with two specialised operating rooms. *International Journal of Production Economics*, 120(2), 452–462.
- Gonzalez, T., & Sahni, S. (1976). Open shop scheduling to minimize finish time. *Journal of the ACM (JACM)*, 23(4), 665–679.
- Graham, R. L., Lawler, E. L., Lenstra, J. K., & Kan, A. R. (1979). Optimization and approximation in deterministic sequencing and scheduling: A survey. *Annals of Discrete Mathematics*, 5, 287–326.

- Gschwind, T., Irnich, S., Tilk, C., & Emde, S. (2020). Branch-cut-and-price for scheduling deliveries with time windows in a direct shipping network. *Journal of Scheduling*, 23(3), 363–377.
- Guéret, C., Jussien, N., & Prins, C. (2000). Using intelligent backtracking to improve branch-and-bound methods: An application to open-shop problems. *European Journal of Operational Research*, 127(2), 344–354.
- Hall, N. G., & Posner, M. E. (1991). Earliness-tardiness scheduling problems, I: Weighted deviation of completion times about a common due date. *Operations Research*, 39(5), 836–846.
- Ho, J. C., & Chang, Y.-L. (1995). Minimizing the number of tardy jobs for M parallel machines. *European Journal of Operational Research*, 84(2), 343–355.
- Irnich, S. and Desaulniers, G. (2005). Shortest path problems with resource constraints. In *Column Generation*, (pp. 33–65). Springer.
- Jepsen, M., Petersen, B., Spoorendonk, S., & Pisinger, D. (2008). Subset-row inequalities applied to the vehicle-routing problem with time windows. *Operations Research*, 56(2), 497–511.
- Kowalczyk, D., & Leus, R. (2018). A branch-and-price algorithm for parallel machine scheduling using ZDDS and generic branching. *INFORMS Journal on Computing*, 30(4), 768–782.
- Laborie, P. (2005). Complete mcs-based search: Application to resource constrained project scheduling. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence, IJCAI'05*, (pp. 181–186), San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.
- Lancia, G., Rinaldi, F., & Serafini, P. (2011). A time-indexed LP-based approach for min-sum job-shop problems. *Annals of Operations Research*, 186(1), 175–198.
- Liaw, C.-F. (2003). An efficient tabu search approach for the two-machine preemptive open shop scheduling problem. *Computers & Operations Research*, 30(14), 2081–2095.
- Liu, C., & Bulfin, R. (1988). Scheduling open shops with unit execution times to minimize functions of due dates. *Operations Research*, 36(4), 553–559.
- Malapert, A., Cambazard, H., Guéret, C., Jussien, N., Langevin, A., & Rousseau, L.-M. (2012). An optimal constraint programming approach to the open-shop problem. *INFORMS Journal on Computing*, 24(2), 228–244.
- Mejía, G., & Yuraszeck, F. (2020). A self-tuning variable neighborhood search algorithm and an effective decoding scheme for open shop scheduling problems with travel/setup times. *European Journal of Operational Research*, 285(2), 484–496.
- Naderi, B., Fatemi Ghomi, S., Aminnayeri, M., & Zandieh, M. (2011). A study on open shop scheduling to minimise total tardiness. *International Journal of Production Research*, 49(15), 4657–4678.
- Naderi, B., Ruiz, R., & Roshanaei, V. (2023). Mixed-integer programming vs. constraint programming for shop scheduling problems: New results and outlook. *INFORMS Journal on Computing*, 35(4), 817–843.
- Ng, C. T., Cheng, T. E., & Yuan, J. (2003). Concurrent open shop scheduling to minimize the weighted number of tardy jobs. *Journal of Scheduling*, 6(4), 405–412.
- Oliveira, D., & Pessoa, A. (2020). An improved branch-cut-and-price algorithm for parallel machine scheduling problems. *INFORMS Journal on Computing*, 32(1), 90–100.
- Ozolins, A. (2021). Dynamic programming approach for solving the open shop problem. *Central European Journal of Operations Research*, 29(1), 291–306.
- Pessoa, A., Uchoa, E., De Aragão, M. P., & Rodrigues, R. (2010). Exact algorithm over an arc-time-indexed formulation for parallel machine scheduling problems. *Mathematical Programming Computation*, 2(3), 259–290.
- Pinedo, M. (2016). *Scheduling: Theory, algorithms, and systems*. Springer International Publishing.
- Raut, S., Gupta, J. N., & Swami, S. (2008). Single machine scheduling with time deteriorating job values. *Journal of the Operational Research Society*, 59(1), 105–118.
- Stewart, I. (1996). Tales of a neglected number. *Scientific American*, 274(6), 102–103.
- Strusevich, V. A. (2022). Complexity and approximation of open shop scheduling to minimize the makespan: A review of models and approaches. *Computers & Operations Research*, 144, Article 105732.
- Tamura, N., Taga, A., Kitagawa, S., & Banbara, M. (2009). Compiling finite linear CSP into sat. *Constraints*, 14(2), 254–272.
- Tang, L., & Bai, D. (2010). A new heuristic for open shop total completion time problem. *Applied Mathematical Modelling*, 34(3), 735–743.
- Van den Akker, J., Hurkens, C. A., & Savelsbergh, M. W. (2000). Time-indexed formulations for machine scheduling problems: Column generation. *INFORMS Journal on Computing*, 12(2), 111–124.
- van Den Akker, J. M., Hoogeveen, J. A., & van de Velde, S. L. (1999). Parallel machine scheduling by column generation. *Operations Research*, 47(6), 862–872.
- Wang, T., Baldacci, R., Lim, A., & Hu, Q. (2018). A branch-and-price algorithm for scheduling of deteriorating jobs and flexible periodic maintenance on a single machine. *European Journal of Operational Research*, 271(3), 826–838.
- Wirth, M., & Emde, S. (2018). Scheduling trucks on factory premises. *Computers & Industrial Engineering*, 126, 175–186.
- Xiao, L., Zhen, L., Laporte, G., Baldacci, R., & Wang, C. (2022). A column generation-based heuristic for a rehabilitation patient scheduling and routing problem. *Computers & Operations Research*, 148, 105970.
- Zhuang, Z., Huang, Z., Sun, Y., & Qin, W. (2021). Optimization for cooperative task planning of heterogeneous multi-robot systems in an order picking warehouse. *Engineering Optimization*, 53(10), 1715–1732.

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.