J Sched (2017) 20:147–164
DOI 10.1007/s10951-016-0477-x

Improved approaches to the exact solution of the machine covering problem

Rico Walter[1] · **Martin Wirth**[2] · **Alexander Lawrinenko**[3]

Published online: 22 April 2016

Abstract For the basic problem of scheduling a set of n independent jobs on a set of m identical parallel machines with the objective of maximizing the minimum machine completion time—also referred to as machine covering—we propose a new exact branch-and-bound algorithm. Its most distinctive components are a different symmetry-breaking solution representation, enhanced lower and upper bounds, and effective novel dominance criteria derived from structural patterns of optimal schedules. Results of a comprehensive computational study conducted on benchmark instances attest to the effectiveness of our approach, particularly for small ratios of n to m.

Keywords Identical parallel machines · Machine covering · Dominance criteria · Branch-and-bound

✉ Rico Walter
rico.walter@gmail.com; rico.walter@itwm.fraunhofer.de

Martin Wirth
wirth@bwl.tu-darmstadt.de

Alexander Lawrinenko
alexander.lawrinenko@uni-jena.de

[1] Department of Optimization, Fraunhofer Institute for Industrial Mathematics ITWM, Fraunhofer-Platz 1, 67663 Kaiserslautern, Germany

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

[3] Lehrstuhl für ABWL/Management Science, Friedrich-Schiller-Universität Jena, Carl-Zeiß-Straße 3, 07743 Jena, Germany

1 Introduction

One of the most fundamental and well-studied $\mathcal{NP}$-hard problems in the field of machine scheduling is the makespan minimization on identical parallel machines where a set of n independent jobs $\mathcal{J} = \{J_1, \ldots, J_n\}$ with positive processing times $t_1, \ldots, t_n$ has to be assigned to m identical parallel machines $\mathcal{M} = \{M_1, \ldots, M_m\}$ in order to minimize the latest machine completion time. A related and in some sense dual but by far not as well-studied $\mathcal{NP}$-hard problem is obtained when the objective is changed from minimizing the makespan $C_{\max}$ to maximizing the minimum completion time $C_{\min} = \min\{C_1, \ldots, C_m\}$—without introducing idle times—where C_i is the sum of processing times of all jobs assigned to M_i. While the former problem (which is denoted by $P||C_{\max}$ using the three-field notation of Graham et al. 1979) is a kind of packing problem, the latter problem (abbreviated as $P||C_{\min}$), which is the subject of this paper, belongs to the class of covering problems and therefore is also referred to as the machine covering problem. It has been first described by Friesen and Deuermeyer (1981) in the context of spare parts assignments to machines that undergo repeated repair. As another application of $P||C_{\min}$, Haouari and Jemmali (2008) mentioned (fair) regional allocations of investments.

Although both problems basically intend to balance the workload among a given set of resources, $P||C_{\min}$ has received less attention than its much more prominent counterpart $P||C_{\max}$. To the best of our knowledge, the rather scarce literature on $P||C_{\min}$ is limited to a few studies on approximation algorithms and their worst-case ratios and up to now only one exact solution procedure has been proposed. For the well-known longest processing time rule (LPT), Deuermeyer et al. (1982) showed the minimum completion time of the LPT-schedule $C_{\min}^{\text{LPT}}$ to never be less than 3/4 times the optimal minimum completion time $C^*_{\min}$. Ten years later, Csirik et al. (1992) tightened this performance bound by proving that $C^{\mathrm{LPT}}_{\min}/C^*_{\min} \geq (3m-1)/(4m-2)$ is fulfilled for any fixed m. In this context, Walter (2013) recently examined the performance relationship between the LPT-rule and a restricted version of it—known as RLPT—and he proved that $C^{\mathrm{LPT}}_{\min} \geq C^{\mathrm{RLPT}}_{\min}$. Further publications are concerned with a polynomial-time approximation scheme (PTAS) (cf. Woeginger 1997) and on-line as well as semi-on-line versions of $P||C_{\min}$ (cf., e.g., Azar and Epstein 1998; He and Tan 2002; Luo and Sun 2005; Ebenlendr et al. 2006; Cai 2007; Tan and Wu 2007; Epstein et al. 2011). The sole publication devoted to exact solution procedures is due to Haouari and Jemmali (2008). The main features of their branch-and-bound algorithm are tight lower and upper bounds and a symmetry-breaking solution structure. However, except for small-sized instances, computational results revealed that their algorithm fails to (quickly) solve instances where the ratio of n to m ranges between two and about three.

To overcome this drawback, we approach the machine covering problem from a similar perspective as done by Walter and Lawrinenko (2014) for the dual problem $P||C_{\max}$. In their very recent contribution, the authors present structural properties of (potentially) makespan-optimal schedules. These properties are then transformed into problem-specific dominance criteria and implemented in a tailored branch-and-bound algorithm that performs very well on instances with small n/m-values.

Motivated by Walter and Lawrinenko's results, in this paper we propose a tailored branch-and-bound algorithm for problem $P||C_{\min}$. Our contribution differs substantially from the contribution by Walter and Lawrinenko (2014) in the following respects: (i) we show that their central result on makespan-optimal schedules also applies to problem $P||C_{\min}$, (ii) we extend their basic dominance criterion by several novel $P||C_{\min}$-specific dominance criteria derived from properties of optimal $P||C_{\min}$-schedules, and (iii) we develop new upper bounds on $C^*_{\min}$. Some of these bounds are derived from the solution structure, while others exploit the coherence between $P||C_{\min}$ and the bin covering problem (BCP). The latter problem consists in packing a set of indivisible items into as many bins as possible so that the total weight of each bin equals at least C. To the best of our knowledge, this coherence has not been mentioned before in the literature.

The paper is organized as follows. In the technical part we describe properties of $C_{\min}$-optimal schedules (cf. Sect. 2) and translate them into novel $P||C_{\min}$-specific dominance criteria (cf. Sect. 3). In the algorithmic part, we provide a concise description of the proposed branch-and-bound algorithm (Sect. 4) and we evaluate its performance on different sets of benchmark instances (cf. Sect. 5). The paper concludes with a short summary and some interesting ideas for future research in Sect. 6.

For economy of notation, we usually identify both machines and jobs by their index. Moreover, w.l.o.g, we assume the jobs to be indexed so that $t_1 \geq \cdots \geq t_n$. In addition, to avoid trivial instances we presuppose $n > m \geq 2$.

2 Theoretical background

In this section, we derive structural properties of $C_{\min}$-optimal schedules, which will be transformed later into problem-specific dominance criteria.

2.1 Solution representation and illustration

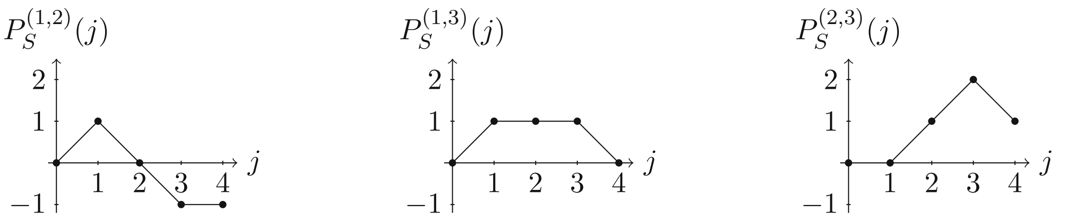
Throughout this paper, we assume schedules to be *non-permuted*. According to Walter and Lawrinenko (2014), a schedule $S \in \{1, 2, \ldots, m\}^n$—where $S(j) = i$ means that job j is assigned to machine i—is said to be non-permuted if S fulfills the following two conditions:

(i) $S(1) = 1$ and
(ii) $S(j) \in \left\{1, \ldots, \min\{m, 1 + \max_{1\leq k\leq j-1} S(k)\}\right\}$ for all $j = 2, \ldots, n$.

Note that due to this representation, symmetric reflections obtained by a simple renumbering of the machines are avoided.

Furthermore, instead of using Gantt charts we adopt the illustration of schedules as sets of *paths* which has also been introduced by Walter and Lawrinenko (2014). More precisely, a schedule S is represented by $\binom{m}{2}$ paths $P_S^{(i_1,i_2)}$—one for each pair (i_1, i_2) of machines $(1 \leq i_1 < i_2 \leq m)$. Each path is a string of length $n+1$ where the j-th entry $(j = 1, \ldots, n)$ represents the difference between the number of jobs assigned to i_1 and i_2 in S after the assignment of the j *longest* jobs. Additionally, to allow for initially empty machines, we set $P_S^{(i_1,i_2)}(0) = 0$ for all pairs. In a graphical illustration, the entries of a path are linearly connected (see Example 2.1).

Example 2.1 Let $n = 4$, $m = 3$, and consider the non-permuted schedule $S = (1, 2, 2, 3)$. The corresponding paths $P_S^{(1,2)} = (0, 1, 0, -1, -1)$, $P_S^{(1,3)} = (0, 1, 1, 1, 0)$, and $P_S^{(2,3)} = (0, 0, 1, 2, 1)$ are illustrated below.



2.2 Potential optimality

The concept of potential optimality has recently been proposed by Walter and Lawrinenko (2014) and originates from the question: "Are there certain general patterns in the structure of schedules that cannot lead to optimal solutions?" Admittedly, at first glance, this approach appears to be a bit unusual as actually we intend to derive properties of optimal solutions. However, by identifying properties of solutions that can never become (uniquely) optimal, we also implicitly gain insights into the structure of solutions that have the potential to become (uniquely) optimal—which are therefore called *potentially* (unique) optimal.

At this point, we want to mention that the concept of potential optimality is to some extent related with the concept of inverse optimization (for a review, see Ahuja and Orlin 2001, as well as Heuberger 2004) where unknown exact values of some adjustable parameters, e.g., processing times, should be determined within given boundaries in such a way that a pre-specified solution becomes optimal and the deviation between the determined and the given values of the parameters is minimal. Although inverse optimization has attracted many researchers in different areas of combinatorial optimization during the last two decades, applications to scheduling problems (e.g., see Koulamas 2005, as well as Brucker and Shakhlevich 2009, 2011) are still rather rare. Our approach—which differs slightly from the basic idea of inverse optimization in that we intend to identify a preferably large set of solutions for which we cannot select processing times so that any of these solutions becomes uniquely $P||C_{\min}$-optimal—constitutes another contribution in this field.

As will be seen next (cf. Theorem 2.2), solutions contained in the set $\mathcal{S}$ play a crucial role in the context of potentially unique optimal $P||C_{\min}$-schedules. The set $\mathcal{S}$ is formally defined as

$$\mathcal{S} = \Big\{ S \,:\, \text{for each pair } (i_1, i_2) \text{ there exists either a } j \in \{3, \dots, n\} \text{ so that } P_S^{(i_1,i_2)}(j) < 0, \text{ or } 0 < j_1 \le j_2 < n \text{ so that } P_S^{(i_1,i_2)}(j) = 1 \text{ for } j = j_1, \dots, j_2 \text{ and } P_S^{(i_1,i_2)}(j) = 0 \text{ else} \Big\}$$

and contains all schedules where each machine processes at least one job and each path of a pair of machines that processes more than two jobs in total has at least one negative entry. We say that schedules in $\mathcal{S}$ fulfill the *path-conditions* or, equivalently, all $\binom{m}{2}$ paths fulfill their respective path-condition.

Revisiting the schedule $S = (1, 2, 2, 3)$ introduced in Example 2.1, we readily see that S is no element of $\mathcal{S}$, although no machine remains empty and the paths $P_S^{(1,2)}$ and $P_S^{(1,3)}$ fulfill their path-condition. However, the path $P_S^{(2,3)}$ does not fulfill its path-condition as it has no negative entry, although the two machines process more than two jobs in total.

Our main theorem on potential optimality reads as follows.

Theorem 2.2 *Let S be a schedule which is no element of $\mathcal{S}$. Then, S is not a potentially unique $C_{\min}$-optimal solution.*

Proof Consider an arbitrary schedule S which is no element of $\mathcal{S}$. Then, there exists a pair of machines—say (i_1, i_2)—whose corresponding path does not fulfill the path-condition. These two machines and the respective set of jobs currently assigned to them constitute a solution to the machine covering problem on two identical parallel machines. Since machine covering and makespan minimization on two identical parallel machines are equivalent, we can apply a result by Walter and Lawrinenko (2014). They proved that with their so-called *two-machine path-modification*, a two-machine schedule which does not fulfill its path-condition can be turned into a schedule whose respective path does fulfill its path-condition without increasing the maximum completion time of i_1 and i_2. Note that the latter is equivalent to the fact that the minimum completion time of i_1 and i_2 does not decrease during the modification of the given schedule.

As the two-machine path-modification does not affect any jobs on the remaining $m - 2$ machines, the minimum completion time of the transformed schedule cannot be smaller than the minimum completion time of S. Hence, S cannot be potentially unique $C_{\min}$-optimal. □

Summarizing the result of Theorem 2.2, we know that every instance of the problem $P||C_{\min}$ has an optimal solution where all corresponding $\binom{m}{2}$ paths fulfill their path-condition.

In what follows, we will make use of the previous result and deduce several $P||C_{\min}$-specific dominance criteria—

subsumed under the term *path-related* dominance criteria—which will later on prove to be effective in guiding the search of a tailored branch-and-bound algorithm toward schedules which are elements of $\mathcal{S}$.

3 Dominance criteria based on potential optimality

For a better understanding, we start with a brief repetition of the basic dominance criterion developed by Walter and Lawrinenko (2014). All other dominance criteria presented within this section are novel and $P||C_{\min}$-specific.

3.1 The basic criterion

Given a partial solution $\tilde{S}$ where the k longest jobs have already been assigned, the basic criterion is readily obtained from the characterization of potentially unique optimal schedules (cf. Theorem 2.2). Recalling that each pair (i_1, i_2) of machines $1 \le i_1 < i_2 \le m$ has to fulfill its path-condition, for each $i_2 \in \{m, m-1, \dots, 2\}$ we simply have to count the minimum number of jobs $v_{i_2}^k$ that still have to be assigned to i_2 so that all paths fulfill their path-condition. According to Walter and Lawrinenko (2014), $v_{i_2}^k$ can be computed as

$$v_{i_2}^k = \max_{\substack{i_1=1,\dots,i_2-1 \\ PF_{\tilde{S}}^{(i_1,i_2)}(k)=0}} \left\{P_{\tilde{S}}^{(i_1,i_2)}(k)\right\} + \gamma_{i_2}^k, \tag{1}$$

where $PF_{\tilde{S}}^{(i_1,i_2)}(k) = 0$ indicates that the pair (i_1, i_2) does not currently fulfill its path-condition and $\gamma_{i_2}^k$ is a correction term that is either 0 (iff all machines $i_1 < i_2$ process at most one of the first k jobs) or 1 (iff at least one machine $i_1 < i_2$ processes more than one job), respectively. Clearly, $v_{i_2}^k$ is set to 0 if all pairs (i_1, i_2) currently fulfill their path-condition. Then, the basic criterion reads as follows.

Criterion 3.1 *If*

$$\sum_{i=1}^m v_i^k > n - k \tag{2}$$

for some $k < n$, then the current partial solution can be fathomed.

With regard to the next section, we define $v_1^k = 0$ for all $k > 1$.

3.2 Further improvements

Up to now, the basic dominance Criterion 3.1 does not consider any machine completion times and therefore offers the

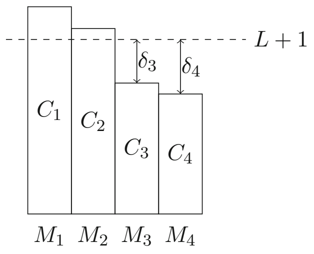


Fig. 1 Illustration of δ_i

potential for some improvements. Clearly, in a new incumbent solution, each machine completion time has to be at least as large as $L+1$ where L is the best known minimum completion time so far. Based on this information, at first we will show how some of the demands v_i^k can be tightened and secondly we will check whether the current partial solution admits the possibility to become the new incumbent.

Recalling that we consider a partial solution where the k longest jobs have already been assigned, we let LOW^k denote the set of machines with current completion time C_i^k at most L, i.e., $\text{LOW}^k = \{i : C_i^k \le L\}$, and $\delta_i^k = L + 1 - C_i^k$ for $i \in \text{LOW}^k$ denotes the gap between $L+1$ and C_i^k (see Fig. 1).

In other words, to improve on the currently best solution (i.e., the incumbent), machine $i \in \text{LOW}^k$ has to run at least δ_i^k units of time longer than now. Thus, at least

$$l_i^k = \min\left\{\beta \in \{1, \dots, n-k\} : \sum_{j=1}^{\beta} t_{k+j} \ge \delta_i^k\right\} \tag{3}$$

jobs still have to be assigned to $i \in \text{LOW}^k$ in order to finally yield $C_i > L$. If no such β exists, then it is impossible to complete the current partial solution in such a way that a new incumbent solution is obtained. In this case, we set $l_i^k = \infty$. Eventually, the number of required jobs can be updated:

$$\bar{v}_i^k = \begin{cases} \max\{v_i^k, l_i^k\} & \text{if } i \in \text{LOW}^k \\ v_i^k & \text{else.} \end{cases} \tag{4}$$

Hence, a tighter version of Criterion 3.1 reads as follows.

Criterion 3.2 *If*

$$\sum_{i=1}^m \bar{v}_i^k > n - k \tag{5}$$

for some $k < n$, then the current partial solution can be fathomed.

After having updated the number of required jobs, we will incorporate the processing times of the remaining jobs and derive further criteria. Therefore, we let $T_{\text{rem}}^k = \sum_{j=k+1}^n t_j$ denote the *total remaining processing time* and we define

$\Delta^k = \sum_{i \in \text{LOW}^k} \delta_i^k$. Then, it is easy to observe that there is no need to further consider a partial solution if $T_{\text{rem}}^k < \Delta^k$. In this case, it is not realizable to assign the remaining jobs to the machines in LOW^k so that finally $C_i > L$ is fulfilled for all $i \in \text{LOW}^k$, i.e., no matter how we complete the current partial solution, we cannot obtain a new incumbent solution.

In what follows, we are concerned with tightening the inequality $T_{\text{rem}}^k < \Delta^k$, i.e., we want to identify (at low computational costs) a feasible value $R^k > 0$ so that the current solution can already be excluded from further searching whenever

$$T_{\text{rem}}^k - R^k < \Delta^k \tag{6}$$

is fulfilled. For this purpose, in a first step we take a look at those machines which already run longer than L but require at least one more job to fulfill the path-conditions. We summarize these machines in the set $\text{PATH}^k = \{i : C_i^k > L \text{ and } v_i^k > 0\}$ and define

$$V_{\text{PATH}}^k = \sum_{i \in \text{PATH}^k} v_i^k \tag{7}$$

which gives the number of required jobs added over all $i \in \text{PATH}^k$. Then, we readily observe that at least V_{PATH}^k of the remaining $n - k$ jobs cannot be used to fill the gaps on the machines in LOW^k or, equivalently, not more than $n - k - V_{\text{PATH}}^k$ jobs are available for being assigned to the machines in LOW^k. Hence, we can conclude that R^k is at least as large as the sum of the V_{PATH}^k shortest processing times, i.e., $R^k = \sum_{j=1}^{V_{\text{PATH}}^k} t_{n-j+1}$, and (6) appears as follows.

Criterion 3.3 *If*

$$T_{\text{rem}}^k - \sum_{j=1}^{V_{\text{PATH}}^k} t_{n-j+1} < \Delta^k \tag{8}$$

for some $k < n$, then the current partial solution can be fathomed.

To shorten notation, in the remainder of this section, we set $n' := n - V_{\text{PATH}}^k$. Moreover, for sake of readability and since we always consider a partial solution after the assignment of the k longest jobs, we will omit the upper index k.

We will now show that there is still some potential to further increase the value of R. So far, we do not adequately take into account that the jobs are indivisible. Since preemption is not allowed, the processing of a job cannot be interrupted and continued on another machine. Therefore, in contrast to our current criteria, if it is not realizable to select some of the remaining $n' - k$ jobs so that a machine $i \in \text{LOW}$ finishes exactly at time $L + 1$, then the time difference (or surplus) $C_i - (L + 1)$ on machine i cannot be used to increase the completion time of other machines in LOW. Recalling that each machine $i \in \text{LOW}$ requires at least $\bar{v}_i$ more jobs to fulfill its path-conditions and to allow for a new incumbent, we simply check whether the sum of the $\bar{v}_i$ shortest available jobs, i.e., $n', \ldots, n' - \bar{v}_i + 1$, already exceeds the gap δ_i on that machine. If this is the case, then R can be further increased as follows. For each $i \in \text{LOW}$, we compute

$$s_i = \max\left\{C_i + \sum_{j=1}^{\bar{v}_i} t_{n'-j+1} - (L+1), 0\right\} \tag{9}$$

which we call the *individual surplus* on machine i as well as

$$S = \sum_{i \in \text{LOW}} s_i \tag{10}$$

which represents a lower bound on the cumulative surplus caused by the machines $i \in \text{LOW}$. Then, it is obvious that in any complete solution, R will be at least as large as $\sum_{j=1}^{V_{\text{PATH}}^k} t_{n-j+1} + S$.

At this point, we shall remark that (9) (and 10) still assume the V_{PATH} overall shortest jobs to be assigned to the machines in PATH. This assumption is correct because of the following trivial lemma which we state without giving a proof.

Lemma 3.4 *If it is possible to fill all gaps on the machines in LOW using at most $z \leq n' - k$ of the jobs $k + 1, \ldots, n$, then it is possible to fill all gaps on these machines using the z longest jobs $k + 1, \ldots, k + z$.*

As mentioned above, the calculation of the individual surplus s_i for each $i \in \text{LOW}$ (cf. Eq. 9) is a basic approach because the shortest available jobs are supposed to be repeatedly assignable to each machine in LOW. Clearly, this constitutes an essential simplification since in a feasible schedule each job has to be assigned to one machine. Taking this into account, for the subset of machines $\text{LOW1} := \{i \in \text{LOW} : l_i = 1\}$, we will now propose an alternative approach to calculate a lower bound on the cumulative surplus caused by these machines. As will be seen, the restriction to the subset LOW1 of LOW ensures the alternative cumulative surplus to be easily and fast computable. The approach builds on the fact that in total (at least) $|\text{LOW1}|$ jobs have to be assigned to the machines in LOW1 and, since $l_i = 1$ for all $i \in \text{LOW1}$, it might be sufficient to assign a single job to each machine $i \in \text{LOW1}$ to fill the gaps. So, we can take the $|\text{LOW1}|$ shortest available jobs, i.e., $n', \ldots, n' - |\text{LOW1}| + 1$, and—assuming that each of which is assigned to exactly one machine in LOW1—we can calculate a lower bound on the cumulative surplus caused by the machines in LOW1 as follows. Let

$$s_i'(j) = \max\{C_i + t_j - (L+1), 0\} \text{ for } i \in \text{LOW1} \tag{11}$$

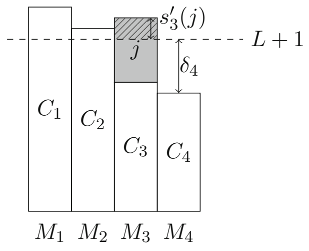


Fig. 2 Illustration of $s'_i(j)$

denote the *(single) surplus* generated by assigning job j to machine $i \in$ LOW1 (see also Fig. 2), the following lemma is readily verified.

Lemma 3.5 *Consider two jobs j_1, j_2 with $t_{j_1} \geq t_{j_2}$ and two machines $i_1, i_2 \in LOW1$ with $C_{i_1} \geq C_{i_2}$. Then, $s'_{i_1}(j_2) + s'_{i_2}(j_1) \leq s'_{i_1}(j_1) + s'_{i_2}(j_2)$.*

Proof If $s'_{i_1}(j_2) = 0$, then $s'_{i_1}(j_2) + s'_{i_2}(j_1) = s'_{i_2}(j_1) \leq s'_{i_1}(j_1) \leq s'_{i_1}(j_1) + s'_{i_2}(j_2)$ since $C_{i_2} + t_{j_1} - (L+1) \leq C_{i_1} + t_{j_1} - (L+1)$.

If $s'_{i_2}(j_1) = 0$, then $s'_{i_1}(j_2)+s'_{i_2}(j_1) = s'_{i_1}(j_2) \leq s'_{i_1}(j_1) \leq s'_{i_1}(j_1)+s'_{i_2}(j_2)$ since $C_{i_1}+t_{j_2}-(L+1) \leq C_{i_1}+t_{j_1}-(L+1)$.

Now, assume that $C_{i_1}+t_{j_2} > L+1$ and $C_{i_2}+t_{j_1} > L+1$. Then,

$$\begin{aligned} s'_{i_1}(j_2) + s'_{i_2}(j_1) &= C_{i_1} + t_{j_2} - (L+1) + C_{i_2} + t_{j_1} - (L+1) \\ &= C_{i_1} + t_{j_1} - (L+1) + C_{i_2} + t_{j_2} - (L+1) \\ &\leq s'_{i_1}(j_1) + s'_{i_2}(j_2). \end{aligned}$$

□

According to Lemma 3.5, assigning the longer job to the machine with the larger gap in LOW1 and the shorter job to the machine with the smaller gap results in a smaller cumulative surplus (on these two machines) than the other way round. Clearly, this pairwise consideration can easily be extended to all machines in LOW1 as summarized in the following corollary.

Corollary 3.6 *Assume the machines in LOW1 to be sorted according to non-increasing gaps. Then, the smallest cumulative surplus caused by the machines in LOW1 is obtained by assigning the |LOW1| shortest available jobs in non-decreasing order of processing times to the machines in LOW1.*

Proof The corollary can easily be proved by contradiction. At first, observe that considering any |LOW1| jobs out of $k+1, \ldots, n'$ cannot yield a smaller cumulative surplus than selecting the |LOW1| shortest ones because the surpluses are monotonically increasing in increasing processing times.

Now, assume that the smallest cumulative surplus is not achieved by the assignment described in Corollary 3.6. Then, there must exist a pair of machines (and jobs) where the machine with the larger gap is assigned the job with the shorter processing time and the machine with the smaller gap is assigned the job with the longer processing time. According to Lemma 3.5, swapping these two jobs results in a smaller cumulative surplus, which is a contradiction. □

Formally, we can summarize Corollary 3.6 as follows: Let π be a permutation of the machines in LOW1 so that $C_{\pi(1)} \geq C_{\pi(2)} \geq \cdots \geq C_{\pi(|\text{LOW1}|)}$, then the smallest cumulative surplus $S1'$ is obtained by assigning job $n'-i+1$ to machine $\pi(i)$ for $i = 1, \ldots, |\text{LOW1}|$, i.e.,

$$S1' = \sum_{i=1}^{|\text{LOW1}|} s'_{\pi(i)}(n'-i+1). \tag{12}$$

Note that again Lemma 3.4 is taken as granted in the computation of the cumulative surplus $S1'$ on the machines in LOW1.

Altogether, we have developed two different approaches to calculate a lower bound on the cumulative surplus caused by the machines in LOW1, namely $S1 := \sum_{i\in\text{LOW1}} s_i$ and $S1'$. It is readily verified that neither $S1$ dominates $S1'$ (think of cases where $\bar{v}_i = l_i = 1$ for all $i \in$ LOW1) nor $S1'$ dominates $S1$. Hence, a lower bound on the cumulative surplus caused by all machines in LOW is given by

$$\bar{S} = \sum_{i\in\text{LOW}\setminus\text{LOW1}} s_i + \max\{S1, S1'\} = S - S1 + \max\{S1, S1'\}. \tag{13}$$

Consequently, we can further increase R by $\bar{S}$ and (6) results in the following criterion that is tighter than Criterion 3.3.

Criterion 3.7 *If*

$$T_{\text{rem}} - \sum_{j=1}^{v_{\text{PATH}}} t_{n'+j} - \bar{S} < \Delta, \tag{14}$$

then the current partial solution can be fathomed.

Summarizing the results developed within this subsection, we can record the following. Given a partial solution where the k longest jobs have already been assigned, in order to obtain a new incumbent solution that fulfills the path-conditions at least $R = \sum_{j=1}^{v_{\text{PATH}}} t_{n'+j} + \bar{S}$ units of the total remaining processing time T_{rem} cannot be used effectively to fill the gaps on the machines in LOW—no matter how the remaining $n-k$ jobs will be assigned.

Example 3.8 The following example briefly illustrates the functionality of the four proposed dominance criteria. Assume $n = 12$, $m = 5$, and the vector of processing times

Table 1 Completion times and gaps

M_i	1	2	3	4	5
C_i	95	93	149	141	129
δ_i	36	38	–	–	2

Table 2 Number of required jobs

M_i	1	2	3	4	5	$\sum$
v_i	0	0	0	1	0	1
l_i	1	1	–	–	1	3
$\bar{v}_i$	1	1	0	1	1	4

Table 3 Surpluses

M_i	1	2	5	$\sum$
s_i	0	0	8	8
s_i'	0	7	8	15

$T = (95, 93, 87, 86, 66, 63, 62, 55, 45, 25, 10, 6)$ for which a lower and an upper bound value of $L = 130$ and $U = 136$ can be determined (cf. Sect. 4), respectively. Furthermore, let $\tilde{S} = (1, 2, 3, 4, 5, 5, 3, 4)$ be the current partial schedule where the $k = 8$ longest jobs have already been assigned and $T_{\text{rem}} = \sum_{j=9}^{12} t_j = 45 + 25 + 10 + 6 = 86$.

Table 1 contains the current machine completion times C_i and the gaps δ_i. The number of required jobs according to Criteria 3.1 and 3.2 is given in Table 2. Since $n - k = 4$ jobs still have to be assigned, the current partial schedule cannot be fathomed on the basis of the first two criteria.

From Tables 1 and 2, we get LOW = LOW1 = $\{1, 2, 5\}$, PATH = $\{4\}$, and $V_{\text{PATH}} = 1$ so that in a first step R is equal to $t_{12} = 6$. Since $T_{\text{rem}} - R = 86 - 6$ is not smaller than $\Delta = 36 + 38 + 2 = 76$, we cannot fathom the current partial solution yet (cf. Criterion 3.3).

Table 3 contains the results of the two alternative surplus computations for the machines in LOW1. According to Criterion 3.7, the current partial solution can be fathomed because $T_{\text{rem}} - 6 - \bar{S} = 86 - 6 - 15 = 65 < 76$.

4 A branch-and-bound algorithm

This section provides details on the developed branch-and-bound algorithm.

4.1 Upper bounds

4.1.1 A trivial bound and its worst-case ratio

Let $T_{\text{sum}} = \sum_{j=1}^{n} t_j$, then $U_0 = \left\lfloor \frac{T_{\text{sum}}}{m} \right\rfloor$ represents the simplest upper bound on the optimal minimum machine completion time $C^*_{\min}$ (cf. Haouari and Jemmali 2008). It is readily verified that the worst-case performance of U_0 can be arbitrarily bad. For instance, assume $n = m + 1$ and consider the processing times $t_1 = K \gg 1$ and $t_j = 1$ for $j = 2, \ldots, n$. Then, $U_0 = \lfloor K/m \rfloor + 1$, whereas $C^*_{\min} = 1$ so that the ratio $U_0/C^*_{\min}$ approaches infinity as K grows.

However, based on the following two observations, the performance of U_0 can be drastically improved. Firstly, note that in case $n = m + k$ ($k \in \{1, \ldots, m - 1\}$) at least $m - k$ machines will process exactly one job in any optimal solution. Thus, the minimum of t_{m-k} and $\lfloor \frac{1}{k} \sum_{j=m-k+1}^{n} t_j \rfloor$ is a valid upper bound on $C^*_{\min}$. Secondly, note that any job whose processing time is greater than or equal to U_0 can be eliminated so that $\lfloor \frac{1}{m-|\mathcal{J}'|} \sum_{j \in \mathcal{J} \setminus \mathcal{J}'} t_j \rfloor$ where $\mathcal{J}' = \{j : t_j \geq U_0\}$ constitutes a valid upper bound on $C^*_{\min}$. Note that $|\mathcal{J}'| \leq m - 1$. Moreover, note that the aforementioned elimination (or reduction) can possibly be repeated up to $m - 1$ times in an iterative manner.

As a result of the previous observations, we can reduce any given instance I to $\tilde{I}$ in such a way that (i) the number of remaining jobs is at least twice the number of remaining machines and (ii) the longest (remaining) processing time is smaller than $U_0(\tilde{I})$. Then, we can prove that $U_0(\tilde{I})$ is never more than $2 - 1/m$ times the optimal minimum completion time.

Theorem 4.1 *Let I be an instance of $P||C_{\min}$ which fulfills both $n \geq 2m$ and $t_1 < U_0(I)$. Then,*

$$\frac{U_0(I)}{C^*_{\min}(I)} \leq 2 - \frac{1}{m} \tag{15}$$

for all instances I and this bound is asymptotically tight for any fixed $m \geq 2$.

Proof We prove the theorem by contradiction. Assume that $U_0(I)/C^*_{\min}(I) > 2 - 1/m$. This is equivalent to $C^*_{\min}(I) < U_0(I)/(2 - 1/m) \leq T_{\text{sum}}/(2m - 1)$. So, in an optimal schedule, the completion time of at least one machine, say i_1, is less than $T_{\text{sum}}/(2m - 1)$. Consequently, there exists at least another machine, say i_2, whose completion time C_{i_2} is at least $(T_{\text{sum}} - T_{\text{sum}}/(2m - 1))/(m - 1) = 2T_{\text{sum}}/(2m - 1) > T_{\text{sum}}/m > t_1$. Thus, i_2 processes at least two jobs among which the shortest job's processing time is at most $C_{i_2}/2 \geq T_{\text{sum}}/(2m - 1)$. Shifting this job from i_2 to i_1 yields a better schedule which is a contradiction.

To verify that the bound is asymptotically tight for any fixed $m \geq 2$, consider $n = 2m$ jobs with processing times $t_1 = \cdots = t_{n-1} = K \gg 1$ and $t_n = 1$. Then, $C^*_{\min} = K + 1$, whereas $U_0 = (2 - 1/m)K + 1/m$. Hence, the ratio of U_0 to $C^*_{\min}$ approaches $2 - 1/m$ as K grows. □

4.1.2 Improvements derived from $P||C_{\max}$

Following Haouari and Jemmali (2008), an upper bound on $C^*_{\min}$ can be derived from a known lower bound $L_{C_{\max}}$ on the optimal makespan by computation of

$$U_1 = \left\lfloor \frac{T_{\text{sum}} - L_{C_{\max}}}{m-1} \right\rfloor. \tag{16}$$

To obtain a good lower bound on the optimal makespan, we implemented an enhanced version (due to Haouari and Jemmali 2008) of the bound L_3 by Dell'Amico and Martello (1995). For further details, we refer the reader to the literature.

4.1.3 Lifting procedure and further enhancement

Haouari and Jemmali (2008) proposed two procedures to tighten upper bounds for $P||C_{\min}$. The first one is the so-called lifting procedure which bases upon the fact that in any feasible schedule there exists at least a set of l machines ($1 \le l \le m$) that process at most

$$\mu_l(n) = l\lfloor n/m \rfloor + \max\{0, n - m(\lfloor n/m \rfloor + 1) + l\} \tag{17}$$

jobs (cf. Haouari and Jemmali 2008). Then, a lifted bound can be obtained by applying an upper bound procedure on the partial instance restricted to l machines and the $\mu_l(n)$ longest jobs.

The second procedure (cf. Haouari and Jemmali 2008) aims at enhancing a given upper bound value U by solving a specific subset sum problem (SSP) that checks whether there exists a subset of $\mathcal{J}$ whose processing times sum up to exactly U. If no such subset exists, the smallest realizable sum (denoted by U_{SSP}) of processing times that does not exceed U constitutes an upper bound.

4.1.4 Improvements derived from bin covering

To the best of our knowledge, we are the first to describe the coherence between $P||C_{\min}$ and the bin covering problem (BCP) in order to improve upper bounds on $C^*_{\min}$. The idea is to transform a given $P||C_{\min}$-instance into a BCP-instance where (i) jobs and processing times correspond to items and weights, respectively, and (ii) the capacity C of the bins is set to the best known upper bound on the minimum completion time. Then, a procedure is applied to determine an upper bound on the maximum number of bins that can be covered. If this number is at most $m-1$, then the optimal minimum completion time of the corresponding $P||C_{\min}$-instance is at most $C-1$.

In our implementation, we used four BCP-upper bounds (U_0 from Peeters and Degraeve 2006, as well as U_1, U_2, and U_3 from Labbé et al. 1995, including their reduction Criteria 1 and 2) and an improvement procedure (see Theorem 5 in Labbé et al. 1995). Again, we refrain from reporting any further details on these bounding techniques but refer the interested reader to the literature.

4.1.5 Bounds derived from the solution structure

Assume that a lower bound L on $C^*_{\min}$ is given (cf. Sect. 4.2). Then, in order to generate a new incumbent solution, i.e., $C_{\min} > L$, we can deduce that at least

$$j_{\min} = \min\{k : t_1 + \cdots + t_k > L\} \tag{18}$$

and at most

$$j_{\max} = \max\{k : t_{n-k+1} + \cdots + t_n \le \bar{C}\} \tag{19}$$

jobs have to be assigned to each machine and

$$\bar{C} = \max\left\{C : \left\lfloor \frac{T_{\text{sum}} - C}{m-1} \right\rfloor > L\right\}. \tag{20}$$

Note that if a machine's completion time is greater than $\bar{C}$, it is impossible that each of the remaining $m-1$ machines runs longer than L.

When the special case $j_{\max} = j_{\min}+1$ occurs, an immediate upper bound is obtained after determining the number of machines $m_{\min}$ ($m_{\max}$) on which exactly $j_{\min}$ ($j_{\max}$) jobs are processed each. It is readily verified that $m_{\min} = j_{\max} \cdot m - n$ and $m_{\max} = n - j_{\min} \cdot m$. The resulting upper bound is

$$U(L) = \min\left\{\left\lfloor \frac{1}{m_{\min}} \sum_{j=1}^{j_{\min}\cdot m_{\min}} t_j \right\rfloor, \left\lfloor \frac{1}{m_{\max}} \sum_{j=1}^{j_{\max}\cdot m_{\max}} t_j \right\rfloor\right\}. \tag{21}$$

In case $j_{\max} = 2$, note that an optimal solution is readily obtained by assigning job j ($j = 1, \ldots, m$) to machine j and job $m+k$ ($k = 1, \ldots, n-m$) to machine $m-k+1$.

If $j_{\max} > j_{\min} + 1$, we propose the following strategy. Firstly, if $\lfloor (T_{\text{sum}} - t_1 - \cdots - t_{j_{\min}})/(m-1) \rfloor \le L$, it is impossible to obtain a new incumbent by assigning the longest $j_{\min}$ jobs to the same machine. In general, a new incumbent can only be obtained at all if none of the machines runs longer than $\tilde{C} := \max\{C > L : \lfloor \frac{T_{\text{sum}}-C}{m-1} \rfloor > L\}$. Thus, if there exists no $j_{\min}$-element subset of $\mathcal{J}$ whose cumulative processing time falls into the interval $[L+1, \tilde{C}]$ we can increase $j_{\min}$.

Instead of checking each subset individually, we solve the following binary program (requiring pseudo-polynomial time) for a possible increase of $j_{\min}$:

$$\text{Minimize } \tilde{j}_{\min} = \textstyle\sum_{j=1}^{n} x_j \tag{22}$$

subject to

$$L + 1 \leq \sum_{j=1}^{n} t_j x_j \leq \tilde{C} \tag{23}$$

$$x_j \in \{0, 1\} \quad \forall j = 1, \ldots, n. \tag{24}$$

In case that now $j_{\max} = \tilde{j}_{\min} + 1$, an immediate upper bound can be determined as described above.

Secondly, we consider restricted instances with l ($l = m-1, m-2, \ldots, 2$) machines and the longest μ_l out of all n jobs where μ_l is determined according to Eq. (17). Then, for each l, we compute a (restricted) $C_{\min}$-lower bound L_l by application of the LPT-rule and we take the maximum of L and L_l (note that the optimal restricted $C_{\min}$-value is at least as large as L) to determine $j_{\min}(l)$ (cf. Eq. 18) as well as $j_{\max}(l)$ (cf. Eq. 19). In case $j_{\max}(l) = j_{\min}(l) + 1$, we obtain an upper bound $U(L_l)$ according to Eq. (21).

4.2 Lower bounds

We implemented three construction procedures as well as an improvement procedure. The first constructive algorithm is the prominent longest processing time (LPT)-rule due to Graham (1969). As a second procedure, we implemented a randomized LPT-version due to Haouari and Jemmali (2008) that randomly decides in each iteration whether the longest or the second longest unassigned job is assigned to the next machine available. Our third procedure is an adaptation of a construction heuristic—referred to as Multi-Subset (MS) (cf. Alvim et al. 2004)—and consists of two phases. In the first phase, the machines are considered one by one. For each machine, a subset of the yet unassigned jobs is determined (by solving a subset sum problem) so that the longest unassigned job is contained and the sum of the respective processing times is closest to—without exceeding—a given target value T. If not all jobs are assigned after phase 1, the second phase completes the partial solution by assigning the remaining jobs according to the LPT-rule. We used two different values for T, namely $T = UB_{\text{best}}$ and $T = LB_{\text{best}} + 1$ where UB_{best} and LB_{best} are the best known upper and lower bound values so far, respectively. The better of the two solutions produced by MS is chosen as *the* MS-solution.

The implemented improvement heuristic—referred to as Multi-Start Local Search (MSLS)—is also due to Haouari and Jemmali (2008). Starting with an initial solution, the procedure attempts to balance the workloads of the machines by iteratively solving a sequence of specific $P2||C_{\min}$-instances. For further details, we refer to Haouari and Jemmali (2008).

4.3 Application of the bounds

At the root node, the bounds are computed in the following order (as long as no optimal solution has been verified). Presupposing a (possibly preprocessed) instance where $n \geq 2m$, at first we determine U_0 and we apply the LPT-rule yielding a global lower bound value L_G. Secondly, we make use of the lifting and enhancement procedure (see Sect. 4.1.3), i.e., for $l = m, m-1, \ldots, 2$ we compute U_1 restricted to l machines and the μ_l longest jobs and afterward we solve for each l the respective SSP. Thirdly, we iteratively apply the bounding techniques derived from bin covering (see Sect. 4.1.4) as long as at least one of them leads to an improved global upper bound U_G. In a fourth step, we try to further improve L_G by application of (i) MS and (ii) MSLS (see Sect. 4.2). The latter is applied to the LPT-solution, the MS-solution as well as to 25 randomized LPT-solutions. Lastly, we apply our upper bounds derived from the solution structure as developed within Sect. 4.1.5.

At each node in the tree, we pursue the following two ideas to obtain local upper bounds. Firstly, we adopt the rationale behind U_0 as follows. In the current partial solution, we replace each currently unassigned job j ($j = k+1, \ldots, n$) by t_j jobs of length 1 and apply the LPT-rule to assign the T^k_{rem} jobs of length 1 to the current partial solution. Clearly, the resulting minimum completion time constitutes a local upper bound which we denote by U_0^{mod}.

Secondly, we partition $\mathcal{M}$ into two subsets $M_{UB^-} = \{M_i \in \mathcal{M} : C_i^k < UB_{loc}\}$ and $M_{UB^+} = \mathcal{M} \setminus M_{UB^-}$ where UB_L denotes the best known local upper bound for the considered node. Note that UB_{loc} is the minimum of U_0^{mod} and the parent node's upper bound value. Then, we compute the following modified variant U_1^{mod} of U_1:

$$U_1^{\text{mod}} = \left\lfloor \frac{T_{\text{sum}} - \sum_{i \in M_{UB^+}} C_i^k - L_3}{|M_{UB^-}| - 1} \right\rfloor. \tag{25}$$

Here L_3 is computed for a transformed instance restricted to (i) the machines in M_{UB^-}, (ii) the $n-k$ remaining jobs, and (iii) $|M_{UB^-}|$ fictitious jobs having processing times C_i^k ($i \in M_{UB^-}$).

To possibly improve on the lower bound, we apply the LPT-rule to partial solutions for which at least m jobs have already been assigned and no more than $2m$ jobs remain unassigned.

4.4 The branching scheme

We implemented a depth-first branching scheme which has originally been proposed by Dell'Amico and Martello (1995) in the context of solving $P||C_{\max}$. We decided on this scheme as it allows for a straightforward incorporation of the path-related dominance criteria. It is different from the one developed by Haouari and Jemmali (2008) and works as follows. At level k, the current node generates at most m son-nodes by sequentially assigning job k, i.e., the longest unassigned job, to each machine M_i that fulfills both $C_i^{k-1} <$

UB_{loc} and $|M_i| < j_{\max}$. The corresponding machines are selected according to increasing current completion times.

Clearly, each new incumbent solution updates the global lower bound. As soon as an optimal solution has been identified, the branching process stops immediately.

4.5 Dominance criteria

In case that a current partial solution at level k cannot be fathomed due to the computation of the local upper bounds and not each pair of machines currently fulfills its path-condition, we apply the path-related dominance criteria in the same order as they are introduced in Sect. 3.

Furthermore, we apply the following criterion derived from Sect. 4.1.5.

Criterion 4.2 *If* $|M_i| + \bar{v}_i^k > j_{\max}$ *for some* $i \in \{1, \ldots, m\}$, *then the current partial solution can be fathomed.*

The branching process itself can be limited according to four dominance criteria that have originally been introduced by Dell'Amico and Martello (1995). In Walter and Lawrinenko (2014) it is shown how these four criteria have to be modified in order to be compatible with the path-related criteria. For further details, we refer to the literature.

5 Computational study

We have coded the algorithm described within Sect. 4 in C++ using the Visual C++ 2010 compiler and experimentally tested its performance on (i) various difficult sets of $P||C_{\min}$-instances as reported by Haouari and Jemmali (2008) and (ii) a large set of instances from the literature. Our computational experiments were performed on a personal computer with an Intel Core i7-2600 processor and 8GB RAM while running Windows 7 Professional SP 1 (64-bit). The maximal computation time per instance was set to 600 s.

5.1 Performance on Dell'Amico and Martello's instances

In a first experiment, we have run our algorithm (denoted by WWL) on the following five problem classes originally proposed by Dell'Amico and Martello (1995):

- Class 1: discrete uniform distribution on [1, 100]
- Class 2: discrete uniform distribution on [20, 100]
- Class 3: discrete uniform distribution on [50, 100]
- Class 4: cut-off normal distribution with $\mu = 100$ and $\sigma = 50$
- Class 5: cut-off normal distribution with $\mu = 100$ and $\sigma = 20$

According to the results stated in the paper by Haouari and Jemmali (2008), with their algorithm (denoted by HJ) branching was required for only 166 out of 2050 instances. Since our global bounds are in general at least as strong, there is no need to reconsider all of their investigated constellations. Instead, we restricted our study to the difficult (n, m)-constellations $(10, 3), (10, 5), (25, 10), (25, 15), (50, 15)$ where branching was required by HJ. Like Haouari and Jemmali (2008), for each constellation we randomly generated 10 independent instances resulting in a total of 250 instances. A summary of the results is provided in Table 4. In this table we document the mean CPU time in seconds (labeled as "Time") as well as the mean number of explored nodes ("NN"). Moreover, the number of unsolved instances if greater than 0 is given in brackets. The results of algorithm HJ are taken from Haouari and Jemmali (2008). Their algorithm was coded in Visual C++ 6.0 and ran on a Pentium IV 3.2GHz with 3GB RAM. The time limit for HJ was set to 800 s.

Additionally, in Table 4, we report on the performance of the new upper and lower bound techniques (cf. columns labeled as "UB_{impr}" and "LB_{MS}") introduced within the Sects. 4.1.4, 4.1.5, and 4.2. Concerning the new upper bounds, the column labeled as "#" gives the number of instances where the new upper bounds improved on the best upper bound obtained by the existing procedures (cf. Sects. 4.1.1–4.1.3). Numbers in brackets indicate for how many instances (if less than 10) the application of at least one of the two new upper bounding techniques has been needed. The columns labeled as "avg" and "max" give the average and maximum relative deviation (in %) between the best existing upper bound and the best new upper bound. Concerning the new lower bound, the column labeled as "#" gives the number of instances where Multi-Subset (MS) generates the best lower bound value. Numbers in brackets indicate for how many instances (if less than 10) the application of MS has been needed. For those cases, the columns labeled as "avg" and "max" give the average and maximum relative deviation (in %) between the best lower bound value and the value produced by MS.

As can be seen from the entries in Table 4, except for constellation (50, 15) where algorithm HJ performed generally better than WWL, the opposite is true for each of the four other constellations. Here, our algorithm is strictly superior to algorithm HJ in terms of NN. Moreover, WWL was able to quickly solve all 200 instances, while algorithm HJ failed to solve two of them. The dominance of WWL is particularly impressive for the constellations (25, 10) and (25, 15) where the average number of generated nodes and the computation time were reduced to fractions of up to 1/14,500,000 and 1/8650, respectively. However, the benefit of the dominance criteria (cf. Sect. 3) seems to be limited to constellations where $n/m \leq 2.5$. These observations comply with the findings in Walter and Lawrinenko (2014) for problem $P||C_{\max}$.

Table 4 Results on difficult instances generated according to Dell'Amico and Martello (1995)

n	m	Class	HJ		WWL							
			Time	NN	Time	NN	UB_{impr}			LB_{MS}		
							#	avg	max	#	avg	max
10	3	1	0.001	90	0.016	21	0	0	0	4	2.307	5.645
		2	0.001	137	0.010	13	1	0.054	0.543	4	1.627	2.183
		3	0.002	48	0.020	16	0	0	0	3	1.161	3.309
		4	0.002	215	0.011	6	2	0.095	0.610	5	1.342	2.215
		5	0.002	100	0.012	30	0	0	0	3	1.265	2.652
10	5	1	0.002	128	0.010	3	5 (9)	3.179	13.084	4 (5)	0.323	1.613
		2	0.002	152	0.009	3	9	5.573	12.598	1 (3)	10.149	19.200
		3	<0.001	188	0.015	2	9	1.868	3.472	0 (1)	4.138	4.138
		4	0.003	150	0.009	2	9	2.939	8.938	0 (1)	7.292	7.292
		5	0.002	140	0.009	5	8	7.093	18.239	1 (4)	4.968	10.329
25	10	1	85.000	19,649,828	0.115	3,534	0	0	0	0	3.300	5.185
		2	129.000 (1)	31,675,367	0.083	1,965	5	0.325	0.704	0	5.226	8.725
		3	1.380	235,178	0.027	34	3	0.340	1.143	0	3.308	5.650
		4	19.625	4,294,945	0.018	13	5	1.001	3.196	0	1.830	3.644
		5	38.675	10,014,760	1.599	53,203	5	0.708	3.968	0	5.412	8.696
25	15	1	128.000 (1)	24,572,119	0.019	26	1 (5)	0.606	3.030	1 (5)	10.868	28.000
		2	1.900	268,948	0.009	4	1 (1)	5.952	5.952	0 (1)	2.500	2.500
		3	0.006	42	0.023	1	0 (0)	–	–	0 (0)	–	–
		4	77.878	14,537,221	0.009	1	0 (0)	–	–	0 (0)	–	–
		5	0.005	48	0.010	22	0 (1)	0	0	0 (1)	1.515	1.515
50	15	1	0.003	17	4.050 (1)	95,149	0	0	0	0	6.963	12.500
		2	0.001	21	0.015 (1)	1	0	0	0	1	3.021	6.283
		3	0.001 (2)	1	104.883 (6)	2,740,736	0	0	0	0	4.740	6.996
		4	20.335	4,498,587	93.392 (2)	1,961,145	0	0	0	0	5.683	6.548
		5	0.001 (2)	1	94.611 (4)	2,146,798	0	0	0	0	5.883	15.805

The performance of the new bounding techniques can be summarized as follows. Starting with the upper bounds, application of the new techniques has been required for 206 out of 250 instances and improvements were achieved for 63 of them (mostly due to the bounds derived from BCP). While the overall average relative improvement is about 1.153 %, maximum relative improvements of up to almost 19 % were obtained (cf. constellation (10, 5), Class 5). The new upper bounding techniques performed remarkably well at constellation (10, 5) where upper bounds could be tightened for 40 (of 49) instances. With regard to the new lower bound, application of the MS-heuristic has been required for 171 out of 250 instances (note that for the remaining 79 instances the LPT-solution had already been identified as an optimal solution). For 27 of the 171 instances, MS generated the best lower bound or, in other words, the MSLS-heuristic was not able to improve the MS-solution. All in all, our proposed construction heuristic MS performed quite well yielding a deviation from the MSLS-value of less than 3.5 % on average.

5.2 Performance on Haouari and Jemmali's instances

In the second experiment, we have run our algorithm on a class of problems where the processing times are drawn from a discrete uniform distribution on $[1, n]$ as proposed by Haouari and Jemmali (2008). Again, we restricted our study to those (n, m)-constellations where not all of the 10 generated instances in Haouari and Jemmali (2008) had been solved at the root node by their algorithm. This time, these are the three constellations (10, 5), (25, 10), and (25, 15). For each of them we randomly generated 10 instances. Table 5 provides a comparison of our results with the ones reported in Haouari and Jemmali (2008).

Obviously, our findings of the first experiment also apply to the results obtained for the second experiment (see Table 5)

Table 5 Results on instances generated according to Haouari and Jemmali (2008)

n	m	HJ		WWL							
		Time	NN	Time	NN	UB_{impr}			LB_{MS}		
						#	avg	max	#	avg	max
10	5	<0.001	101	0.013	1	0 (0)	–	–	0 (0)	–	–
25	10	21.165	3,954,509	0.017	169	0 (6)	0	0	1 (6)	4.163	6.250
25	15	73.786	12,301,985	0.014	15	1 (4)	1.191	4.762	1 (3)	9.450	13.636

which reveal a clear dominance of our algorithm for the constellations (25, 10) and (25, 15). Compared to algorithm HJ, our approach reduced the average number of generated nodes and the computation time to fractions of up to 1/820,000 and 1/5270, respectively.

5.3 Performance on benchmark instances

In a third experiment, we investigated WWL's performance on 780 benchmark instances originally proposed by França et al. (1994) as well as Frangioni et al. (2004) for problem $P||C_{\max}$. While the former set consists of 390 uniform instances, the latter contains 390 non-uniform instances. For a detailed description, we refer to the literature. All instances can be downloaded from http://www.or.deis.unibo.it/research.html. To the best of our knowledge, we are the first to use this established benchmark set in the context of $P||C_{\min}$.

Table 6 reports on the results obtained by our algorithm. In addition to the mean CPU time and the mean number of nodes, this time we also document the average as well as maximum relative deviation in % (labeled as "Gap" and "maxGap," respectively) between the best upper bound value and the best lower bound value computed at the root node. Averages are taken over all 10 instances per triple $(n, m, Interval)$.

The results indicate that WWL's performance on the uniform instances is better than that on the set of non-uniform

Table 6 Results on benchmark instances

n	m	Interval	Uniform				Non-uniform			
			Time	NN	Gap	maxGap	Time	NN	Gap	maxGap
10	5	$[1, 10^2]$	0.011	5	0.820	5.747	0.015	1	0	0
		$[1, 10^3]$	0.009	5	0.681	6.570	0.010	1	0	0
		$[1, 10^4]$	0.034	7	2.643	9.320	0.010	1	0	0
50	5	$[1, 10^2]$	0.009	1	0	0	– (10)	–	3.078	3.807
		$[1, 10^3]$	0.017	1	0	0	– (10)	–	3.561	4.224
		$[1, 10^4]$	0.125	1	0	0	– (10)	–	3.605	4.274
	10	$[1, 10^2]$	0.012	1	0	0	– (10)	–	13.826	17.085
		$[1, 10^3]$	0.016 (5)	1	0.021	0.044	– (10)	–	14.397	17.485
		$[1, 10^4]$	– (10)	–	0.016	0.025	– (10)	–	14.471	17.561
	25	$[1, 10^2]$	0.192	3,881	3.280	10.000	0.009	1	0	0
		$[1, 10^3]$	3.124 (3)	29,326	5.292	16.404	0.021	1	0	0
		$[1, 10^4]$	25.984 (5)	17,577	4.850	10.069	0.103	1	0	0
100	5	$[1, 10^2]$	0.009	1	0	0	0.017	1	0	0
		$[1, 10^3]$	0.033	1	0	0	0.117 (1)	1	0.001	0.005
		$[1, 10^4]$	0.182	1	0	0	2.236	1	0	0
	10	$[1, 10^2]$	0.012	1	0	0	– (10)	–	3.460	4.566
		$[1, 10^3]$	0.037	1	0	0	– (10)	–	3.813	4.813
		$[1, 10^4]$	0.335	1	0	0	– (10)	–	3.859	4.856
	25	$[1, 10^2]$	0.015	1	0	0	0.035 (6)	1	12.560	22.923
		$[1, 10^3]$	– (10)	–	0.075	0.112	– (10)	–	21.047	23.617
		$[1, 10^4]$	– (10)	–	0.051	0.087	– (10)	–	21.127	23.741

Table 6 continued

n	m	Interval	Uniform				Non-uniform			
			Time	NN	Gap	maxGap	Time	NN	Gap	maxGap
500	5	$[1, 10^2]$	0.006	1	0	0	0.085	1	0	0
		$[1, 10^3]$	0.334	1	0	0	0.567	1	0	0
		$[1, 10^4]$	2.679	1	0	0	5.293	1	0	0
	10	$[1, 10^2]$	0.019	1	0	0	0.127	1	0	0
		$[1, 10^3]$	0.395	1	0	0	0.787	1	0	0
		$[1, 10^4]$	2.418	1	0	0	4.989	1	0	0
	25	$[1, 10^2]$	0.058	1	0	0	0.114	1	0	0
		$[1, 10^3]$	0.443	1	0	0	1.076	1	0	0
		$[1, 10^4]$	3.027	1	0	0	7.302	1	0	0
1000	5	$[1, 10^2]$	0.008	1	0	0	0.304	1	0	0
		$[1, 10^3]$	0.434	1	0	0	1.825	1	0	0
		$[1, 10^4]$	9.559	1	0	0	19.402	1	0	0
	10	$[1, 10^2]$	0.011	1	0	0	0.220	1	0	0
		$[1, 10^3]$	1.044	1	0	0	1.798	1	0	0
		$[1, 10^4]$	8.543	1	0	0	17.481	1	0	0
	25	$[1, 10^2]$	0.063	1	0	0	0.373	1	0	0
		$[1, 10^3]$	1.630	1	0	0	3.568	1	0	0
		$[1, 10^4]$	8.857	1	0	0	24.293	1	0	0

instances. While 347 out of the 390 uniform instances have been solved optimally, only 273 out of the 390 non-uniform instances have been solved optimally leading to a total number of 620 solved benchmark instances within the time limit of 600 s per instance. The unsolved uniform instances belong to the three (n, m)-constellations (50, 10), (50, 25), and (100, 25) and the intervals $[1, 10^3]$ and $[1, 10^4]$. In contrast, all but one of the unsolved instances of the non-uniform set belong to the four (n, m)-constellations (50, 5), (50, 10), (100, 10), and (100, 25) and all three intervals. Taking a look at the entries in the columns Gap and maxGap of the unsolved instances, we record considerably smaller values for the uniform than for the non-uniform instances. For the latter, we observed average and maximum deviations of up to 21.1 and 23.7%, respectively. This indicates that non-uniform instances seem to be more difficult to solve than uniform instances.

To allow for future comparisons, we document detailed results on the best found objective function value as well as the best found upper bound value for each of the 780 instances in the "Appendix".

6 Conclusions

The paper on hand addressed the machine covering problem $P||C_{\min}$ and its exact solution. We identified structural properties of optimal schedules from which we deduced the so-called *path-related* dominance criteria. Representing the key characteristic of the proposed branch-and-bound algorithm, these novel criteria proved to be effective in limiting the search space—particularly in the case of rather small ratios of n to m. For those constellations, our approach is superior to the one presented in Haouari and Jemmali (2008).

For future research on $P||C_{\min}$, we suggest three interesting directions with regard to exact as well as heuristic procedures. Firstly, a quite promising advancement of our branch-and-bound algorithm might consist in the implementation of a sophisticated branching scheme that (even) more directly exploits the structural properties of potentially (unique) optimal solutions and thus enforces the generation of respective (partial) solutions. Secondly, we encourage the development of an innovative dynamic programming (DP) formulation of $P||C_{\min}$ that makes use of the structural properties identified in this paper to trim the state space of the DP aiming at a possible improvement on the time complexity or the space requirements of the sole PTAS existing so far. Thirdly, due to the exponential nature of exactly solving the machine covering problem, efficient (meta-) heuristic approaches such as Tabu search or population-based algorithms are still needed to tackle large-sized instances. Here, it is conceivable to punish the non-fulfillment of the path-conditions by adequately reducing the fitness value of such solutions.

Acknowledgements We would like to thank the two anonymous referees for their valuable comments that helped to improve the presentation of the paper and for suggesting shorter proofs of Theorem 4.1 and Lemma 3.5.

Appendix: Detailed results on benchmark instances

Tables 7 and 8 contain detailed results for each of the 390 uniform and 390 non-uniform benchmark instances, respectively. Each row corresponds to a triple $(n, m, \text{Interval})$ and each column to one of the 10 instances per triple. For each

Table 7 Detailed results on all 390 uniform instances

n	m	Interval		No.									
				0	1	2	3	4	5	6	7	8	9
10	5	$[1, 10^2]$	Best	**87**	**75**	**101**	**100**	**68**	**95**	**93**	**89**	**100**	**85**
			UB	92	76	101	100	68	95	93	90	100	85
		$[1, 10^3]$	Best	**1149**	**898**	**844**	**1046**	**631**	**805**	**545**	**1066**	**1112**	**939**
			UB	1149	957	846	1046	631	805	545	1066	1112	939
		$[1, 10^4]$	Best	**11,493**	**10,634**	**8813**	**6763**	**9454**	**8970**	**9943**	**10,016**	**9968**	**10,223**
			UB	11,493	10,634	8813	6823	9454	9806	9943	10,016	10,878	10,948
50	5	$[1, 10^2]$	Best	**514**	**559**	**451**	**571**	**539**	**496**	**540**	**535**	**471**	**497**
			UB	514	559	451	571	539	496	540	535	471	497
		$[1, 10^3]$	Best	**4950**	**5810**	**4624**	**4854**	**4409**	**5072**	**5148**	**5007**	**5485**	**5334**
			UB	4950	5810	4624	4854	4409	5072	5148	5007	5485	5334
		$[1, 10^4]$	Best	**55,927**	**53,504**	**50,056**	**45,688**	**55,528**	**51,621**	**45,106**	**41,226**	**51,986**	**42,169**
			UB	55,927	53,504	50,056	45,688	55,528	51,621	45,106	41,226	51,986	42,169
	10	$[1, 10^2]$	Best	**241**	**226**	**209**	**271**	**251**	**276**	**256**	**229**	**234**	**234**
			UB	241	226	209	271	251	276	256	229	234	234
		$[1, 10^3]$	Best	2546	2308	2497	**2176**	**2612**	2403	**2417**	**2795**	2283	**2456**
			UB	2547	2309	2498	2176	2612	2404	2417	2795	2284	2456
		$[1, 10^4]$	Best	26,660	26,227	23,667	27,758	27,202	25,289	23,383	25,476	32,262	26,701
			UB	26,661	26,233	23,673	27,764	27,204	25,295	23,387	25,478	32,265	26,706
	25	$[1, 10^2]$	Best	**97**	**96**	**95**	**99**	**99**	**100**	**94**	**86**	**93**	**100**
			UB	102	97	99	99	99	110	94	86	94	107
		$[1, 10^3]$	Best	863	**997**	**953**	**989**	**951**	**907**	921	**940**	**942**	908
			UB	899	1073	977	999	1107	929	938	1059	950	936
		$[1, 10^4]$	Best	**10,355**	9013	7807	7390	**9311**	**9802**	8267	**9856**	**8819**	9079
			UB	11,004	9306	7884	7483	9572	10,789	8537	10,674	9361	9298
100	5	$[1, 10^2]$	Best	**921**	**951**	**972**	**1050**	**1008**	**1082**	**1044**	**976**	**1063**	**1004**
			UB	921	951	972	1050	1008	1082	1044	976	1063	1004
		$[1, 10^3]$	Best	**10,472**	**10,858**	**10,028**	**10,236**	**9070**	**9234**	**10,537**	**10,133**	**10,270**	**9001**
			UB	10,472	10,858	10,028	10,236	9070	9234	10,537	10,133	10,270	9001
		$[1, 10^4]$	Best	**105,081**	**91,044**	**91,927**	**96,921**	**91,927**	**96,925**	**98,933**	**106,194**	**110,073**	**105,081**
			UB	105,081	91,044	91,927	96,921	91,927	96,925	98,933	106,194	110,073	105,081
	10	$[1, 10^2]$	Best	**545**	**441**	**507**	**492**	**517**	**529**	**515**	**514**	**493**	**460**
			UB	545	441	507	492	517	529	515	514	493	460
		$[1, 10^3]$	Best	**5335**	**5416**	**5011**	**5005**	**5183**	**4722**	**4984**	**5034**	**5503**	**5016**
			UB	5335	5416	5011	5005	5183	4722	4984	5034	5503	5016
		$[1, 10^4]$	Best	**46,655**	**46,158**	**52,285**	**51,413**	**52,540**	**56,580**	**45,522**	**52,421**	**48,460**	**53,151**
			UB	46,655	46,158	52,285	51,413	52,540	56,580	45,522	52,421	48,460	53,151
	25	$[1, 10^2]$	Best	**194**	**194**	**198**	**199**	**190**	**218**	**193**	**213**	**204**	**202**
			UB	194	194	198	199	190	218	193	213	204	202

Table 7 continued

n	m	Interval		No.									
				0	1	2	3	4	5	6	7	8	9
		$[1, 10^3]$	Best	2089	1934	1940	1950	1790	2125	2192	1877	1894	2195
			UB	2091	1935	1941	1952	1792	2127	2193	1878	1895	2197
		$[1, 10^4]$	Best	21,161	17,181	21,561	20,830	20,557	20,682	20,014	19,260	20,591	19,111
			UB	21,168	17,196	21,571	20,841	20,567	20,694	20,020	19,271	20,597	19,123
500	5	$[1, 10^2]$	Best	**5106**	**4970**	**5085**	**4887**	**4925**	**5220**	**5152**	**4840**	**4917**	**5004**
			UB	5106	4970	5085	4887	4925	5220	5152	4840	4917	5004
		$[1, 10^3]$	Best	**49,375**	**49,881**	**49,375**	**52,902**	**51,401**	**52,852**	**49,558**	**50,613**	**47,180**	**48,867**
			UB	49,375	49,881	49,375	52,902	51,401	52,852	49,558	50,613	47,180	48,867
		$[1, 10^4]$	Best	**504,251**	**500,430**	**513,001**	**485,975**	**501,999**	**492,923**	**508,864**	**488,853**	**494,248**	**505,127**
			UB	504,251	500,430	513,001	485,975	501,999	492,923	508,864	488,853	494,248	505,127
	10	$[1, 10^2]$	Best	**2519**	**2450**	**2537**	**2392**	**2504**	**2541**	**2533**	**2512**	**2568**	**2543**
			UB	2519	2450	2537	2392	2504	2541	2533	2512	2568	2543
		$[1, 10^3]$	Best	**24,921**	**24,640**	**24,255**	**24,218**	**25,308**	**25,086**	**24,206**	**25,992**	**25,876**	**24,875**
			UB	24,921	24,640	24,255	24,218	25,308	25,086	24,206	25,992	25,876	24,875
		$[1, 10^4]$	Best	**265,229**	**263,643**	**256,023**	**244,038**	**249,160**	**261,390**	**239,883**	**257,333**	**241,659**	**254,780**
			UB	265,229	263,643	256,023	244,038	249,160	261,390	239,883	257,333	241,659	254,780
	25	$[1, 10^2]$	Best	**1015**	**950**	**997**	**994**	**1012**	**974**	**990**	**999**	**952**	**987**
			UB	1015	950	997	994	1012	974	990	999	952	987
		$[1, 10^3]$	Best	**9628**	**9924**	**10,107**	**9988**	**10,224**	**10,474**	**10,227**	**9729**	**10,128**	**9683**
			UB	9628	9924	10,107	9988	10,224	10,474	10,227	9729	10,128	9683
		$[1, 10^4]$	Best	**99,712**	**101,630**	**96,835**	**96,084**	**96,221**	**102,001**	**102,493**	**100,419**	**99,712**	**97,049**
			UB	99,712	101,630	96,835	96,084	96,221	102,001	102,493	100,419	99,712	97,049
1000	5	$[1, 10^2]$	Best	**9989**	**10,262**	**10,072**	**10,105**	**9903**	**9782**	**10,048**	**9860**	**10,064**	**9826**
			UB	9989	10,262	10,072	10,105	9903	9782	10,048	9860	10,064	9826
		$[1, 10^3]$	Best	**102,391**	**98,203**	**98,085**	**101,873**	**99,749**	**100,072**	**98,725**	**99,387**	**101,817**	**99,329**
			UB	102,391	98,203	98,085	101,873	99,749	100,072	98,725	99,387	101,817	99,329
		$[1, 10^4]$	Best	**1,001,418**	**993,453**	**1,012,868**	**1,011,507**	**988,262**	**968,878**	**996,506**	**1,006,553**	**999,716**	**1,012,238**
			UB	1,001,418	993,453	1,012,868	1,011,507	988,262	968,878	996,506	1,006,553	999,716	1,012,238
	10	$[1, 10^2]$	Best	**4836**	**5020**	**5109**	**4925**	**5118**	**4884**	**4954**	**5152**	**4854**	**4885**
			UB	4836	5020	5109	4925	5118	4884	4954	5152	4854	4885
		$[1, 10^3]$	Best	**50,317**	**48,373**	**48,582**	**50,546**	**50,555**	**49,107**	**49,300**	**50,101**	**50,232**	**49,954**
			UB	50,317	48,373	48,582	50,546	50,555	49,107	49,300	50,101	50,232	49,954
		$[1, 10^4]$	Best	**495,462**	**482,122**	**494,773**	**502,895**	**483,590**	**488,311**	**492,996**	**511,505**	**506,910**	**512,238**
			UB	495,462	482,122	494,773	502,895	483,590	488,311	492,996	511,505	506,910	512,238
	25	$[1, 10^2]$	Best	**2024**	**1930**	**2026**	**2023**	**2007**	**1958**	**2032**	**1965**	**2024**	**2047**
			UB	2024	1930	2026	2023	2007	1958	2032	1965	2024	2047
		$[1, 10^3]$	Best	**20,305**	**20,100**	**19,821**	**20,462**	**19,360**	**20,021**	**20,556**	**20,319**	**20,122**	**19,119**
			UB	20,305	20,100	19,821	20,462	19,360	20,021	20,556	20,319	20,122	19,119
		$[1, 10^4]$	Best	**202,497**	**198,820**	**205,820**	**202,422**	**197,545**	**203,788**	**195,468**	**201,132**	**202,464**	**200,507**
			UB	202,497	198,820	205,820	202,422	197,545	203,788	195,468	201,132	202,464	200,507

Table 8 Detailed results on all 390 non-uniform instances

n	m	Interval		No.									
				0	1	2	3	4	5	6	7	8	9
10	5	$[1, 10^2]$	Best	**100**	**115**	**103**	**104**	**110**	**113**	**102**	**119**	**113**	**107**
			UB	100	115	103	104	110	113	102	119	113	107
		$[1, 10^3]$	Best	**992**	**1135**	**1030**	**1035**	**1093**	**1121**	**1006**	**1187**	**1124**	**1066**
			UB	992	1135	1030	1035	1093	1121	1006	1187	1124	1066
		$[1, 10^4]$	Best	**9913**	**11,335**	**10,290**	**10,351**	**10,921**	**11,204**	**10,055**	**11,862**	**11,230**	**10,653**
			UB	9913	11,335	10,290	10,351	10,921	11,204	10,055	11,862	11,230	10,653
50	5	$[1, 10^2]$	Best	902	912	905	893	898	901	906	901	909	914
			UB	935	931	931	927	925	932	935	931	941	931
		$[1, 10^3]$	Best	8984	9068	9004	8901	8942	8987	9032	8956	9051	9100
			UB	9348	9307	9316	9277	9265	9328	9348	9312	9406	9321
		$[1, 10^4]$	Best	89,811	90,647	90,001	88,978	89,389	89,836	90,290	89,511	90,473	90,953
			UB	93,483	93,084	93,162	92,781	92,647	93,283	93,477	93,118	94,053	93,214
	10	$[1, 10^2]$	Best	398	413	410	412	419	405	414	410	410	402
			UB	466	465	468	468	465	463	469	465	464	465
		$[1, 10^3]$	Best	3969	4103	4084	4102	4181	4034	4120	4084	4082	4005
			UB	4663	4660	4677	4678	4654	4638	4691	4654	4649	4660
		$[1, 10^4]$	Best	39,661	41,002	40,811	40,991	41,800	40,314	41,176	40,820	40,796	40,026
			UB	46,626	46,602	46,776	46,777	46,548	46,382	46,907	46,547	46,495	46,604
	25	$[1, 10^2]$	Best	**104**	**117**	**105**	**108**	**101**	**103**	**109**	**113**	**118**	**119**
			UB	104	117	105	108	101	103	109	113	118	119
		$[1, 10^3]$	Best	**1039**	**1162**	**1044**	**1076**	**1006**	**1028**	**1085**	**1125**	**1176**	**1183**
			UB	1039	1162	1044	1076	1006	1028	1085	1125	1176	1183
		$[1, 10^4]$	Best	**10,378**	**11,614**	**10,432**	**10,751**	**10,054**	**10,266**	**10,843**	**11,244**	**11,750**	**11,825**
			UB	10,378	11,614	10,432	10,751	10,054	10,266	10,843	11,244	11,750	11,825
100	5	$[1, 10^2]$	Best	**1873**	**1861**	**1863**	**1864**	**1873**	**1870**	**1862**	**1869**	**1867**	**1866**
			UB	1873	1861	1863	1864	1873	1870	1862	1869	1867	1866
		$[1, 10^3]$	Best	**18,718**	**18,616**	**18,641**	**18,654**	18,727	**18,700**	**18,625**	**18,690**	**18,667**	**18,656**
			UB	18,718	18,616	18,641	18,654	18,728	18,700	18,625	18,690	18,667	18,656
		$[1, 10^4]$	Best	**187,165**	**186,177**	**186,406**	**186,542**	**187,273**	**187,000**	**186,260**	**186,915**	**186,668**	**186,557**
			UB	187,165	186,177	186,406	186,542	187,273	187,000	186,260	186,915	186,668	186,557
	10	$[1, 10^2]$	Best	911	898	908	904	910	901	898	905	896	895
			UB	934	933	936	930	934	933	939	936	930	933
		$[1, 10^3]$	Best	9081	8959	9039	9010	9078	8963	8955	9016	8937	8925
			UB	9339	9338	9356	9304	9344	9329	9386	9360	9301	9333
		$[1, 10^4]$	Best	90,767	89,548	90,354	90,068	90,754	89,582	89,513	90,114	89,331	89,213
			UB	93,390	93,386	93,568	93,040	93,441	93,293	93,860	93,596	93,016	93,328
	25	$[1, 10^2]$	Best	304	**312**	303	312	305	318	**314**	**306**	**313**	301
			UB	373	312	370	374	371	370	314	306	313	370
		$[1, 10^3]$	Best	3030	3111	3022	3110	3042	3169	3136	3047	3127	3002
			UB	3730	3734	3706	3739	3721	3703	3741	3734	3748	3711
		$[1, 10^4]$	Best	30,288	31,091	30,209	31,086	30,410	31,671	31,345	30,454	31,260	29,995
			UB	37,310	37,338	37,071	37,398	37,216	37,043	37,416	37,343	37,481	37,116
500	5	$[1, 10^2]$	Best	**9406**	**9419**	**9398**	**9400**	**9380**	**9407**	**9380**	**9395**	**9371**	**9391**
			UB	9406	9419	9398	9400	9380	9407	9380	9395	9371	9391

Table 8 continued

				No.									
n	m	Interval		0	1	2	3	4	5	6	7	8	9
		$[1, 10^3]$	Best	**94,044**	**94,178**	**93,985**	**94,003**	**93,820**	**94,064**	**93,822**	**93,937**	**93,761**	**93,884**
			UB	94,044	94,178	93,985	94,003	93,820	94,064	93,822	93,937	93,761	93,884
		$[1, 10^4]$	Best	**940,444**	**941,762**	**939,851**	**940,029**	**938,230**	**940,641**	**938,208**	**939,385**	**937,623**	**938,833**
			UB	940,444	941,762	939,851	940,029	938,230	940,641	938,208	939,385	937,623	938,833
	10	$[1, 10^2]$	Best	**4702**	**4698**	**4685**	**4700**	**4695**	**4705**	**4703**	**4706**	**4694**	**4705**
			UB	4702	4698	4685	4700	4695	4705	4703	4706	4694	4705
		$[1, 10^3]$	Best	**47,012**	**46,979**	**46,863**	**46,998**	**46,953**	**47,061**	**47,022**	**47,055**	**46,943**	**47,060**
			UB	47,012	46,979	46,863	46,998	46,953	47,061	47,022	47,055	46,943	47,060
		$[1, 10^4]$	Best	**470,117**	**469,780**	**468,636**	**469,993**	**469,537**	**470,608**	**470,225**	**470,546**	**469,430**	**470,592**
			UB	470,117	469,780	468,636	469,993	469,537	470,608	470,225	470,546	469,430	470,592
	25	$[1, 10^2]$	Best	**1877**	**1877**	**1875**	**1878**	**1881**	**1881**	**1880**	**1883**	**1883**	**1878**
			UB	1877	1877	1875	1878	1881	1881	1880	1883	1883	1878
		$[1, 10^3]$	Best	**18,775**	**18,769**	**18,755**	**18,782**	**18,810**	**18,805**	**18,804**	**18,824**	**18,830**	**18,783**
			UB	18,775	18,769	18,755	18,782	18,810	18,805	18,804	18,824	18,830	18,783
		$[1, 10^4]$	Best	**187,751**	**187,692**	**187,560**	**187,824**	**188,102**	**188,060**	**188,047**	**188,245**	**188,291**	**187,832**
			UB	187,751	187,692	187,560	187,824	188,102	188,060	188,047	188,245	188,291	187,832
1000	5	$[1, 10^2]$	Best	**18,801**	**18,804**	**18,801**	**18,821**	**18,812**	**18,824**	**18,807**	**18,818**	**18,820**	**18,805**
			UB	18,801	18,804	18,801	18,821	18,812	18,824	18,807	18,818	18,820	18,805
		$[1, 10^3]$	Best	**188,043**	**188,038**	**188,013**	**188,211**	**188,106**	**188,206**	**188,081**	**188,167**	**188,169**	**188,061**
			UB	188,043	188,038	188,013	188,211	188,106	188,206	188,081	188,167	188,169	188,061
		$[1, 10^4]$	Best	**1,880,395**	**1,880,345**	**1,880,141**	**1,882,117**	**1,881,050**	**1,882,024**	**1,880,834**	**1,881,686**	**1,881,697**	**1,880,627**
			UB	1,880,395	1,880,345	1,880,141	1,882,117	1,881,050	1,882,024	1,880,834	1,881,686	1,881,697	1,880,627
	10	$[1, 10^2]$	Best	**9409**	**9421**	**9402**	**9396**	**9408**	**9404**	**9388**	**9408**	**9398**	**9406**
			UB	9409	9421	9402	9396	9408	9404	9388	9408	9398	9406
		$[1, 10^3]$	Best	**94,087**	**94,190**	**94,012**	**93,957**	**94,091**	**94,041**	**93,926**	**94,081**	**93,993**	**94,061**
			UB	94,087	94,190	94,012	93,957	94,091	94,041	93,926	94,081	93,993	94,061
		$[1, 10^4]$	Best	**940,881**	**941,899**	**940,139**	**939,562**	**940,924**	**940,407**	**939,260**	**940,806**	**939,921**	**940,600**
			UB	940,881	941,899	940,139	939,562	940,924	940,407	939,260	940,806	939,921	940,600
	25	$[1, 10^2]$	Best	**3766**	**3758**	**3762**	**3766**	**3759**	**3766**	**3764**	**3759**	**3760**	**3762**
			UB	3766	3758	3762	3766	3759	3766	3764	3759	3760	3762
		$[1, 10^3]$	Best	**37,655**	**37,585**	**37,631**	**37,660**	**37,595**	**37,658**	**37,645**	**37,598**	**37,603**	**37,622**
			UB	37,655	37,585	37,631	37,660	37,595	37,658	37,645	37,598	37,603	37,622
		$[1, 10^4]$	Best	**376,546**	**375,852**	**376,315**	**376,605**	**375,956**	**376,584**	**376,449**	**375,988**	**376,037**	**376,224**
			UB	376,546	375,852	376,315	376,605	375,956	376,584	376,449	375,988	376,037	376,224

instance, we record the best found objective function value (labeled as "Best") as well as the best found upper bound value (labeled as "UB"). Bold entries indicate optimal values.

References

Ahuja, R. K., & Orlin, J. B. (2001). Inverse optimization. *Operations Research*, *49*, 771–783.

Alvim, A. C. F., Ribeiro, C. C., Glover, F., & Aloise, D. J. (2004). A hybrid improvement heuristic for the one-dimensional bin packing problem. *Journal of Heuristics*, *10*, 205–229.

Azar, Y., & Epstein, L. (1998). On-line machine covering. *Journal of Scheduling*, *1*, 67–77.

Brucker, P., & Shakhlevich, N. V. (2009). Inverse scheduling with maximum lateness objective. *Journal of Scheduling*, *12*, 475–488.

Brucker, P., & Shakhlevich, N. V. (2011). Inverse scheduling: Two-machine flow-shop problem. *Journal of Scheduling*, *14*, 239–256.

Cai, S.-Y. (2007). Semi-online machine covering. *Asia-Pacific Journal of Operational Research*, *24*, 373–382.

Csirik, J., Kellerer, H., & Woeginger, G. (1992). The exact LPT-bound for maximizing the minimum completion time. *Operations Research Letters*, *11*, 281–287.

Dell'Amico, M., & Martello, S. (1995). Optimal scheduling of tasks on identical parallel processors. *ORSA Journal on Computing*, *7*, 191–200.

Deuermeyer, B. L., Friesen, D. K., & Langston, M. A. (1982). Scheduling to maximize the minimum processor finish time in a multiprocessor system. *SIAM Journal on Algebraic and Discrete Methods*, *3*, 190–196.

Ebenlendr, T., Noga, J., Sgall, J., & Woeginger, G. (2006). A note on semi-online machine covering. In T. Erlebach & G. Persiano (Eds.), *Approximation and online algorithms. Lecture Notes in Computer Science* (Vol. 3879, pp. 110–118). Berlin: Springer.

Epstein, L., Levin, A., & van Stee, R. (2011). Max-min online allocations with a reordering buffer. *SIAM Journal on Discrete Mathematics*, *25*, 1230–1250.

França, P. M., Gendreau, M., Laporte, G., & Müller, F. M. (1994). A composite heuristic for the identical parallel machine scheduling problem with minimum makespan objective. *Computers & Operations Research*, *21*, 205–210.

Frangioni, A., Necciari, E., & Scutellà, M. G. (2004). A multi-exchange neighborhood for minimum makespan parallel machine scheduling problems. *Journal of Combinatorial Optimization*, *8*, 195–220.

Friesen, D. K., & Deuermeyer, B. L. (1981). Analysis of greedy solutions for a replacement part sequencing problem. *Mathematics of Operations Research*, *6*, 74–87.

Graham, R. L. (1969). Bounds on multiprocessing timing anomalies. *SIAM Journal on Applied Mathematics*, *17*, 416–429.

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

Haouari, M., & Jemmali, M. (2008). Maximizing the minimum completion time on parallel machines. *4OR: A Quarterly Journal of Operations Research*, *6*, 375–392.

He, Y., & Tan, Z. Y. (2002). Ordinal on-line scheduling for maximizing the minimum machine completion time. *Journal of Combinatorial Optimization*, *6*, 199–206.

Heuberger, C. (2004). Inverse combinatorial optimization: A survey on problems, methods, and results. *Journal of Combinatorial Optimization*, *8*, 329–361.

Koulamas, C. (2005). Inverse scheduling with controllable job parameters. *International Journal of Services and Operations Management*, *1*, 35–43.

Labbé, M., Laporte, G., & Martello, S. (1995). An exact algorithm for the dual bin packing problem. *Operations Research Letters*, *17*, 9–18.

Luo, R.-Z., & Sun, S.-J. (2005). Semi on-line scheduling problem for maximizing the minimum machine completion time on *m* identical machines. *Journal of Shanghai University*, *9*, 99–102.

Peeters, M., & Degraeve, Z. (2006). Branch-and-price algorithms for the dual bin packing and maximum cardinality bin packing problem. *European Journal of Operational Research*, *170*, 416–439.

Tan, Z., & Wu, Y. (2007). Optimal semi-online algorithms for machine covering. *Theoretical Computer Science*, *372*, 69–80.

Walter, R. (2013). Comparing the minimum completion times of two longest-first scheduling-heuristics. *Central European Journal of Operations Research*, *21*, 125–139.

Walter, R., & Lawrinenko, A. (2014). Effective solution space limitation for the identical parallel machine scheduling problem. Working Paper. http://s.fhg.de/WrkngPprRW.

Woeginger, G. J. (1997). A polynomial-time approximation scheme for maximizing the minimum machine completion time. *Operations Research Letters*, *20*, 149–154.