Exact makespan minimization of unrelated parallel machines

OJMO

Open Journal of Mathematical Optimization

Edvin Åblad, Ann-Brith Strömberg & Domenico Spensieri

Exact makespan minimization of unrelated parallel machines

Volume 2 (2021), article no. 2 (15 pages)

<http://ojmo.centre-mersenne.org/item/OJMO_2021__2__A2_0>

Article submitted on April 2, 2020, revised on April 1, 2021,
accepted on April 25, 2021.

Open Journal of Mathematical Optimization is member of the
Centre Mersenne for Open Scientific Publishing
www.centre-mersenne.org

Exact makespan minimization of unrelated parallel machines

Edvin Åblad
Fraunhofer-Chalmers Research Centre and Chalmers University of Technology, Gothenburg, Sweden
edvin.ablad@fcc.chalmers.se

Ann-Brith Strömberg
Chalmers University of Technology and University of Gothenburg, Gothenburg, Sweden
anstr@chalmers.se

Domenico Spensieri
Fraunhofer-Chalmers Research Centre and Chalmers University of Technology, Gothenburg, Sweden
domenico.spensieri@fcc.chalmers.se

Abstract

We study methods for the exact solution of the unrelated parallel machine problem with makespan minimization, generally denoted as $R||C_{\max}$. Our original application arises from the automotive assembly process where tasks needs to be distributed among several robots. This involves the solutions of several $R||C_{\max}$ instances, which proved hard for a MILP solver since the makespan objective induces weak LP relaxation bounds. To improve these bounds and to enable the solution of larger instances, we propose a branch–and–bound method based on a Lagrangian relaxation of the assignment constraints. For this relaxation we derive a criterion for variable fixing and prove the zero duality gap property for the case of two parallel machines. Our computational studies indicate that the proposed algorithm is competitive with state-of-the-art methods on different types of instances. Moreover, the impact of each proposed feature is analysed.

Digital Object Identifier 10.5802/ojmo.4

Keywords unrelated parallel machine problem, makespan, variable fixing, binary knapsack, Lagrangian relaxation.

1 Introduction

We study the *unrelated parallel machine problem* (e.g., [29, 31]), which is to assign n tasks to m machines, where the processing time of an assignment depends on both task and machine, and the goal is to minimize the makespan (i.e., the finish time of the last machine); see [23]. By the three-field notation from scheduling theory, cf. [20], this problem is denoted $R||C_{\max}$, where R denotes that the processing times are unrelated, $||$ denotes that there are no additional constraints on the tasks, and $C_{\max}$ denotes that the makespan is to be minimized. The $R||C_{\max}$ problem is a classical problem in the field of scheduling and it can be utilized when scheduling tasks on, e.g., production lines or multiprocess computers.

A previous study, considering an application from the automotive manufacturing industry, concerned the makespan minimization of several industrial robots completing a set of welding tasks without entering each other's workspaces; it also included sequences of tasks, robot routing, and path planning; see [1, 3]. The requirement of not intersecting each other's workspaces while working on the robots' respective assigned tasks resulted in an extension of the $R||C_{\max}$ with a few set packing side-constraints, the extension was iteratively solved in order to provide candidate robot–task assignments. Each of these candidate assignments were later analysed in order to find improved sequences of the tasks as well as the corresponding motions of the robots.

The main reasons to perform the study resulting in this article is that our suggested model proved hard to solve with a general *mixed–integer linear programming* (MILP) solver; see [3], and that the issue of long computing times remained when relaxing our model to the $R||C_{\max}$. Thus, this study aims to construct exact methods for the $R||C_{\max}$, preferably such that can be generalized to our extension of the $R||C_{\max}$, and using considerably less computing time as compared to a general MILP solver applied to the original $R||C_{\max}$ formulation. For the sake of generality, we consider general $R||C_{\max}$ instances; see, e.g., [29, 32, 42]. The main contributions of our study are a new variable fixing criterion and, for $m = 2$, a proof of no duality gap.

Related work on $R||C_{\max}$ is reviewed in Section 2. Inspired by the strongest Lagrangian relaxation suggested in [29], (i) the knapsack structure of the subproblems is utilized to derive a variable fixing criterion, (ii) a deflected

subgradient method is used to maximize the dual function, (iii) the duality gap is investigated, and (iv) a branch–and–bound rule is suggested in Section 3. A local search heuristic is presented in Section 4. Some MILP approaches, test instances, and results are presented in Section 5 and conclusions are presented in Section 6.

We apply the Lagrangian relaxation suggested in [29], however, with a variable fixing criterion that differs from their implementation. Moreover, in [29] the focus is on the branch–and–bound root node, in which several heuristics are employed and the strong Lagrangian dual function is maximized, whereas in the child nodes only weaker Lagrangian relaxations are utilized. We, however, apply local search heuristics and spend most of the computational effort on maximizing the strong dual function using a deflected subgradient method in every branch–and–bound node.

2 Related work

In 1975, Sandi [37] studied a mixture of $R||C_{\max}$ and $P||C_{\max}$, where P denotes that the processing times are *identical* over the machines. In 1976, Horowitz and Sahni [23] were among the first to suggest an exact solution method for the $R||C_{\max}$ based on dynamic programming (however, exponential in nature) and in 1990, Lenstra et al. [27] showed that no polynomial $\frac{3}{2}$-approximation algorithm for the $R||C_{\max}$ can exist unless P=NP, and that the $R||C_{\max}$ is NP-hard even for $m = 2$. Hence, approximation algorithms are often suggested for the $R||C_{\max}$, e.g., a classical 2-approximation algorithm based on rounding the *linear programming* (LP) solution; see [27] and [40, Ch. 17]. For fixed values of m there exist fully polynomial-time $(1 + \varepsilon)$-approximation algorithms; e.g., in 2001, Jansen and Porkolab [25] presented a running time of $n(m/\varepsilon)^{\mathcal{O}(m)}$. If m and n are not too large (e.g., $m \leq 20$, $n \leq 200$), exact algorithms are also applicable. In 1991, Velde et al. [39] suggested a branch–and–bound algorithm based on Lagrangian relaxation. In 1997, Martello et al. [29] improved the branch–and–bound algorithm by using several Lagrangian relaxations and so-called residual costs [14]. In 2002, Mokotoff and Chrétienne [32] suggested a cutting plane method; by assuming integer valued processing times, the value of the makespan is iteratively increased until a feasible solution exists. A branch–and–price algorithm was suggested in 2007 by Wotzlaw [42] along with a survey of 2-approximation and heuristic algorithms comprising, e.g., the one described in [16].

Local search heuristics are often applied in order to find good feasible solutions to the $R||C_{\max}$. In 1994, Glass et al. [19] defined neighbourhoods as *reassign a task to another machine* and *interchange two tasks between two machines*. Wotzlaw [42] emphasized the importance of applying local search techniques in a branch–and–price implementation to allow early pruning. In 2004, Frangioni et al. [15] suggested two larger neighbourhoods, a *cyclic exchange* and a *path exchange* of tasks, and showed that the problem of verifying that these neighbourhoods do not contain an incumbent solution is NP-complete; similar but smaller neighbourhoods are employed in our suggested methodology (see Section 4). In 2010, Fanjul-Peryro and Ruiz [12] introduced an *iterated greedy heuristic* based on a destruction–reconstruction methodology and the local search heuristics suggested in [19]; they compared their result with a general MILP solver and found that, given a short time limit, their heuristic generally outperformed the MILP solver. In 2005, Ghirardi and Potts [18] suggested a so-called *recovery beam* heuristic for the $R||C_{\max}$; it uses a so-called *truncated branch–and–bound* algorithm, where the lower and upper bounds in each node are computed as in [39] and [29], respectively.

The $R||C_{\max}$ is a common relaxation of scheduling problems including additional constraints; hence, the $R||C_{\max}$ yields a lower bound for such problems. In 2014, such relaxations were used by Borba and Ritt [9] and Vilá and Pereira [41]: both studied precedence constraints among tasks and used lower bounds for the $R||C_{\max}$ in their respective branch–and–bound algorithms. In 2017, Åblad et al. [3] solved the $R||C_{\max}$ with additional set packing constraints using a branch–and–cut solver.

3 A Lagrangian relaxation of the $R||C_{\max}$

In this section we present a MILP model for the $R||C_{\max}$ and the corresponding Lagrangian relaxation of the task-assignment constraints. We then exploit the lower bound of this relaxation and the structure in the resulting subproblems to fix the values of variables which result in a reduction of the total computation time.

A MILP model for the $R||C_{\max}$ (cf. [39]) is the following: let $\mathcal{I} := \{1, \dots, m\}$ denote the set of machines, $\mathcal{J} := \{1, \dots, n\}$ the set of tasks, and $p_{ij} \in \mathbb{Z}_+$ the processing time of task $j \in \mathcal{J}$ on machine $i \in \mathcal{I}$. Define

$$x_{ij} = \begin{cases} 1, & \text{if machine } i \text{ performs task } j, \\ 0, & \text{otherwise,} \end{cases} \qquad i \in \mathcal{I},\ j \in \mathcal{J}. \tag{1}$$

Introducing the set

$$X(z) := \left\{ \boldsymbol{x} \in \mathbb{B}^{mn} \;\middle|\; \sum_{j \in \mathcal{J}} p_{ij} x_{ij} \leq z,\, i \in \mathcal{I} \right\}, \qquad z \in \mathbb{R}, \tag{2a}$$

in which each machine must respect the makespan value z, and the set

$$X_{\mathtt{sa}} := \left\{ \boldsymbol{x} \in \mathbb{B}^{mn} \;\middle|\; \sum_{i \in \mathcal{I}} x_{ij} = 1,\, j \in \mathcal{J} \right\}, \tag{2b}$$

in which each task is assigned to exactly one machine (semi-assignments, sa), the $R||C_{\max}$ is formulated as

$$\underset{\boldsymbol{x} \in \mathbb{B}^{mn},\, z \in \mathbb{R}}{\text{minimize}} \; \{ z \mid \boldsymbol{x} \in X(z) \cap X_{\mathtt{sa}} \}. \tag{3}$$

We also let $S := \{(\boldsymbol{x}, z) \in \mathbb{B}^{mn} \times \mathbb{R} \mid \boldsymbol{x} \in X(z) \cap X_{\mathtt{sa}}\}$ denote the feasible set of (3).

A Lagrangian relaxation of the constraints in (2a) (as suggested in [39]) yields a subproblem that is polynomially solvable, while relaxing the constraints in (2b) (as suggested by Martello et al. [29]) leads to a subproblem involving m binary knapsack problems, which are NP-hard. Moreover, as shown in [29], it leads to a stronger lower bound. We consider the Lagrangian relaxation of the constraints in (2b), which results in the expression of the Lagrangian dual function $h : \mathbb{R}^n \mapsto \mathbb{R}$ as

$$h(\boldsymbol{u}) := \sum_{j \in \mathcal{J}} u_j + \underset{\underline{z} \leq z \leq \overline{z}}{\text{minimum}} \left(z - f(z, \boldsymbol{u}) \right), \tag{4a}$$

$$f(z, \boldsymbol{u}) := \sum_{i \in \mathcal{I}} \underset{\boldsymbol{x}_{i\cdot} \in X_i(z)}{\text{maximum}} \sum_{j \in \mathcal{J}} u_j x_{ij}, \tag{4b}$$

where $X_i(z) := \{\boldsymbol{x}_{i\cdot} \in \mathbb{B}^n \mid \sum_{j \in \mathcal{J}} p_{ij} x_{ij} \leq z\}$ (i.e., $X(z) = X_1(z) \times \cdots \times X_m(z)$), $\boldsymbol{u} \in \mathbb{R}^n$ is the Lagrangian multiplier vector, and $\underline{z}$ and $\overline{z}$ are the current best known lower and upper bounds on the optimal objective value z^* of (3), respectively. The optimal value of the Lagrangian dual is found by maximizing $h(\boldsymbol{u})$ over $\boldsymbol{u} \in \mathbb{R}^n$.

Note that the constraints $\underline{z} \leq z \leq \overline{z}$ are redundant in (3), however, tight bounds can improve the lower bound $h(\boldsymbol{u})$. For example, if $h(\boldsymbol{u}) > \underline{z}$ for some $\boldsymbol{u} \in \mathbb{R}^n$ then $\underline{z} := \lceil h(\boldsymbol{u}) \rceil$, where the ceiling operation is valid due to the integrality of p_{ij}. Hence, the dual function (4a) gets redefined whenever tighter bounds $\underline{z}$ and $\overline{z}$ are found, cf. Section 3.3. The lower bound $\underline{z}$ is initialized using the LP relaxation of (3) and updated using $h(\boldsymbol{u})$ and branch–and–bound. The upper bound is provided and updated by our heuristics; see Section 4.

For fixed values of z and $\boldsymbol{u}$ the problem (4b) amounts to solving m binary knapsack problems. In [29] these knapsack problems are solved by a branch–and–bound algorithm; we, however, employ the classical dynamic programming approach presented in [30, p. 38]. Although a more sophisticated dynamic programming approach can be applied (see, e.g., [34, 28]), our preliminary tests indicated that the classical implementation suffices for our instances, where twenty years of development in computer memory size and speed might be the enabler. This classical approach also provides a direct computation of bounds to be used in the *variable fixing* procedure specified next, whereas a more sophisticated dynamic programming approach might not provide the computation of these bounds and is thus left for future studies. The *variable fixing* is one of our main contributions. Moreover, in Section 3.3 we emphasize the strength of the lower bound (4a) and describe how to maximize it in Section 3.2.

3.1 Variable fixing using the subproblem solution

Given a lower bound for a problem with a binary variable restricted to one, if this bound exceeds a known upper bound for the unrestricted problem then the variable can be set to zero; this is called *variable fixing*. This lower bound is often deduced by the *reduced costs* of the LP relaxation of the problem, being a special case of *residual cost*, cf. [14]. However, we will compute $h(\boldsymbol{u} \mid x_{\bar{i}\bar{j}} = 1)$ and $f(z, \boldsymbol{u} \mid x_{\bar{i}\bar{j}} = 1)$, i.e., restricting $x_{\bar{i}\bar{j}}$ to one by replacing the set $X_{\bar{i}}(z)$ in (4b) by $X_{\bar{i}}(z) \cap \{\boldsymbol{x}_{\bar{i}\cdot} \in \mathbb{B}^n \mid x_{\bar{i}\bar{j}} = 1\}$ for some $\bar{i} \in \mathcal{I}$, $\bar{j} \in \mathcal{J}$. Moreover, the feasible set $X_{\mathtt{sa}}$ is respected by also restricting $x_{i\bar{j}} = 0$, $i \in \mathcal{I} \setminus \{\bar{i}\}$, i.e., the task $\bar{j}$ is assigned to only machine $\bar{i}$. Thus, if $h(\boldsymbol{u} \mid x_{\bar{i}\bar{j}} = 1) \geq \overline{z}$, we may fix $x_{\bar{i}\bar{j}} = 0$, since the constraint $x_{\bar{i}\bar{j}} = 1$ does not allow for an improved upper bound.

Before describing the computation of $h(\boldsymbol{u} \mid x_{\bar{i}\bar{j}} = 1)$ we present the classical dynamic programming approach ([30, p. 38]) for solving binary knapsack problems. For each $i \in \mathcal{I}$ a value matrix $\boldsymbol{V}^i \in \mathbb{R}_+^{n \times (\bar{z}+1)}$ is introduced and to find an optimal solution to the i-th binary knapsack problem, the items are recursively inserted as

$$V^i_{jw} := \begin{cases} V^i_{(j-1)w}, & w \in \{0, \ldots, p_{ij} - 1\}, \\ \max\{V^i_{(j-1)w}, V^i_{(j-1)(w-p_{ij})} + u_j\}, & w \in \{p_{ij}, \ldots, \bar{z}\}, \end{cases} \qquad j \in \mathcal{J}, \tag{5}$$

where $\boldsymbol{V}^i_{0\cdot} := \mathbf{0}$. As a result, for any $z \in \{0,\dots,\bar{z}\}$, V^i_{jz} is the optimal value of the knapsack of size z using items $\{1,\dots,j\}$; hence, $f(z,\boldsymbol{u}) = \sum_{i\in\mathcal{I}} V^i_{nz}$. Thus, these value matrices enable a direct evaluation of the Lagrangian dual function $h(\boldsymbol{u})$.

A variable fixing criterion is available as a by-product of the value matrices $V^i, i \in \mathcal{I}$. This is accomplished by bounding $f(z,\boldsymbol{u} \mid x_{\bar{\imath}\bar{\jmath}} = 1)$ from above, by having $x_{i\bar{\jmath}}$ unrestricted for $i \in \mathcal{I}$ and ensuring that a duplicate item of $x_{\bar{\imath}\bar{\jmath}}$ is inserted in the $\bar{\imath}$-th knapsack, resulting in

$$f(z,\boldsymbol{u} \mid x_{\bar{\imath}\bar{\jmath}} = 1) \le u_{\bar{\jmath}} + V^{\bar{\imath}}_{n(z-p_{\bar{\imath}\bar{\jmath}})} + \sum_{i\in\mathcal{I}\setminus\{\bar{\imath}\}} V^i_{nz}. \tag{6}$$

By using the inequality in (6) a lower bound on $h(\boldsymbol{u} \mid x_{\bar{\imath}\bar{\jmath}} = 1)$ is available at low computational cost when the knapsack subproblems in (4b), are solved using the dynamic programming approach. We next show how the value matrices can be utilized to compute $f(z,\boldsymbol{u} \mid x_{\bar{\imath}\bar{\jmath}} = 1)$ and $h(\boldsymbol{u} \mid x_{\bar{\imath}\bar{\jmath}} = 1)$ without having to resolve the subproblem for each $\bar{\imath} \in \mathcal{I}, \bar{\jmath} \in \mathcal{J}$.

The idea is to also compute the entities $\widetilde{V}^i_{jz}$ by an analogous recursion as for V^i_{jz}, where the items are inserted in reversed order: $j = n,\dots,1$. Hence, for each $i \in \mathcal{I}$, $\widetilde{V}^i_{jz}$ is the optimal value of the i-th knapsack of size z using items $\{j,\dots,n\}$. According to Lemma 1, the value of $f(z,\boldsymbol{u} \mid x_{\bar{\imath}\bar{\jmath}} = 1)$ can be computed by combining V^i_{jz} and $\widetilde{V}^i_{jz}$. As a result, the inequality $h(\boldsymbol{u} \mid x_{\bar{\imath}\bar{\jmath}} = 1) \ge \overline{z}$ can be efficiently checked; if it holds the corresponding variable $x_{\bar{\imath}\bar{\jmath}}$ can be fixed to zero.

▶ **Lemma 1.** *For $i \in \mathcal{I}, \bar{\jmath} \in \mathcal{J}$, $W^i_{\bar{\jmath}z} := \text{maximum}_{v\in\{0,\dots,z\}}\{V^i_{(\bar{\jmath}-1)v} + \widetilde{V}^i_{(\bar{\jmath}+1)(z-v)}\}$ is the optimal value of the knapsack i with capacity z and the items $\mathcal{J}\setminus\{\bar{\jmath}\}$. Moreover, $f(z,\boldsymbol{u} \mid x_{\bar{\imath}\bar{\jmath}} = 1) = u_{\bar{\jmath}} + W^{\bar{\imath}}_{\bar{\jmath}(z-p_{\bar{\imath}\bar{\jmath}})} + \sum_{i\in\mathcal{I}\setminus\{\bar{\imath}\}} W^i_{\bar{\jmath}z}$.*

Proof. For each $i \in \mathcal{I}$, consider the binary knapsack formulation where the $\bar{\jmath}$-th item is not present:

$$\max_{\boldsymbol{x}_{i\cdot}\in\mathbb{B}^n} \left\{ \sum_{j\in\mathcal{J}\setminus\{\bar{\jmath}\}} u_j x_{ij} \;\middle|\; \sum_{j\in\mathcal{J}\setminus\{\bar{\jmath}\}} p_{ij} x_{ij} \le z \right\}. \tag{7}$$

First, consider the cases $\bar{\jmath} = 1$ and $\bar{\jmath} = n$, the corresponding optimal values of (7) are $\widetilde{V}^i_{(\bar{\jmath}+1)z}$ and $V^i_{(\bar{\jmath}-1)z}$, respectively. Moreover, $V^i_{0z} = \widetilde{V}^i_{(n+1)z} = 0$, hence $W^i_{\bar{\jmath}z}$ is the optimal value of (7). Second, assume $\bar{\jmath} \in \mathcal{J}\setminus\{1,n\}$ and let $\bar{\boldsymbol{x}}_{i\cdot}$ be feasible in (7) and let $\bar{v} := \sum_{j=1}^{\bar{\jmath}-1} p_{ij}\bar{x}_{ij}$. Then $(\bar{\boldsymbol{x}}_{i\cdot}, \bar{v})$ is feasible in the following formulation:

$$\max_{v\in\{0,\dots,z\}} \left\{ \max_{\boldsymbol{x}_{i\cdot}\in\mathbb{B}^n} \left\{ \sum_{j\in\mathcal{J}\setminus\{\bar{\jmath}\}} u_j x_{ij} \;\middle|\; \sum_{j=1}^{\bar{\jmath}-1} p_{ij}x_{ij} \le v,\ \sum_{j=\bar{\jmath}+1}^{n} p_{ij}x_{ij} \le z - v \right\} \right\}. \tag{8}$$

Similarly, let $(\widehat{\boldsymbol{x}}_{i\cdot}, \widehat{v})$ be feasible in (8). By aggregating the constraints in (8) it follows that $\widehat{\boldsymbol{x}}_{i\cdot}$ is feasible in (7). Thus, the formulations (7) and (8) are equivalent. For a fixed value $v = \widetilde{v}$, the inner maximization problem in (8) decomposes into two knapsack problems using only the items before $\bar{\jmath}$ and after $\bar{\jmath}$, respectively. The optimal values of these two knapsack problems are thus $V^i_{(\bar{\jmath}-1)\widetilde{v}}$ and $\widetilde{V}^i_{(\bar{\jmath}+1)(z-\widetilde{v})}$, respectively. Hence, by maximizing the decomposed problem (8) for every $v \in \{0,\dots,z\}$, the value of the knapsack without item $\bar{\jmath}$ can be computed by the proposed formula for $W^i_{\bar{\jmath}z}$. By the definition of $W^i_{\bar{\jmath}z}$ for $i \in \mathcal{I}, \bar{\jmath} \in \mathcal{J}$, and size z, the value of knapsack $\bar{\imath} \in \mathcal{I}$ restricted by $x_{\bar{\imath}\bar{\jmath}} = 1$ (i.e., with feasible set $X_{\bar{\imath}}(z) \cap \{\boldsymbol{x}_{\bar{\imath}\cdot} \in \mathbb{B}^n \mid x_{\bar{\imath}\bar{\jmath}} = 1\}$) equals $W^{\bar{\imath}}_{\bar{\jmath}(z-p_{\bar{\imath}\bar{\jmath}})} + u_{\bar{\jmath}}$ while the values of the knapsacks $i \in \mathcal{I}\setminus\{\imath\}$ that are restricted by $x_{i\bar{\jmath}} = 0$ (i.e., with feasible sets $X_i(z) \cap \{\boldsymbol{x}_{i\cdot} \in \mathbb{B}^n \mid x_{i\bar{\jmath}} = 0\}$) equal $W^i_{\bar{\jmath}z}$. The sum of these values then yields the proposed formula for $f(z,\boldsymbol{u} \mid x_{\bar{\imath}\bar{\jmath}} = 1)$. ◀

▶ Remark 2. In contrast to the by-product (6), using Lemma 1 for variable fixing requires computing W^i_{jz} for $z \in \{\max\{0,\underline{z} - p_{ij}\},\dots,\overline{z} - p_{ij}\} \cup \{\underline{z},\dots,\overline{z}\}, i \in \mathcal{I}, j \in \mathcal{J}$. However, the variable fixing is done once in each branch–and–bound node, whereas finding suitable values of $\boldsymbol{u}$ (see Section 3.2) typically requires solving many subproblem instances. Our preliminary results indicated that less than 5% of the overall computation time was spent on the variable fixing.

▶ Remark 3. Our idea of using V^i_{jz} and $\widetilde{V}^i_{jz}$ originates from the bidirectional search approach suggested in [24] for the computation of residual costs of a shortest path problem with resource constraints. Moreover, in [36] V^i_{jz}, $\widetilde{V}^i_{jz}$, and the set $X_{\mathtt{sa}}$ are also used to assign a job exactly once and to compute the similar residual costs for the generalized assignment problem. However, our approach considers minimizing the makespan.

We found that the variable fixing is highly useful for reducing the computation time. Hence, we suggest a further variable fixing similar to the *dominance criterion* suggested in [29]. Given a machine $\bar{\imath} \in \mathcal{I}$ and two tasks $j, \bar{\jmath} \in \mathcal{J}$, we attempt to assign both tasks to the machine and use a bound similar to that in (6), i.e.,

$$f(z, \boldsymbol{u} \mid x_{\bar{\imath}j} = x_{\bar{\imath}\bar{\jmath}} = 1) \leq u_j + u_{\bar{\jmath}} + V^{\bar{\imath}}_{n(z-p_{\bar{\imath}j}-p_{\bar{\imath}\bar{\jmath}})} + \sum_{i \in \mathcal{I} \setminus \{\bar{\imath}\}} V^i_{nz}. \tag{9}$$

If the bound (9) does not allow an incumbent solution (i.e., if it implies that $h(\boldsymbol{u} \mid x_{\bar{\imath}j} = x_{\bar{\imath}\bar{\jmath}} = 1) \geq \overline{z}$) then the inequality $x_{\bar{\imath}j} + x_{\bar{\imath}\bar{\jmath}} \leq 1$ can be introduced and the following *dominance criterion* can be checked. Let $\mathcal{I}_j \subset \mathcal{I}$ denote the subset of machines i available for task j, i.e., those that have not been removed by variable fixing or by branching. If the inequalities $p_{\bar{\imath}j} \leq p_{\bar{\imath}\bar{\jmath}}$ and $p_{ij} \geq p_{i\bar{\jmath}}$ hold for all $i \in \mathcal{I}_j \setminus \{\bar{\imath}\}$, it follows that any solution satisfying $x_{\bar{\imath}\bar{\jmath}} = 1$ and $x_{\bar{\imath}j} = 0$ is dominated by a solution satisfying $x_{\bar{\imath}\bar{\jmath}} = 0$ and $x_{\bar{\imath}j} = 1$, hence the variable $x_{\bar{\imath}\bar{\jmath}}$ can be fixed to zero; cf. [29]. We also apply the *dominance criterion* suggested in [29], where the inequality $x_{\bar{\imath}j} + x_{\bar{\imath}\bar{\jmath}} \leq 1$ is instead ensured by the decisions in the branch-and-bound tree. Note that computing $f(z, \boldsymbol{u} \mid x_{\bar{\imath}j} = x_{\bar{\imath}\bar{\jmath}} = 1)$ as in Lemma 1 for each combination of $\bar{\imath}$, j, and $\bar{\jmath}$, would be computationally too expensive.

3.2 Maximizing the Lagrangian dual

In order to find the best lower bound $\underline{z}$ on z^*, the Lagrangian dual function $h : \mathbb{R}^n \mapsto \mathbb{R}$, which is non-smooth and concave, should be maximized. Hence, we need to find $h^* := \max_{\boldsymbol{u} \in \mathbb{R}^n} h(\boldsymbol{u})$. This maximization is often done using subgradient optimization methods; e.g., Martello et al. [29] used it for the $R||C_{\max}$. However, since evaluating $h(\boldsymbol{u})$ comprises solving m binary knapsack problems (which are NP-hard) we need to limit the number of such evaluations. Note that, by the non-negativity of p_{ij}, the search can be restricted to $\boldsymbol{u} \geq \boldsymbol{0}$, since $u_j < 0$ is dominated by $u_j = 0$ in (4a).

An issue with the classical subgradient algorithm is that it often stalls due to zigzagging; cf. [6]. We therefore employ the *modified deflected subgradient* (MDS) method by Belgacem and Amir [7], which combines, the *modified gradient technique* (MGT) by Camerini et al. [10] and the *average direction strategy* (ADS) by Sherali and Ulular [38]. The MDS method reduces the zigzagging behaviour by determining the step direction as a weighted average of a current subgradient and the previous step direction.

Letting $\boldsymbol{s}^k$ be a subgradient of h at $\boldsymbol{u}^k$ and $\boldsymbol{d}^k$ be the step direction, and defining $[\boldsymbol{x}]_+ := \max\{0, \boldsymbol{x}\}$, we employ the formulas

$$\boldsymbol{d}^k := \boldsymbol{s}^k + \Gamma^k_{\texttt{MDS}} \boldsymbol{d}^{k-1}, \qquad \Gamma^k_{\texttt{MDS}} := (1 - \alpha_k)\Gamma^k_{\texttt{MGT}} + \alpha_k \Gamma^k_{\texttt{ADS}}, \tag{10a}$$

$$\Gamma^k_{\texttt{MGT}} := \left[-\eta_k \frac{(\boldsymbol{s}^k)^\top \boldsymbol{d}^{k-1}}{\|\boldsymbol{d}^{k-1}\|^2} \right]_+, \qquad \Gamma^k_{\texttt{ADS}} := \frac{\|\boldsymbol{s}^k\|}{\|\boldsymbol{d}^{k-1}\|}. \tag{10b}$$

Following the suggestions in [7], we let $\eta_k := \frac{1}{2-\alpha_k}$ and $\alpha_k := \left[-\frac{(\boldsymbol{s}^k)^\top \boldsymbol{d}^{k-1}}{\|\boldsymbol{s}^k\| \|\boldsymbol{d}^{k-1}\|} \right]_+$.

To determine the step lengths, we follow the classical rule given by Polyak [35] and let the step length parameter $\delta_k > 0$ decrease as suggested by Held et al. [22], i.e.,

$$\boldsymbol{u}^{k+1} := [\boldsymbol{u}^k + t_k \boldsymbol{d}^k]_+, \qquad t_k := \delta_k \frac{\overline{z}_k - h(\boldsymbol{u}^k)}{\|\boldsymbol{d}^k\|^2}, \qquad k = 0, 1, \ldots \tag{11}$$

where $\delta_0 := \frac{1}{2}$, and if no improvement has been made to $h(\boldsymbol{u}^k)$ over the last twenty iterations then $\delta_{k+1} := \frac{\delta_k}{2}$; otherwise $\delta_{k+1} := \delta_k$. The algorithm (11) is terminated when $\delta_k < \underline{\delta}$, where we use $\underline{\delta} = 10^{-3}$, which in our preliminary tests showed to be a reasonable choice.

3.3 Duality gap

The strength of the Lagrangian relaxation as defined in (4a) was studied in [29], where the corresponding lower bound $h^* \leq z^*$ was found to be stronger than the LP bound. In Theorem 4, we strengthen this result further and show that for $m = 2$, either $h^* > \underline{z}$ or $h^* = z^*$ hold. Hence, the best known lower bound $\underline{z}$ can be increased until $\underline{z} = z^*$ holds. For convenience, we first restate the Lagrangian dual function (4a) in its original form where the structure of the set $X(z)$ has not yet been utilized:

$$h(\boldsymbol{u}) = \sum_{j \in \mathcal{J}} u_j + \underset{(\boldsymbol{x}, z) \in \mathbb{B}^{mn} \times [\underline{z}, \overline{z}]}{\text{minimum}} \left\{ z - \sum_{i \in \mathcal{I}} \sum_{j \in \mathcal{J}} u_j x_{ij} \;\middle|\; \boldsymbol{x} \in X(z) \right\}. \tag{12}$$

▶ **Theorem 4.** *Let $m = 2$. If $h^* \leq \underline{z}$ then $h^* = z^*$.*

Proof. By the relation between the Dantzig–Wolfe reformulation ([11, Ch. 8.2]) and Lagrangian relaxation, it is holds that

$$h^* = \underset{\boldsymbol{\alpha} \geq \mathbf{0}}{\text{minimum}} \quad \sum_{k \in \mathcal{K}} \alpha_k z^k, \tag{13a}$$

$$\text{such that} \quad \sum_{k \in \mathcal{K}} \alpha_k \sum_{i \in \mathcal{I}} x_{ij}^k = 1, \quad j \in \mathcal{J}, \tag{13b}$$

$$\sum_{k \in \mathcal{K}} \alpha_k = 1, \tag{13c}$$

where $(\boldsymbol{x}^k, z^k)$, $k \in \mathcal{K}$, denote the extreme points of the polytope $\operatorname{conv}\{(\boldsymbol{x}, z) \in \mathbb{B}^{mn} \times [\underline{z}, \overline{z}] \mid \boldsymbol{x} \in X(z)\}$; cf. (12). Let $\boldsymbol{\alpha}^*$ be an optimal solution to the Dantzig–Wolfe master problem (13) and define the set $\bar{\mathcal{K}} := \{k \in \mathcal{K} \mid \alpha_k^* > 0\}$. It holds that $z^k = \underline{z}$ for all $k \in \bar{\mathcal{K}}$, since otherwise the strict inequality $h^* > \underline{z}$ must hold. Moreover, let $\mathcal{J}_l^k := \{j \in \mathcal{J} \mid \sum_{i \in \mathcal{I}} x_{ij}^k = l\}, k \in \bar{\mathcal{K}}$, and $\mathcal{K}_l^j := \{k \in \bar{\mathcal{K}} \mid \sum_{i \in \mathcal{I}} x_{ij}^k = l\}, j \in \mathcal{J}, l \in \{0, 1, 2\}$. From (13b) and (13c) it then follows that

$$2 \sum_{k \in \mathcal{K}_2^j} \alpha_k^* + \sum_{k \in \mathcal{K}_1^j} \alpha_k^* - \sum_{k \in \bar{\mathcal{K}}} \alpha_k^* = 0 \quad \Longrightarrow \quad \sum_{k \in \mathcal{K}_2^j} \alpha_k^* = \sum_{k \in \mathcal{K}_0^j} \alpha_k^*, \qquad j \in \mathcal{J}. \tag{14}$$

Now, assume that the inequalities

$$\sum_{j \in \mathcal{J}_0^k} p_{ij} > \sum_{j \in \mathcal{J}_2^k} p_{ij}, \qquad k \in \bar{\mathcal{K}}, \quad i \in \mathcal{I}, \tag{15}$$

hold, scale the inequalities (15) by α_k^* and sum over $k \in \bar{\mathcal{K}}$, for each $i \in \mathcal{I}$. Then, it follows that

$$\sum_{k \in \bar{\mathcal{K}}} \alpha_k^* \sum_{j \in \mathcal{J}_0^k} p_{ij} > \sum_{k \in \bar{\mathcal{K}}} \alpha_k^* \sum_{j \in \mathcal{J}_2^k} p_{ij} \quad \Longrightarrow \quad \sum_{j \in \mathcal{J}} p_{ij} \left(\sum_{k \in \mathcal{K}_0^j} \alpha_k^* - \sum_{k \in \mathcal{K}_2^j} \alpha_k^* \right) > 0. \tag{16}$$

Then, inserting the rightmost equality of (14) in (16) yields a contradiction. We conclude that the inequalities in (15) cannot all hold, and hence that there exists a $\bar{k} \in \bar{\mathcal{K}}$ and an $\bar{\imath} \in \mathcal{I}$ such that the inequality $\sum_{j \in \mathcal{J}_0^{\bar{k}}} p_{\bar{\imath}j} \leq \sum_{j \in \mathcal{J}_2^{\bar{k}}} p_{\bar{\imath}j}$ holds. Thus, there exists an $\bar{\boldsymbol{x}} \in X_{\mathsf{sa}} \cap X(z^{\bar{k}})$ such that $\bar{x}_{\bar{\imath}j} = 1$ for $j \in \mathcal{J}_0^{\bar{k}}$, $\bar{x}_{\bar{\imath}j} = x_{\bar{\imath}j}^{\bar{k}}$ for $j \in \mathcal{J}_1^{\bar{k}}$, $\bar{x}_{\bar{\imath}j} = 0$ for $j \in \mathcal{J}_2^{\bar{k}}$, and $\bar{x}_{(3-\bar{\imath})j} = x_{(3-\bar{\imath})j}^{\bar{k}}$ for $j \in \mathcal{J}$. Now since $z^{\bar{k}} = \underline{z}$ and $(\bar{\boldsymbol{x}}, z^{\bar{k}}) \in S$ it follows that $z^{\bar{k}} = z^*$, and hence that $h^* = \sum_{k \in \bar{\mathcal{K}}} \alpha_k^* z^k = \underline{z} = z^*$. ◀

▶ Remark 5. A consequence of Theorem 4 is that there is no duality gap for $m = 2$, since either the lower bound $\underline{z}$ can be strengthened, or the bound coincides with the optimal value.

▶ Remark 6. From (13) follows that $h^* \geq \underline{z}$ also for $m > 2$. However, if $h^* = \underline{z}$ then the lower bound $\underline{z}$ cannot be strengthened and the duality gap $z^* - h^*$ can be non-zero. Consider the following counterexample for $m = 3$, $n = 6$, $\underline{z} = 4$, $\overline{z} = 5$, and $\mathcal{K} \subseteq \{1, 2\}$:

$$\boldsymbol{p} = \begin{pmatrix} 6 & 1 & 2 & 2 & 3 & 6 \\ 1 & 6 & 3 & 3 & 6 & 1 \\ 1 & 1 & 6 & 6 & 4 & 2 \end{pmatrix}, \boldsymbol{x}^1 = \begin{pmatrix} 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}, \boldsymbol{x}^2 = \begin{pmatrix} 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 \end{pmatrix}, \alpha_1 = \alpha_2 = \frac{1}{2}. \tag{17}$$

It can be verified that $z^1 = z^2 = 4$ and $z^* = 5$ and that α_1 and α_2 constitute a feasible solution to the LP master problem (13). Hence, the relations $4 = \underline{z} \leq h^* \leq 4 < z^*$ hold, i.e., the duality gap is non-zero.

▶ Remark 7. In Theorem 4, the Dantzig–Wolfe reformulation was used. Preliminary results indicated that this is, however, an inefficient approach for computing h^*, because (i) when $\underline{z}$ was increased, most of the generated columns became infeasible and needed to be discarded from the master problem and (ii) solving the master problem was computationally expensive. Consider instead maximizing the Lagrangian dual function (4a) using a subgradient optimization method; then there is no negative computational impact of increasing the value of $\underline{z}$.

In [29] it was shown that the lower bound $h(\boldsymbol{u})$ can be improved according to

$$\bar{h}(\boldsymbol{u}) := \underset{z\in\{\underline{z},\ldots,\overline{z}\}}{\text{minimum}}\left\{ z \;\middle|\; \sum_{j\in\mathcal{J}} u_j \le f(z,\boldsymbol{u}) \right\}, \tag{18}$$

i.e., that the inequality $h(\boldsymbol{u}) \le \bar{h}(\boldsymbol{u})$, holds for all $\boldsymbol{u} \in \mathbb{R}^n$. The fact that f is an non-decreasing function of z was used to efficiently evaluate $\bar{h}(\boldsymbol{u})$, unlike our approach of computing $f(z,\boldsymbol{u})$ for every $z \in \{\underline{z},\ldots,\overline{z}\}$ (recall Section 3.1). Despite being the strongest bound suggested in [29], the authors stated that since $\bar{h}(\boldsymbol{u})$ was cumbersome to maximize their weaker bounds were sometimes more useful.

Note that the maximum of $\bar{h}(\boldsymbol{u})$ will not provide an improved bound compared to h^*, as shown in Lemma 8 below. However, since h^* is not always retrieved due to early termination of the subgradient algorithm, the improved bound (18) is evaluated for all values of $\boldsymbol{u}$ encountered. Moreover, we suggest not to use the improved bound (18) in the subgradient based algorithm (11) to decide the step length and direction. This motivated by Lemma 8 and because $\bar{h}$ is a discrete function and thus progress becomes less frequent and less regular, resulting in a quickly diminishing step size.

▶ **Lemma 8.** *If $h^* \le \underline{z}$, where $h^* = \max_{\boldsymbol{u}\in\mathbb{R}^n} h(\boldsymbol{u})$, then $\bar{h}(\boldsymbol{u}) \le h^*$, $\forall\, \boldsymbol{u} \in \mathbb{R}^n$.*

Proof. From (13) it is clear that $h^* \ge \underline{z}$; thus our assumption implies that $h^* = \underline{z}$ holds. Moreover, in every optimal solution $\boldsymbol{\alpha}^*$ of (13) we have that $\alpha^*_k = 0$ for all $k \in \mathcal{K}$ such that $z^k > \underline{z}$. Hence, restricting (13) to the subset of extreme points satisfying $z^k = \underline{z}$ does not modify the set of optimal solutions. As a consequence, the feasible set to the minimization in (4a) can be restricted to $z = \underline{z}$, i.e., it holds that

$$h^* = \max_{\boldsymbol{\mu}\in\mathbb{R}^n}\left\{ \sum_{j\in\mathcal{J}} \mu_j + \underline{z} - f(\underline{z},\boldsymbol{\mu}) \right\} \ge \sum_{j\in\mathcal{J}} u_j + \underline{z} - f(\underline{z},\boldsymbol{u}), \qquad \boldsymbol{u} \in \mathbb{R}^n. \tag{19}$$

Now assume that $\exists\, \bar{\boldsymbol{u}} \in \mathbb{R}^n : \bar{h}(\bar{\boldsymbol{u}}) > \underline{z}$, where $\bar{h}(\boldsymbol{u})$ is defined by (18). It follows that

$$\sum_{j\in\mathcal{J}} \bar{u}_j - f(\underline{z},\bar{\boldsymbol{u}}) > 0 \quad\Longrightarrow\quad h^* > \underline{z}, \tag{20}$$

which contradicts our assumption that $h^* \le \underline{z}$. Hence, $\bar{h}(\boldsymbol{u}) \le \underline{z}$ must hold for all $\boldsymbol{u} \in \mathbb{R}^n$ and the lemma follows, since the equality $h^* = \underline{z}$ holds. ◀

3.4 Strategies for the branch–and–bound algorithm

For the primal–dual pair z^* and h^* there might be a non-zero duality gap. Hence, the branch–and–bound algorithm is used in order to find an optimal solution. In order to get a small branch–and–bound tree we choose to use the *worst–first* node selection strategy, i.e., always choose the branch–and–bound node with the highest lower bound. To determine which task to branch on we use the rule suggested in [29, §4], but employ our residual costs $\bar{c}_{ij} := h(\boldsymbol{u} \mid x_{ij} = 1) - h(\boldsymbol{u})$, i.e., $\bar{\jmath} \in \arg\max_{j\in J:|\mathcal{I}_j|>1}\{m - |\mathcal{I}_j| + \Theta \min_{i\in\mathcal{I}_j} \bar{c}_{ij}\}$, where $\mathcal{I}_j$ is the set of machines available for task j and $\Theta = 20$, as suggested in [29]. For a selected task j, we partition the available machines into two equally sized sets by repeatedly assigning the unassigned machine having the maximum $\bar{c}_{ij}$, in an alternating fashion. The idea is to make the two branches equally promising, and preliminary results indicated that this approach is more effective than branching on a single variable. We think that a more advanced partitioning rule, such as reliability branching [4], local branching [13], or machine learning based branching [5], could improve our algorithm and intend to develop these ideas in future research.

In the root node, the LP dual solution is utilized as a starting point. This provides an initial lower bound $\underline{z}$ and initial values for the Lagrangian multipliers $\boldsymbol{u}$ as the corresponding LP dual variable values. In the child nodes, the Lagrangian multipliers are initialized from the parent node. Note that, as in [29] the Lagrangian relaxation of the constraints in (2a) is used to compute the LP dual solution.

4 Heuristics

In Section 3 we seek to improve the lower bound $\underline{z} \le z^*$; however, its usefulness relies on a tight upper bound. As noted in Section 2, there are many heuristic algorithms that provide feasible solutions to the $R||C_{\max}$ (3). First, to find a feasible solution from a solution to (4a) we use a greedy *destruction–reconstruction heuristic*

that removes machines from multiply-assigned tasks, and assign a machine to each unassigned task, always greedily minimizing the makespan. Second, we construct a fractional solution as the convex combination of all subproblem solutions $\boldsymbol{x}^k$ (from the current branch–and–bound node) weighted proportionally to the step lengths t^k; this forms a so-called *ergodic sequence* of subproblem solutions; see [26, 33] for details. To construct a binary solution each job is assigned to the machine with the highest fraction. Third, we use a local search heuristic with the two neighbourhoods *3-cycle* and *2-1-swap* defined below; preliminary results indicated these to be sufficiently small w.r.t. the overall computing time. These neighbourhoods include the *reassign* and *interchange* neighbourhoods from [19] and are partially included in *cyclic exchange* and *path exchange* from [15].

(a) *3-cycle* **(b)** *2-1-swap*

Figure 1 Examples of operations allowed within the two neighbourhood definitions; each dot represents a variable x_{ij} and a filled dot encodes the value 1.

3-cycle: All perturbations of a feasible solution such that one task is assigned to another machine. The receiving machine can reassign one of its tasks to a third machine; the third machine can reassign one of its tasks to any of the two previous machines. See the illustration in Figure 1a. This is thus a special case of either a *cyclic exchange* or a *path exchange*.

2-1-swap: All perturbations of a feasible solution such that two tasks, assigned to a machine with zero slack in (3), are reassigned to another machine and one task from this machine is reassigned to the first machine. See the illustration in Figure 1b.

The neighbourhoods are searched (greedily) in the following order: task *reassign*, task *interchange*, *2-1-swap*, and *3-cycle*. Note that the entire neighbourhoods are searched, not just partially as done in [15] for the *cyclic exchange* and *path exchange* neighbourhoods. The heuristics described in this section are used in the branch–and–bound algorithm described in Section 3.4. We found it beneficial to run the heuristics every tenth iteration of the subgradient algorithm in the root node while only in the final iteration in the child nodes.

5 Tests and results

Our computational results were generated on a computer with an AMD Ryzen 9 3900X 12-Core 3.79 GHz and 32GB of RAM. To simplify the presentation of the results, all algorithms were limited to run on a single core, and report the CPU time. The implementations, see [2], are made in C++.

The instances and the algorithms tested are presented in Section 5.1. An analysis of the features of the Lagrangian relaxation algorithm is presented in Section 5.2. An overview of the performance is given in Section 5.3, and details on the results are given by type of test instances in Sections 5.4–5.6.

5.1 Test instances and algorithms

Since we were unable to use previously published work (instances and implementations), we use the MILP solver Gurobi 9 [21] as a point of reference by solving (3). Further, to ease future comparisons our instances are available online. In addition to this, we solve an extended formulation of (3) that includes the following *aggregated* variables subject to integrality constraints:

$$y_i = \sum_{j \in \mathcal{J}} x_{ij}, \qquad y_i \in \mathbb{Z}_+, \qquad i \in \mathcal{I}. \tag{21}$$

The idea behind the inclusion of these aggregated variables is the ability to branch on the number of tasks assigned to a machine, which is beneficial in particular when a machine contributing to the makespan is assigned a fractional number of tasks; after branching it is assigned more tasks in one of the branches and thus the makespan is likely to increase, leading to an improved quality of the lower bound. We disable presolve when using the aggregated variables to prevent them from being removed. A technique for branching based on presolve information is suggested in [17], hence presolve might not need to be disabled in other MILP solvers.

We have tested our formulations and algorithms on three sets of instances denoted: (i) *uncorrelated*, (ii) *correlated jobs*, and (iii) *correlated machines*. All combinations of the sizes $m \in \{3, 5, 10, 15, 20\}$ and $n \in \{30, 50, 80, 100, 200\}$ were tested, with ten random instances for each combination. *Uncorrelated* means that the processing times possess no correlation between machines or jobs, i.e., p_{ij} is generated from a discrete uniform distribution with the support $\{10, 11, \ldots, 100\}$, denoted $U\{10, 100\}$. *Correlated jobs* means that the processing times correlate with jobs, i.e., $p_{ij} = d_{ij} + b_j$, where b_j and d_{ij} are generated from $U\{1, 100\}$ and $U\{1, 20\}$, respectively. *Correlated machines* means that the processing times correlate with machines, i.e., $p_{ij} = a_i + d_{ij}$, where a_i and d_{ij} are generated from $U\{1, 100\}$ and $U\{1, 20\}$, respectively. Our instance settings are nearly identical to the ones used in [42], where we do not use $U\{1, 100\}$ for the *uncorrelated instances*, since such instances are solved faster (see, e.g., [12]) and since $U\{10, 100\}$ is a more common choice (see, e.g., [29, 32]).

Each instance is solved by eight different algorithms. DEF: solve the model (3) (*default* algorithm). CUT: solve model (3) using the *aggressive cuts* parameter setting in Gurobi. MOK: use the cutting plane method by Mokotoff and Chrétienne [32], with two differences: we use (a) Gurobi instead of CPLEX and (b) our heuristics instead of the suggested one to initialize the upper bound. DEF+$\boldsymbol{y}$, CUT+$\boldsymbol{y}$, and MOK+$\boldsymbol{y}$: include the aggregated variables and the constraints (21) in the models of the three methods, DEF, CUT, and MOK, respectively. LR: use the branch–and–bound algorithm based on the Lagrangian relaxation (4a), with the local search heuristic from Section 4. LR-F: LR with the variable fixing based on the reduced costs of the LP relaxation instead of Lemma 1.

The use of CUT and CUT+$\boldsymbol{y}$ are motivated by the fact that Gurobi's parameter tuning tool detected that the *aggressive cuts* parameter reduced the CPU time the most. For MOK and MOK+$\boldsymbol{y}$ the tuning tool did not find any significant reductions as compared with the default parameters. All algorithms were given a time limit of 120 s to solve each instance (to verified optimality). The components of the LR algorithm is analysed separately in Section 5.2.

5.2 Sensitivity analysis of the Lagrangian relaxation algorithm

In Sections 3 and 4 several features of the LR algorithm were presented; the impact of each of these is analysed here. The idea is to switch all of these features on and off and measure the corresponding overall solution times. Moreover, when a feature is switched off, a well-known alternative is sometimes used instead.

- *Var. fix.* denotes the use of the variable fixing based on Lemma 1 (on) or based on the reduced costs of the LP relaxation of the $R||C_{\max}$ (3) (off). Both options apply the fixing in each branch–and–bound node.
- *Subgr. iter.* denotes *few* (off) / *many* (on) subgradient iterations, to emphasize the importance of maximizing the strong dual function (4a). *Few* uses a maximum of 300 and 10 subgradient iterations in the root and child nodes respectively (cf. [29]). The corresponding numbers for *many* are 1000 and 100. Note that a small step length parameter δ_k also terminates the subgradient algorithm; cf. Section 3.2.
- *Deflect* denotes the use of the subgradient deflection method MDS (on) (recall Section 3.2) or to not deflect the subgradient s^k, i.e., letting $d^k := s^k$ in (11) (off).
- *LNBH* denotes the use of large neighbourhoods *3-cycle* and *2-1-swap* from Section 4 (on), or to only use the reassign and interchange neighbourhoods (off).
- *Erg. heur.* denotes the use of the rounding heuristic on fractional values retrieved by the ergodic sequence (on), or to use only the destruction–reconstruction heuristic on subproblem solutions (off) (recall Section 4). The local search heuristic is used to improve the solutions in both cases (on and off).

All combinations of using and not using each of the above stated features, and the resulting performance ratios $T_{\texttt{all}}/T$, are presented in Figure 2; where T and $T_{\texttt{all}}$ denote the execution time for an instance using a certain combination of features and using all features, respectively. Each combination of features was executed on ten *uncorrelated* and *correlated jobs* instances, with m=10 machines and n=100 tasks. We found that the ten *uncorrelated* instances were solved by every feature combination whereas the ten *correlated jobs* instances were solved only by a few of them. The *correlated machines* instances were also easy to solve (see Section 5.6); hence, to get a balanced set of instances these are not included here. Note that when an instance is not solved then the time limit $T = 120$ s is reported and the true execution time $T_{\texttt{true}} \geq 120$ s is underestimated; hence the ratio $T_{\texttt{all}}/T$ is an overestimate. However, for each of the 20 instances $T_{\texttt{all}} \leq 3.47$ s, hence the error introduced by the time limit is bounded by $\frac{T_{\texttt{all}}}{T} - \frac{T_{\texttt{all}}}{T_{\texttt{true}}} \leq \frac{3.47}{120} \leq 2^{-5}$.

Figure 2 shows the consistent benefit (by higher distributed scores) of using the variable fixing from Lemma 1 rather than basing it on the reduced cost from an LP relaxation. The figure also shows the importance of maximizing the Lagrangian dual function, since more computational effort (many subgradient iterations) pays-off,

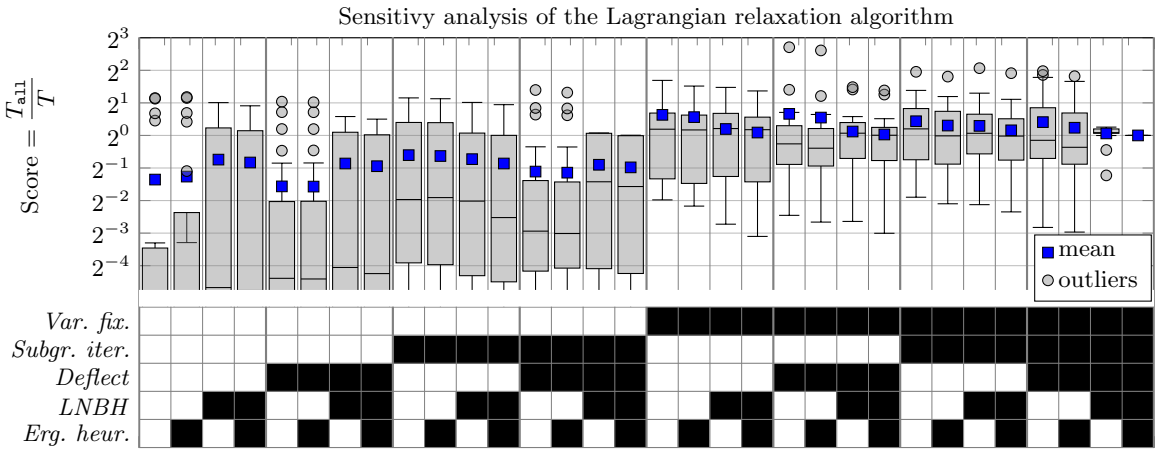

Figure 2 The impact of not using (off, white) or using a feature (on, black). T denotes each execution time on ten *uncorrelated* instances and ten *correlated jobs* instances of size $m = 10$ and $n = 100$. The score for each instance and feature is the ratio between switching all features on ($T_{\texttt{all}}$) and T. The scores for the 20 instances and each feature combination are presented with boxplots (median, lower and upper quartiles, and the most extreme values in the 1.5·IQR (inter-quartile range)), outliers (grey circles), and mean (blue squares).

especially when considering the median values. This is expected due to the strong bound it provides (recall Section 3.3). However, using many subgradient iterations have a negative impact on less demanding instances, thus affecting the mean values. On the primal heuristic side, we observe that searching the larger neighbourhood (i.e., *LNBH*) generally pays off in terms of the median values. For the instances considered in this section, there is no major improvement by using the ergodic sequence heuristic (i.e., *Erg. heur.*). However, for some larger *correlated jobs* instances in Section 5.5 these were slightly useful. Similarly, for the 20 instances in this section, the dominance criterion (9) did not have a measurable impact and is thus not illustrated in the figure; however, for some larger *correlated jobs* it was slightly beneficial, consequently; this criterion is always applied.

Another trend that is visible in Figure 2 is that usage of more features, especially *var. fix.* and *subgr. iter.* decreases the spread of computing times. This more robust behaviour is expected since more computational effort is made in each node aiming at decreasing the number of nodes that will need to be visited. Hence, some computationally easy instance might get an increase in computing time whereas a decrease is expected for harder instances. This manifests when employing all features except *LNBH*, which results in high mean and median scores, at the cost of roughly 25% of the instances having a score of a 2^{-1} or lower. Moreover, the deflected subgradient method seems dependent on using *LNBH*. We suspect that this is due to the smoother trajectory of the deflected version, which tends to improve the lower bound but also to reduce the variety of subproblem solutions; thus, the upper bound suffers without the *LNBH*. Related to the deflected method is that we also attempted to use the ADS deflection method, which had a bit slower convergence of the lower bound but found better feasible solutions faster. The overall difference between MDS and ADS was, however, not significant.

5.3 Overall performance

Figure 3 presents the fraction of solved instances by the eight algorithms described above as a function of CPU time. The figure indicates that the Lagrangian based branch–and–bound method is both the fastest and the most robust, i.e., it is able to solve the most instances within the given time limit of 120 s. Moreover, using the variable fixing from Lemma 1 seems to enable the solution of more instances. Another trend that is observed (although less prominent) is that the inclusion of the aggregated variables $\boldsymbol{y}$ reduces the computing time and enables the solution of more instances, in particular within the algorithms DEF+$\boldsymbol{y}$ and MOK+$\boldsymbol{y}$.

To establish the observed utility of using the aggregated variables $\boldsymbol{y}$, two statistical tests, cf. [8], are used. First, the McNemar's test considers the hypothesis that two algorithms (A and B) solve equally many instances. Let s^A and s^B denote the number of instances solved by only algorithm A and B, respectively; if the differences are random then $\frac{(s^A - s^B)^2}{s^A + s^B}$ follows a χ^2–distribution with one degree of freedom. Second, the one-sample Wilcoxon's signed rank test considers the hypothesis that the logarithm of the solution time ratios, i.e., $\log(T_i^A/T_i^B)$, originates from a distribution with zero median, i.e., that the two algorithms are equally fast. Here T_i^A and T_i^B denote the CPU times for the two algorithms, and only instances i solved by both algorithms are included. As a

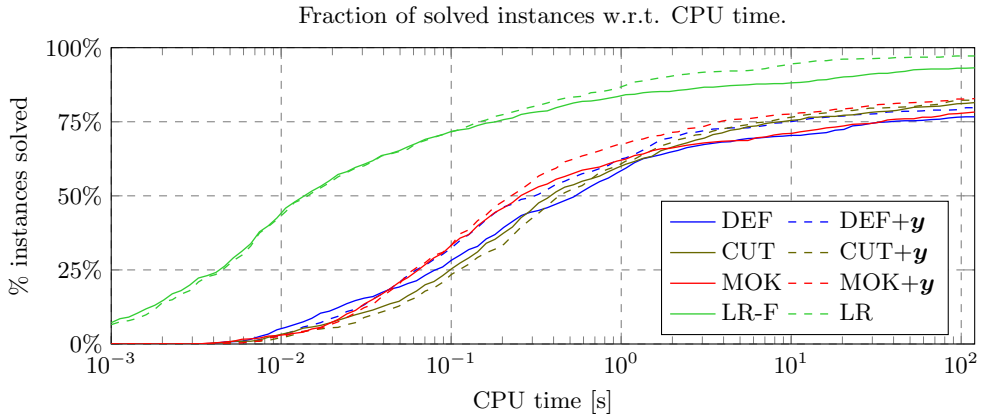

Figure 3 The seven algorithms applied to all the instances, *uncorrelated*, *correlated jobs*, and *correlated machines*.

result, both of these hypotheses are rejected with a certainty greater than 99.9% when comparing DEF with DEF+$\boldsymbol{y}$ and MOK with MOK+$\boldsymbol{y}$, where the average speed up (geometric mean of the ratios T_i^A/T_i^B) are in both cases 1.5. However, the hypotheses are not rejected when comparing CUT with CUT+$\boldsymbol{y}$ and the speed up is 1.0. To conclude, the reason why the algorithms DEF+$\boldsymbol{y}$ and MOK+$\boldsymbol{y}$ solve more instances faster as compared to DEF and MOK, respectively, is very unlikely to be a coincidence.

5.4 Uncorrelated instances

In Table 1, the average CPU times over ten instances and the number of unsolved instances for the *uncorrelated* instances are presented. Comparing the two first columns it seems that the algorithm DEF+$\boldsymbol{y}$ is faster than DEF for large problem instances. Likewise, for the tuned Gurobi parameters, the third and fourth columns

Table 1 The average computing time [s] for solving ten *uncorrelated* instances of size $m \times n$. Within parentheses are the numbers of instances exceeding the time limit of 120 s; these instances do not contribute to the average computing times. The average number of branch–and–bound nodes and the average gap ($\frac{\bar{z}-z}{\bar{z}}$) in the root node is presented for LR.

m	n	DEF	DEF+$\boldsymbol{y}$	CUT	CUT+$\boldsymbol{y}$	MOK	MOK+$\boldsymbol{y}$	LR-F	LR	LR nodes	LR root gap [%]
3	30	0.02	0.02	0.02	0.03	0.04	0.03	**0.0**	**0.0**	1.0	0.0
	50	0.02	0.02	0.03	0.03	0.04	0.03	**0.0**	**0.0**	1.0	0.0
	80	0.02	0.03	0.07	0.07	0.04	0.04	**0.0**	**0.0**	1.0	0.0
	100	0.03	0.04	0.13	0.16	0.07	0.06	**0.0**	**0.0**	1.0	0.0
	200	0.05	0.07	0.14	0.15	0.13	0.12	**0.02**	**0.02**	1.0	0.0
5	30	0.04	0.04	0.06	0.07	0.05	0.05	**0.0**	**0.0**	1.0	0.0
	50	0.06	0.08	0.09	0.11	0.1	0.09	**0.0**	**0.0**	1.0	0.0
	80	0.12	0.1	0.17	0.27	0.12	0.1	**0.0**	**0.0**	1.0	0.0
	100	0.11	0.16	0.29	0.32	0.17	0.14	**0.02**	**0.02**	1.5	0.02
	200	0.91	1.34	2.75	1.84	1.4	0.84	0.17	**0.14**	2.9	0.02
10	30	0.11	0.12	0.12	0.15	0.16	0.12	**0.0**	**0.0**	1.0	0.0
	50	0.43	0.25	0.24	0.54	0.66	0.26	**0.01**	**0.01**	1.0	0.0
	80	12.8	0.75	2.08	2.21	8.8 (2)	0.63	0.03	**0.02**	2.2	0.07
	100	17.1 (7)	10.3 (1)	6.37(1)	9.27	21.9 (5)	8.54	0.09	**0.06**	3.8	0.11
	200	0.39(7)	1.07(7)	15.9 (3)	19.7 (1)	35.5 (7)	17.5	0.85	**0.44**	7.6	0.14
15	30	0.15	0.11	0.1	0.14	0.03	0.06	0.01	**0.0**	1.0	0.0
	50	0.68	0.21	0.31	0.45	0.17	0.34	**0.01**	**0.01**	1.0	0.0
	80	13.1 (2)	3.8 (1)	1.82	1.15	19.8	9.18	0.16	**0.05**	2.4	0.22
	100	24.7 (3)	17.4 (3)	7.48	10.3	22.9 (1)	14.3	0.21	**0.1**	8.4	0.18
	200	84.7 (9)	43.8 (8)	28.0 (5)	37.5 (4)	57.0 (8)	27.8 (5)	3.19	**0.58**	29.6	0.19
20	30	0.16	0.12	0.08	0.12	0.01	0.03	**0.01**	**0.01**	1.0	0.0
	50	0.52	0.5	0.25	0.29	0.11	0.1	**0.01**	**0.01**	1.0	0.0
	80	1.25(1)	2.6	0.9	0.92	7.97(2)	2.23	**0.04**	**0.04**	1.0	0.0
	100	7.53(3)	3.9 (1)	3.75	2.98	34.1 (3)	22.0 (1)	0.12	**0.08**	1.2	0.13
	200	—	80.7 (8)	65.0 (8)	38.4 (5)	28.8 (8)	62.4 (7)	17.4	**1.09**	95.3	0.56

show that the algorithm CUT+$\boldsymbol{y}$ is faster than CUT for the largest instances. The same result holds for the algorithms MOK+$\boldsymbol{y}$ and MOK, for which the importance of the variables $\boldsymbol{y}$ seem to be the greatest. Apart from the LR algorithm, it seems that CUT+$\boldsymbol{y}$, or possibly MOK+$\boldsymbol{y}$, are the fastest and most robust methods for these instances.

Moreover, the LR algorithm is able to solve every *uncorrelated* instance in our test-bed, and branching is necessary only for large instances, verifying the strength of the Lagrangian lower bound (4a), whereas the variable fixing from Lemma 1 only marginally decreases the computing times. To get another point of reference, consider that the number of branch–and–bound nodes required by Martello et al. [29] to solve corresponding (unfortunately not identical) instances of size 10×80 was on average 5723, whereas the corresponding number for our algorithm is 2.2; a similar difference in magnitude holds for all large uncorrelated instances. This improvement is mainly explained by our focus to maximize the Lagrangian lower bound also in the child nodes. Note that the variable fixing also reduce the computing times but it is more important for harder instances; compare LR-F with LR.

5.5 Correlated jobs instances

In Table 2, the average CPU times over ten instances and the number of unsolved instances for the *correlated jobs* instances are presented. The most obvious difference from the *uncorrelated* instances is that the *correlated jobs* instances are harder to solve, and for the largest instance size none of the algorithms can solve all instances within the given time limit. Another different property of for the *correlated jobs* instances is that the aggregated variables $\boldsymbol{y}$ seem to be of less use for the algorithms DEF+$\boldsymbol{y}$ and CUT+$\boldsymbol{y}$, which for some instances require longer computing time and even exceed the time limit. For the algorithm MOK+$\boldsymbol{y}$ the result is, however, unchanged w.r.t. the *uncorrelated jobs* instances: this algorithm seems preferable among the first six algorithms.

The LR algorithm dominates the other algorithms on most instance sizes and it often requires the stronger variable fixing (i.e., Lemma 1). One reason for LR not being able to solve some instances, is that the local search heuristics failed to find the optimal solution. Moreover, we attempted to use a sophisticated heuristic such as the destruction–reconstruction heuristic described in [12], but this yielded no significant improvement as compared

Table 2 The average computing time [s] for solving ten *correlated jobs* instances of size $m \times n$. Within parentheses are the numbers of instances exceeding the time limit of 120 s; these instances do not contribute to the average computing times. The average number of branch–and–bound nodes and the average gap ($\frac{\overline{z}-z}{\overline{z}}$) in the root node is presented for LR.

m	n	DEF	DEF+$\boldsymbol{y}$	CUT	CUT+$\boldsymbol{y}$	MOK [s]	MOK+$\boldsymbol{y}$	LR-F	LR	LR nodes	LR root gap [%]
3	30	0.02	0.04	0.06	0.09	0.04	0.03	**0.0**	**0.0**	1.0	0.0
	50	0.07	0.08	0.08	0.12	0.07	0.06	**0.0**	**0.0**	1.0	0.0
	80	0.08	0.09	0.18	0.2	0.12	0.05	**0.01**	**0.01**	1.0	0.0
	100	0.1	0.12	0.15	0.19	0.09	0.09	**0.02**	**0.02**	1.0	0.0
	200	0.08	0.06	0.1	0.17	**0.06**	0.07	0.09	0.1	1.0	0.0
5	30	4.31	1.65	3.52	2.42	3.72	1.21	**0.0**	**0.0**	1.0	0.0
	50	12.3	8.46	9.61	13.9	8.7	3.77	0.05	**0.04**	3.9	0.06
	80	7.18 (1)	9.03	14.1	32.3	13.9	3.46	0.15	**0.12**	6.0	0.04
	100	26.5	21.1 (1)	43.1 (1)	35.7 (3)	22.0	5.35	0.33	**0.28**	5.6	0.05
	200	15.6	9.83 (1)	36.5	14.1 (1)	15.9	4.26	2.21	**1.85**	16.7	0.04
10	30	33.4 (3)	40.3 (2)	22.5 (1)	20.6 (2)	34.4 (5)	24.5	0.03	**0.02**	4.6	0.14
	50	40.6 (8)	—	65.3 (9)	72.0 (8)	—	27.4 (9)	0.29	**0.09**	13.2	0.37
	80	—	—	114 (9)	—	—	29.3 (9)	9.26	**0.75**	273.1	0.39
	100	—	—	—	—	—	—	28.6	**1.8**	599.0	0.42
	200	—	—	—	—	—	—	48.9 (8)	**10.1**	1611.3	0.23
15	30	3.93 (1)	1.32	1.19	2.45	0.94	2.89 (1)	**0.01**	**0.01**	1.0	0.0
	50	—	—	—	—	—	—	1.15	**0.27**	122.8	1.01
	80	—	—	—	—	—	—	28.9	**3.51**	1565.1	0.84
	100	—	—	—	—	—	—	42.7 (9)	**14.5**	5466.8	0.94
	200	—	—	—	—	—	—	—	**87.5** (9)	23 162.3	0.44
20	30	0.63	0.15 (1)	0.42	0.89	0.01 (1)	0.14	**0.01**	**0.01**	1.0	0.0
	50	79.4 (9)	66.1 (8)	66.5 (9)	5.28 (9)	9.77 (8)	58.4 (7)	0.38	**0.17**	42.5	0.97
	80	—	—	—	—	—	—	54.0 (4)	**6.46**	2903.3	1.33
	100	—	—	—	—	—	—	—	**39.8** (2)	19 980.4	1.18

Table 3 The average computing time [s] for solving ten *correlated machines* instances of size $m \times n$. Within parentheses are the numbers of instances exceeding the time limit of 120 s; these instances do not contribute to the average computing times. The average number of branch–and–bound nodes and the average gap ($\frac{\overline{z}-z}{\overline{z}}$) in the root node is presented for LR.

m	n	DEF	DEF+$\boldsymbol{y}$	CUT	CUT+$\boldsymbol{y}$	MOK	MOK+$\boldsymbol{y}$	LR-F	LR	LR nodes	LR root gap [%]
3	30	0.01	0.02	0.03	0.03	0.02	0.02	**0.0**	**0.0**	1.0	0.0
	50	0.03	0.02	0.03	0.07	0.04	0.03	**0.0**	**0.0**	1.0	0.0
	80	0.05	0.03	0.08	0.05	0.07	0.05	**0.0**	**0.0**	1.0	0.0
	100	0.04	0.04	0.06	0.05	0.06	0.06	**0.0**	**0.0**	1.0	0.0
	200	0.06$^{(1)}$	0.07	10.1	0.2	0.31	0.1	**0.02**	**0.02**	1.0	0.0
5	30	0.03	0.02	0.07	0.03	0.03	0.03	**0.0**	**0.0**	1.0	0.0
	50	0.08	0.05	0.07	0.06	0.07	0.06	**0.0**	**0.0**	1.0	0.0
	80	0.25$^{(1)}$	0.2	1.1	0.6	0.21	0.16	**0.02**	**0.02**	1.3	0.04
	100	0.63$^{(1)}$	0.47	0.5	0.62	10.4	0.14	**0.02**	**0.02**	1.0	0.0
	200	6.97$^{(3)}$	3.98	0.83$^{(2)}$	1.67	5.14$^{(2)}$	0.67	**0.11**	0.12	1.0	0.0
10	30	0.11	0.07	0.19	0.16	0.11	0.1	**0.0**	**0.0**	1.0	0.0
	50	0.24	0.09	0.21	0.21	0.15	0.14	**0.01**	**0.01**	1.0	0.0
	80	0.41	0.18	0.52	0.35	5.78	0.22	**0.03**	**0.03**	1.2	0.05
	100	0.54	0.19	0.48	0.35	0.48	0.33	**0.02**	**0.02**	1.0	0.0
	200	1.19$^{(3)}$	0.63	2.3 $^{(1)}$	1.24	1.89$^{(1)}$	0.93	0.61	**0.38**	1.2	0.01
15	30	0.22	0.1	0.17	0.16	0.07	0.06	**0.0**	0.01	1.0	0.0
	50	0.46	0.16	0.39	0.32	0.18	0.22	**0.01**	**0.01**	1.0	0.0
	80	0.74	0.46	0.44	0.48	0.3	0.24	**0.02**	**0.02**	1.0	0.0
	100	1.14	0.61	0.65	0.6	0.36	0.37	**0.03**	**0.03**	1.0	0.0
	200	4.66	1.28	2.1	1.47	10.6	0.97	1.63	**0.57**	4.3	0.03
20	30	0.29	0.16	0.27	0.26	0.07	0.07	**0.01**	**0.01**	1.0	0.0
	50	0.56	0.4	0.53	0.41	0.15	0.15	**0.02**	**0.02**	1.0	0.0
	80	0.97	0.72	1.22	0.72	0.48	0.46	**0.05**	**0.05**	1.0	0.0
	100	1.08	1.07	1.45	0.91	0.52	0.54	**0.07**	**0.07**	1.0	0.0
	200	2.67$^{(2)}$	2.28	2.54	2.15	7.94	1.1	**0.38**	0.4	1.0	0.0

to our simple local search heuristic. When an optimum was found, the verification of optimality was usually quite immediate. Hence, for these unsolved instances the weakness is in the upper bound. Note, however, that the gap is roughly 1% or less in the root node, thus the quality of either bound is still quite good. Moreover, for the instances we were able to solve, e.g., for $m = 10$ we also computed the lower gap, i.e., $\frac{z^*-z}{z^*}$: its value of roughly 0.05–0.1% again suggests that the issue is not the strength of the lower bound but the weakness of the upper bound.

5.6 Correlated machines instances

In Table 3, the average CPU times over ten instances and the number of unsolved instances for the *correlated machines* instances are presented. We observe that the three Gurobi-based algorithms benefit from the inclusion of the aggregated variables; e.g., DEF+$\boldsymbol{y}$ is preferred over DEF. Furthermore, similarly to the *uncorrelated* instances, the algorithm MOK+$\boldsymbol{y}$ seems to be the fastest and most robust among the first six algorithms. Unlike for the previously reported instances, it also seems to be competing with the algorithm LR. Note that the LR algorithm almost never needs to branch on these instances, since the bounds from the root node are sufficient to determine optimality.

6 Conclusion

We have developed and analysed an exact algorithm for the solution of the $R||C_{\max}$, based on the Lagrangian relaxation suggested by Martello et al. [29]. We derived a variable fixing strategy based on the solution of the subproblem consisting of m binary knapsack problems, in order to reduce the problem size. We showed that for $m = 2$ the Lagrangian relaxation has no duality gap. We performed a sensitivity analysis of our proposed algorithm features by computing their individual contribution to the overall algorithm performance. We also compared our method with several MILP formulations, e.g., including the aggregated variables in a standard

model for $R||C_{\max}$ and in the model suggested by Mokotoff and Chrétienne [32]. These models are of special interest since they can also be applied to extensions of the $R||C_{\max}$, for example with side-constraints.

For the majority of our 750 instances with sizes ranging from $m = 3, n = 30$ to $m = 20, n = 200$, our branch–and–bound algorithm based on Lagrangian bounds is superior to the other algorithms tested, including our versions of the cutting plane algorithm from [32], both with and without branching on the aggregated variables. The numbers of nodes traversed in the branch–and–bound tree are magnitudes fewer than that reported in [29]. This property indicates the strength of the Lagrangian relaxation and the high impact of our variable fixing strategy. Moreover, the sensitivity analysis indicates that our proposed variable fixing outperforms variable fixing based on the LP reduced costs and also that a careful maximization of the Lagrangian dual function successfully reduces the computation time.

Furthermore, we found that, by including the aggregated variables in the model suggested in [32], the algorithm (i.e., MOK+$\boldsymbol{y}$) was able to solve more instances within the given time limit. The general trend is that these variables contribute to an efficient partitioning of the solution space which increases the LP lower bounds.

To conclude, we have identified some possible further studies to conduct. First, we have highlighted some potential improvements to our branch–and–bound method: (i) a further development of the branching strategies, (ii) the efficient solution of the subproblems, (iii) a tuning of the subgradient algorithm, (iv) a concurrent traversal of the branch–and–bound tree, (v) the possibility of branching also on the aggregated variables suggested for the MILP models. Second, it will be interesting to investigate the inclusion of set packing constraints (see Section 1) in the model to find out if a generalization of our Lagrangian based method can still outperform a general MILP solver. Third, since the inclusion of the aggregated variables seems to be beneficial for most problem instances and for both MILP models, we expect that there exist many other applications that can benefit of this kind of extension. Such an extension could be a generalization of the $R||C_{\max}$ with additional side-constraints, or another problem with a makespan objective.

Acknowledgements

This work was carried out within the project Smart Assembly 4.0, supported by the Swedish Foundation for Strategic Research (SSF), project no. RIT15-0025. It is also part of the Sustainable Production Initiative and the Production Area of Advance at Chalmers University of Technology.

Finally, we would like to thank the anonymous reviewers for their valuable feedback.

References

1 Edvin Åblad. Intersection-free load balancing for industrial robots. Master's thesis, Department of Mathematical Sciences, Chalmers University of Technology and University of Gothenburg, 2016.

2 Edvin Åblad. R||Cmax Solver. Fraunhofer-Chalmers Centre GitHub repository, 2021. `https://github.com/Fraunhofer-Chalmers-Centre/RCmax/releases/v1.0`.

3 Edvin Åblad, Domenico Spensieri, Robert Bohlin, and Johan S. Carlson. Intersection-free geometrical partitioning of multirobot stations for cycle time optimization. *IEEE Trans. Autom. Sci. Eng.*, 15(2):842–851, 2018. doi:10/gc2bhc.

4 Tobias Achterberg, Thorsten Koch, and Alexander Martin. Branching rules revisited. *Oper. Res. Lett.*, 33(1):42–54, 2005. doi:10/ckh5pw.

5 Alejandro Marcos Alvarez, Quentin Louveaux, and Louis Wehenkel. A machine learning-based approximation of strong branching. *INFORMS J. Comput.*, 29(1):185–195, 2017. doi:10/f9vv3c.

6 Mokhtar S. Bazaraa, Hanif D. Sherali, and C. M. Shetty. Subgradient optimization methods. In *Nonlinear Programming: Theory and Algorithms*. John Wiley & Sons, 2006. doi:10/dftkhd.

7 Rachid Belgacem and Abdessamad Amir. A new modified deflected subgradient method. *J. King Saud Univ. Sci.*, 30(4):561–567, 2018. doi:10/ggmgxd.

8 Timo Berthold. *Heuristic algorithms in global MINLP solvers*. PhD thesis, Technische Universität Berlin, 2014.

9 Leonardo Borba and Marcus Ritt. A heuristic and a branch-and-bound algorithm for the assembly line worker assignment and balancing problem. *Comput. Oper. Res.*, 45:87–96, 2014. doi:10/ggmgxc.

10 Paolo M. Camerini, Luigi Fratta, and Francesco Maffioli. On improving relaxation methods by modified gradient techniques. In *Nondifferentiable Optimization*, pages 26–34. Springer, 1975. doi:10/ggmgxf.

11 Michele Conforti, Gerard Cornuejols, and Giacomo Zambelli. *Integer Programming*. Springer, 2014. doi:10/ggmgxb.

12 Luis Fanjul-Peyro and Rubén Ruiz. Iterated greedy local search methods for unrelated parallel machine scheduling. *Eur. J. Oper. Res.*, 207(1):55–69, 2010. doi:10/bhts3p.

13 Matteo Fischetti and Andrea Lodi. Local branching. *Math. Program.*, 98(1):23–47, 2003. doi:10/bpxmbz.

14 Matteo Fischetti and Paolo Toth. An additive bounding procedure for combinatorial optimization problems. *Oper. Res.*, 37(2):319–328, 1989. doi:10/bx8h5z.

15 Antonio Frangioni, Emiliano Necciari, and Maria Grazia Scutellà. A multi-exchange neighborhood for minimum makespan parallel machine scheduling problems. *J. Comb. Optim.*, 8(2):195–220, 2004. doi:10/frqzsm.

16 Martin Gairing, Burkhard Monien, and Andreas Woclaw. A faster combinatorial approximation algorithm for scheduling unrelated parallel machines. In Luís Caires, Giuseppe F. Italiano, Luís Monteiro, Catuscia Palamidessi, and Moti Yung, editors, *International Colloquium on Automata, Languages, and Programming*, volume 3580 of *Lecture Notes in Computer Science*, pages 828–839. Springer, 2005. doi:10/fs8r2p.

17 Gerald Gamrath, Anna Melchiori, Timo Berthold, Ambros M Gleixner, and Domenico Salvagnin. Branching on multi-aggregated variables. In Laurent Michel, editor, *Integration of AI and OR Techniques in Constraint Programming*, volume 9075 of *Lecture Notes in Computer Science*, pages 141–156. Springer, 2015. doi:10/d7bf.

18 Marco Ghirardi and Chris N. Potts. Makespan minimization for scheduling unrelated parallel machines: A recovering beam search approach. *Eur. J. Oper. Res.*, 165(2):457–467, 2005. doi:10/bg8d5p.

19 Celia A. Glass, Chris N. Potts, and P. Shade. Unrelated parallel machine scheduling using local search. *Math. Comput. Modelling*, 20(2):41–52, 1994. doi:10/bbpbvc.

20 Ronald L. Graham, Eugene L. Lawler, Jan Karel Lenstra, and A. H. G. Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: A survey. In *Annals of Discrete Mathematics*, volume 5, pages 287–326. Elsevier, 1979. doi:10/bn7sj2.

21 Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2020. `http://www.gurobi.com`.

22 Michael Held, Philip Wolfe, and Harlan P. Crowder. Validation of subgradient optimization. *Math. Program.*, 6(1):62–88, 1974. doi:10/c89h3t.

23 Ellis Horowitz and Sartaj Sahni. Exact and approximate algorithms for scheduling nonidentical processors. *J. Assoc. Comput. Mach.*, 23(2):317–327, 1976. doi:10/c7tmbr.

24 Stefan Irnich, Guy Desaulniers, Jacques Desrosiers, and Ahmed Hadjar. Path-reduced costs for eliminating arcs in routing and scheduling. *INFORMS J. Comput.*, 22(2):297–313, 2010. doi:10/b97zzc.

25 Klaus Jansen and Lorant Porkolab. Improved Approximation Schemes for Scheduling Unrelated Parallel Machines. *Math. Oper. Res.*, 26(2):324–338, 2001. doi:10/ccdbjw.

26 Torbjörn Larsson, Michael Patriksson, and Ann-Brith Strömberg. Ergodic, primal convergence in dual subgradient schemes for convex programming. *Math. Program.*, 86(2):283–312, 1999. doi:10/bk33st.

27 Jan Karel Lenstra, David B. Shmoys, and Éva Tardos. Approximation algorithms for scheduling unrelated parallel machines. *Math. Program.*, 46(1–3):259–271, 1990. doi:10/fntb9s.

28 Silvano Martello, David Pisinger, and Paolo Toth. Dynamic Programming and Strong Bounds for the 0–1 Knapsack Problem. *Manage. Sci.*, 45(3):414–424, 1999. doi:10/fn4nz4.

29 Silvano Martello, François Soumis, and Paolo Toth. Exact and approximation algorithms for makespan minimization on unrelated parallel machines. *Discrete Appl. Math.*, 75(2):169–188, 1997. doi:10/dsjz39.

30 Silvano Martello and Paolo Toth. *Knapsack Problems: Algorithms and Computer Implementations*. John Wiley & Sons, 1990. `www.or.deis.unibo.it/kp/KnapsackProblems.pdf`.

31 Ethel Mokotoff. Parallel machine scheduling problems: A survey. *Asia-Pac. J. Oper. Res.*, 18(2):193–242, 2001.

32 Ethel Mokotoff and Philippe Chrétienne. A cutting plane algorithm for the unrelated parallel machine scheduling problem. *Eur. J. Oper. Res.*, 141(3):515–525, 2002. doi:10/dmz8rq.

33 Magnus Önnheim, Emil Gustavsson, Ann-Brith Strömberg, Michael Patriksson, and Torbjörn Larsson. Ergodic, primal convergence in dual subgradient schemes for convex programming, II: the case of inconsistent primal problems. *Math. Program.*, 163(1–2):57–84, 2017. doi:10/f94w2r.

34 David Pisinger. A minimal algorithm for the 0-1 knapsack problem. *Oper. Res.*, 45(5):758–767, 1997. doi:10/b4ppxw.

35 Boris T. Polyak. Minimization of unsmooth functionals. *U.S.S.R. Comput. Math. Math. Phys.*, 9(3):14–29, 1969. doi:10/cthfpb.

36 Marius Posta, Jacques A Ferland, and Philippe Michelon. An exact method with variable fixing for solving the generalized assignment problem. *Comput. Optim. Appl.*, 52(3):629–644, 2012. doi:10/fh84p5.

37 Claudio Sandi. Solution of the machine loading problem with binary variables. In B. Roy, editor, *Combinatorial Programming: Methods and Applications*, volume 19 of *NATO Advanced Study Institutes Series*, pages 371–378. Springer, 1975. doi:10/dncp.

38 Hanif D. Sherali and Osman Ulular. A primal-dual conjugate subgradient algorithm for specially structured linear and convex programming problems. *Appl. Math. Optim.*, 20(1):193–221, 1989. doi:10/cmgf25.

39 Steef L. van de Velde. Duality-based algorithms for scheduling unrelated parallel machines. *ORSA J. Comput.*, 5(2):192–205, 1993. doi:10/bm6hdj.

40 Vijay V. Vazirani. *Approximation Algorithms*. Springer, 1st edition, 2003. doi:10/ggmgw8.

41 Mariona Vilà and Jordi Pereira. A branch-and-bound algorithm for assembly line worker assignment and balancing problems. *Comput. Oper. Res.*, 44:105–114, 2014. doi:10/f5s6tk.

42 Andreas Wotzlaw. *Scheduling Unrelated Parallel Machines—Algorithms, Complexity, and Performance*. VDM Verlag, 2007.