Multi-machine scheduling lower bounds using decision diagrams

van den Bogaerdt, Pim; de Weerdt, Mathijs

DOI
10.1016/j.orl.2018.11.003

Publication date
2018

Document Version
Final published version

Published in
Operations Research Letters

Operations Research Letters 46 (2018) 616–621

Contents lists available at ScienceDirect

Operations Research Letters

journal homepage: www.elsevier.com/locate/orl

Multi-machine scheduling lower bounds using decision diagrams☆

Pim van den Bogaerdt*, Mathijs de Weerdt

Delft University of Technology, Faculty of Electrical Engineering, Mathematics and Computer Science, Van Mourik Broekmanweg 6, 2628 XE Delft, The Netherlands

ARTICLE INFO

Article history:
Received 14 May 2018
Received in revised form 2 November 2018
Accepted 5 November 2018
Available online 10 November 2018

Keywords:
Multi-machine scheduling
Lower bounds
Decision diagrams

ABSTRACT

We consider parallel multi-machine scheduling with due times, where a partition of jobs is given where jobs in the same partition have a common release time, possibly precedence constraints, and cannot overlap. A formulation of decision diagrams for this problem greatly improves upon a more natural extension of the state-of-the-art for single-machine scheduling, and can provide decent lower bounds, outperforming existing solvers given the same short runtime limit, for problem instances with large time scales and tight due times.

1. Introduction

Multi-machine scheduling is a fundamental scheduling problem modeling a range of important real-life problems from scheduling tasks on a CPU to optimizing manufacturing processes [12, Ch.1]. Such problems have shown to be computationally very challenging and are an active area of research [1]. Recently, single-machine scheduling problems have been solved more quickly by lower bounds produced by relaxed decision diagrams [8,10]. We show that this can be generalized to finding lower bounds for parallel multi-machine scheduling instances. Decision diagrams provide a new class of bounds, based on a discrete relaxation of the problem, which may be useful e.g. in combining with other bounds, such as additive bounding [16].

We are given n *jobs* that are to be scheduled without preemption on $m \leq n$ *machines*. Let $\mathcal{J}$ denote the set of all jobs. Each job $j \in \mathcal{J}$ has a *processing time* and a *due time*; let $\mathbf{p}$ (resp. $\mathbf{d}$) denote the n-vector of processing (resp. due) times.

We assume that the jobs are partitioned into k subsets (partitions), and that the jobs in a given subset may not overlap in time ("partition constraints"). Let $\mathcal{P}$ be the set of partitions. We allow each partition $i \in \mathcal{P}$ to have a *release time*, meaning its jobs cannot start before that time. Let $\mathbf{r}$ be the k-vector of release times, and let $P(j)$ denote the partition of job j. We also allow *precedence constraints* between jobs of the same partition. The precedence constraint $i \to j$ specifies that job i has to end before j starts. Let $\omega(j) = \{i \mid i \to j\}$.

A *schedule* assigns each job a starting time and a machine such that each machine processes at most one job at a time. In a given schedule S, a job j has *starting time* $s(j)$, and $m(j)$ denotes the machine j is scheduled on. The *processing time of a machine* i is $\max_{m(j)=i}(s(j) + p_j)$.

The objective function defines the cost of a schedule. We assume that objective functions are non-decreasing in the job start times. This is a reasonable assumption: we typically want jobs to complete as soon as possible. We also assume that the objective function can be written as a sum over jobs, i.e., $z(S) = \sum_j z_j(s(j))$. For example, the objective used in the experiments in this paper is the *total tardiness*, defined by $t(S) = \sum_j \max\{0, s(j) + p_j - d_j\}$.

Previous results. Relaxations of Mixed Integer Programming formulations (MIPs) for multi-machine scheduling problems (e.g. [2]) can provide lower bounds on the optimum. In particular, Baptiste et al. [3] focus on lower bounds for multi-machine scheduling by giving MIPs and a reduction to a flow problem, among other approaches. For single-machine scheduling, very good lower bounds can be found using decision diagrams [8,10]. In spite of this promising result, there is no previous work on the use of decision diagrams for finding lower bounds for multi-machine scheduling. Finding lower bounds enables one to appreciate the quality of a schedule, and save time by not searching for a better schedule if the lower bound is equal to the optimum.

Our results. We present and prove correctness of a decision diagram (DD) formulation for a multi-machine scheduling problem which greatly improves upon a more natural extension of the state of the art for single-machine scheduling [10]. We use a known reduction of an optimal schedule to a permutation of jobs, and propose a decision diagram relaxation based on this representation. We furthermore show that this provides decent lower bounds, outperforming existing solvers given the same short runtime limit, for problem instances with large time scales and tight due times.

☆ This work is based on the first author's Master's thesis (van den Bogaerdt, 2018).

* Corresponding author.
E-mail address: P.vandenBogaerdt@tudelft.nl (P. van den Bogaerdt).

https://doi.org/10.1016/j.orl.2018.11.003
0167-6377/

2. Background on decision diagrams

Many problems can be seen as a minimization problem involving variables $x_1, \ldots, x_n$ with finite domain, including the scheduling problem we consider. A *decision diagram* (DD) for an instance I of such a problem is a directed graph with the following properties (see, e.g., [7] for an elaborate introduction into decision diagrams for optimization). The nodes are partitioned into $n+1$ *layers* $L_1, \ldots, L_{n+1}$, and each node v contains a *state* S. For a node v in layer L_i, the choices to x_i we can make in state S are defined by the set of *feasible choices* $F(S)$, and each choice $x_i \in F(S)$ induces an outgoing edge of v with cost $c(S, x_i)$ to a node in layer L_{i+1} with state $\varphi(S, x_i)$. Layer L_1 (resp. L_{n+1}) consists of a single *root node* (resp. *leaf node*).

A root–leaf path corresponds to an assignment to $x_1, \ldots, x_n$. The definitions of S, F, c, φ are problem-specific and also depend on the type of DD. We discuss two types of DDs in this paper.

An *exact DD* is a DD such that the cost of a root–leaf path equals the objective value for the corresponding assignment, and the set of root–leaf paths corresponds exactly to the set of feasible solutions. The shortest root–leaf path in an exact DD corresponds with an optimal solution of I.

A problem with exact DDs is their possibly exponential size, and we therefore consider another type of DD: relaxed DDs. These provide *lower bounds* on the optimum. Given an integer $w \geq 1$, a *relaxed DD of width* $\leq w$ is a DD with the following additional properties. The cost of a root–leaf path is *at most as high* as the objective value for the corresponding assignment to $x_1, \ldots, x_n$; the set of root–leaf paths corresponds to a set of solutions that is a *superset* of the feasible solutions; and each layer contains at most w nodes.

A relaxed DD can be built layer-by-layer. After building each layer, multiple nodes can be *merged* into one to prevent exceeding the allowed width w. This algorithm is called *top-down compilation*. We define the merged nodes by means of a merge operator $\oplus$ that transforms two states into one merged state. We assume that the operator is associative, so that we can unambiguously allow it to have multiple states as input.

A theorem by Hooker [10, Theorem 1] describes when a merge operator indeed yields a relaxed DD (called *validity*). The theorem is fundamental for proving the correctness of our DD formulation (in Section 3). We present this theorem in a slightly more formal way, using the following definition of a state relaxation relation.

Definition 1. A *state relaxation relation* is a binary operator $\preceq$ that operates on states with the following properties:

- (R1) $\preceq$ is reflexive and transitive.
- (R2) If $S' \preceq S$, then $F(S') \supseteq F(S)$.
- (R3) If $S' \preceq S$, then for $j \in F(S)$, $c(S', j) \leq c(S, j)$.
- (R4) If $S' \preceq S$, then for $j \in F(S)$, $\varphi(S', j) \preceq \varphi(S, j)$.

Theorem 1 (*Hooker [10]*)**.** *Given a state relaxation relation $\preceq$, a merge operator $\oplus$ is valid if $S \oplus S' \preceq S, S'$ for all states S, S'.*

To compute the shortest root–leaf path in a DD, we keep track of the partial objective in each node. We can use this value also as a heuristic to choose what nodes to merge [10]: nodes with a high partial objective are not likely to lie on the shortest path. Thus, merging these may maintain the same lower bound on the shortest root–leaf path.

3. Multi-machine scheduling using DDs

The DD formulation for multi-machine scheduling we introduce in this paper uses an efficient search space, state description, and merge operator. We explain these elements and prove their correctness in this section.

3.1. Minimal schedules

To reduce the search space, we use the concept of *minimal* schedules and that it is sufficient to consider permutations of jobs with non-decreasing start times [11,13–15,18].

We say a schedule is *minimal* if each job is scheduled (1) as early as possible, and (2) on the machine with earliest completion time (ignoring this and later jobs; breaking ties by the smallest machine index).

The set of minimal schedules contains an optimal one: any optimal schedule can be made minimal without affecting optimality by iteratively scheduling a job earlier or by swapping the machine assignments of a subset of jobs.

Each minimal schedule can be represented as a permutation of the jobs by using the Smallest Processing Time First algorithm (SPTF) to transform a permutation into a schedule: schedule each job on the machine with smallest processing time so far (breaking ties by machine index), taking into account the release time of its partition. When scheduling a job, update the release time of its partition to the end time. SPTF takes into account precedence constraints if for each constraint $i \to j$, job i occurs before j in the permutation.

To see that a minimal schedule can be generated by SPTF, consider the permutation obtained by sorting the jobs on start time, breaking ties by machine index. Executing SPTF on this permutation gives back the same schedule because SPTF chooses the machine and time corresponding to the definition of a minimal schedule [11].

This idea also works with the precedence and partition constraints. Namely, the permutation obtained adheres to the precedence constraints and so the schedule by SPTF also does. Further, the partition constraints may be considered as a "dynamic release time": as we schedule a job in a partition, the release time of the unscheduled jobs of this partition are changed to the end time of that job. Hence, when scheduling a job, the partition constraints reduce to release time constraints, and so do not pose a complication compared to [11].

Moreover, by construction, the jobs in the permutation obtained this way are sorted on non-decreasing start times, so we only need to consider such permutations [18] cited by [11], also when computing a lower bound on the optimum.

3.2. Exact DD formulation

To define a DD, we give definitions for S, F, c, and φ (as explained in Section 2). Our exact DD formulation for multi-machine scheduling represents precisely all permutations with non-decreasing start times.

The state $S(v)$ of a node v is a tuple $(V, U, \mathbf{f}, \mathbf{t}, \mathbf{f^u}, \mathbf{t^u}, g)$. Below we introduce the terms in this tuple step-by-step, using a bar over the variables to indicate the value after a transition. First, the pair $(V, \mathbf{f})$ already provides sufficient information to build an exact DD, but without the partition and precedence constraints. Like [10], we define V as the set of jobs that appear on the path from the root to v. However, instead of a single number f, we now have a *vector* $\mathbf{f} = (f_1, \ldots, f_m)$ of length m that contains the total processing time *of each machine*. The root has $V = \varnothing$ and $\mathbf{f} = \mathbf{0}$. Since we consider identical machines, the ordering of elements in $\mathbf{f}$ is irrelevant. It is efficient for the merge operator of the relaxed DD to ensure that $\mathbf{f}$ is sorted in increasing order.

When we choose job j as the next item in the permutation, the new state $\varphi(S, j)$ contains

$$\overline{V} = V \cup \{j\}.$$

To take the partition constraints into account, the state contains a k-vector $\mathbf{t}$, which represents when a job can start because the

previous jobs of the partition have completed. The root has $\mathbf{t} = \mathbf{r}$ (release times). If we choose job j, resulting in completion time x, the new state $\bar{\mathbf{t}}$ is updated with

$\overline{t_{P(j)}} = x.$

For $1 \leq i \leq k, i \neq P(j)$, we set $\bar{t}_i = t_i$. In words, as we schedule job j, other jobs of its partition can only start after job j ends. We can now define the transition for $\mathbf{f}$: job j is scheduled at the first machine in $\mathbf{f}$ (like SPTF: recall that $\mathbf{f}$ is sorted), and can only start after time $t_{P(j)}$. That is,

$\bar{\mathbf{f}} = \text{sort}(\,(\max\{f_1, t_{P(j)}\} + p_j, f_2, \ldots, f_m)\,).$

The cost of this transaction is computed as

$c(S, j) = z_j(\max\{f_1, t_{P(j)}\}).$

The set U is included in the state to incorporate precedence constraints (similar to [8]). It represents which jobs occur on *some* path from the root to v, as opposed to V, which contains jobs that occur on *every* path from the root to v. The root has $U = \varnothing$. We only include a job j in the feasible set $F(S)$ if $\omega(j) \subseteq U$. When we make a job choice j, we add j to the set U in the new node, just like we do for V:

$\overline{U} = U \cup \{j\}.$

So in an exact DD, $U = V$, but these sets may be different in a relaxed DD.

The last step of our exact DD formulation is related to only considering permutations with non-decreasing start times (to make the DD concise). For this, we include a number g representing the starting time of the (last) incoming job, i.e.,

$\bar{g} = \max\{f_1, t_{P(j)}\},$

and an m-vector $\mathbf{f^u}$ and k-vector $\mathbf{t^u}$. The vectors $\mathbf{f^u}$ and $\mathbf{t^u}$ again represent the finishing times of the machines and partitions, respectively. In our exact DD formulation, $\mathbf{f^u} = \mathbf{f}$ and $\mathbf{t^u} = \mathbf{t}$, but in a relaxed DD these represent the respective upper bounds. The root has $\mathbf{f^u} = \mathbf{0}, \mathbf{t^u} = \mathbf{r}, g = 0$.

We may ignore any (next) job j which would be scheduled at time $\max\{f_1^u, t_{P(j)}^u\} < g$, because it is sufficient to consider permutations with non-decreasing start times (see Section 3.1). Also, we do not schedule jobs that have already been scheduled (V) and jobs for which a preceding job $i \in \omega(j)$ has not been completed. The definition of the feasible set is thus

$F(S) = \mathcal{J} - (V \cup \{j \mid \omega(j) \not\subseteq U\} \cup \{j \mid \max\{f_1^u, t_{P(j)}^u\} < g\}).$

The transition for $\mathbf{f^u}$ uses these vectors as input also:

$\overline{\mathbf{f^u}} = \text{sort}(\,(\max\{f_1^u, t_{P(j)}^u\} + p_j, f_2^u, \ldots, f_m^u)\,).$

Similarly, we define $\overline{t_{P(j)}^u} = \max\{f_1^u, t_{P(j)}^u\} + p_j$ and $\overline{t_i^u} = t_i^u$ for $i \neq P(j)$.

Summarizing, the root node has state

$S(r) = (\varnothing, \varnothing, \mathbf{0}, \mathbf{r}, \mathbf{0}, \mathbf{r}, 0),$

and edges are defined by

$\varphi((V, U, \mathbf{f}, \mathbf{t}, \mathbf{f^u}, \mathbf{t^u}, g), j) = (\overline{V}, \overline{U}, \bar{\mathbf{f}}, \bar{\mathbf{t}}, \overline{\mathbf{f^u}}, \overline{\mathbf{t^u}}, \bar{g}).$

3.3. Relaxed DD formulation

A relaxed DD formulation is implied by a state relaxation relation $\preceq$ and merge operator $\oplus$.

Definition 2. The *state relaxation relation* $\preceq$ between two states holds if each of the following relations hold pairwise on the elements in the state tuples: $\subseteq, \supseteq, \leq, \leq, \geq, \geq, \leq$. On vectors, $\leq$ and $\geq$ denote pairwise comparisons.

Definition 3. The *merge operator* $\oplus$ on two states applies the following operators on the elements in the state tuples: $\cap, \cup$, min, min, max, max, min. The min (max) operator on vectors takes pairwise minima (maxima).

There is now a difference between U and V. Intuitively, to obtain a valid relaxation, we need to underestimate V (to ensure we do not disregard any schedule) but overestimate U (to ensure we are not too strict regarding the precedence constraints). This, and the need for $\mathbf{f^u}$ and $\mathbf{t^u}$, becomes clear in the proof of Lemma 2.

To show that $\preceq$ is a state relaxation relation, we use the following lemma.

Lemma 1. *Let $\mathbf{a} \leq \mathbf{b}$ be sorted vectors of length m, and let $a'_1, b'_1 \in \mathbb{R}$. If $a'_1 \leq b'_1$, then* $\text{sort}\,((a'_1, a_2, a_3, \ldots, a_m)) \leq \text{sort}\,((b'_1, b_2, b_3, \ldots, b_m))$.

The proof of this lemma is straightforward but technical; it can be found in the first author's Master's thesis [17, Section 3.3].

Lemma 2. *The relation $\preceq$ is a state relaxation operator (cf. Definition 1).*

Proof. We show each of the properties (R1)-(R4) in order. Let S, S' be states on the same layer i such that $S' \preceq S$. (R1) follows directly from the reflexivity and transitivity of $\subseteq, \supseteq, \leq, \geq$ (pairwise and regular).

For (R2), let j be a job in $F(S)$. In other words, assume the following about j. First, $j \notin V$. Second, $\omega(j) \subseteq U$. Third, $\max\{f_1^u, t_{P(j)}^u\} \geq g$. We have to show that j is in $F(S')$, that is, we have to show three similar predicates.

The predicate $j \notin V'$ follows from $V' \subseteq V$. Next, $\omega(j) \subseteq U'$ follows from $U' \supseteq U$. For the inequality, note that $\mathbf{f^{u\prime}} \geq \mathbf{f^u}$, $\mathbf{t^{u\prime}} \geq \mathbf{t^u}$, $g' \leq g$, so indeed $\max\{f_1^{u\prime}, t_{P(j)}^{u\prime}\} \geq \max\{f_1^u, t_{P(j)}^u\} \geq g \geq g'$.

For (R3), let j be a feasible choice in S. We have to show $c(S', j) \leq c(S, j)$. The cost $c(S', j)$ is $z_j(\max\{f'_1, t'_{P(j)}\})$, which is indeed at most the cost of making the choice in S, $z_j(\max\{f_1, t_{P(j)}\})$, since $\mathbf{f'} \leq \mathbf{f}$ and $\mathbf{t'} \leq \mathbf{t}$ and the z_j are non-decreasing.

For (R4), consider a job choice j that is feasible in S. Let $\overline{S'}$ and $\overline{S}$ denote the state after making this choice in S' and S, respectively. We have to show $\overline{S'} \preceq \overline{S}$. We do this by showing the predicates in the definition of $\preceq$.

$\overline{V'} \subseteq \overline{V}$ and $\overline{U'} \supseteq \overline{U}$ follow directly from $V' \subseteq V, U' \supseteq U$.

For $\bar{\mathbf{f}'} \leq \bar{\mathbf{f}}$, consider the definition of φ:

$\bar{\mathbf{f}} = \text{sort}(\,(\max\{f_1, t_{P(j)}\} + p_j, f_2, \ldots, f_m)\,)$

$\bar{\mathbf{f}'} = \text{sort}(\,(\max\{f'_1, t'_{P(j)}\} + p_j, f'_2, \ldots, f'_m)\,)$

We have $\max\{f'_1, t'_{P(j)}\} + p_j \leq \max\{f_1, t_{P(j)}\} + p_j$ and we may thus apply Lemma 1 to conclude $\bar{\mathbf{f}'} \leq \bar{\mathbf{f}}$.

For $\bar{\mathbf{t}'} \leq \bar{\mathbf{t}}$, note that from $f'_1 \leq f_1$ and $t'_{P(j)} \leq t_{P(j)}$, we get $\overline{t'_{P(j)}} \leq \overline{t_{P(j)}}$. For $1 \leq i \leq k, i \neq P(j)$, we have equality in $\bar{t}'_i \leq \bar{t}_i$.

For $\overline{\mathbf{f^{u\prime}}} \geq \overline{\mathbf{f^u}}$, consider the equivalent predicate $\overline{\mathbf{f^u}} \leq \overline{\mathbf{f^{u\prime}}}$, which we can prove similarly to $\bar{\mathbf{f}'} \leq \bar{\mathbf{f}}$. The inequality $\overline{\mathbf{t^{u\prime}}} \geq \overline{\mathbf{t^u}}$ can be shown similarly to $\bar{\mathbf{t}'} \leq \bar{\mathbf{t}}$. Lastly, we have $\overline{g'} = \max\{f'_1, t'_{P(j)}\} \leq \max\{f_1, t_{P(j)}\} = \bar{g}$. □

We now consider the merge operator $\oplus$. This merge operator is associative, as required. We now prove the validity of this merge operator.

Theorem 2. *The merge operator $\oplus$ is valid.*

Proof. Let S, S' be states. Using Theorem 1 we only have to show that $S \oplus S' \preceq S, S'$.

First, $V \cap V' \subseteq V, V'$ and $U \cup U' \supseteq U, U'$ are trivial. Next, note that in the merged state $S \oplus S'$, the vectors $\min(\mathbf{f}, \mathbf{f'})$ and

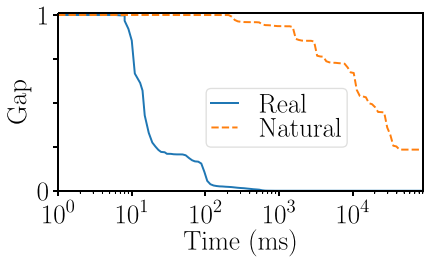

Fig. 1. Average gap over time for our DD formulation and the more natural extension we introduced based on the single-machine formulation by Hooker [10].

$\max(\mathbf{f^u}, \mathbf{f^{u\prime}})$ are automatically sorted because $\mathbf{f}, \mathbf{f}', \mathbf{f^u}, \mathbf{f^{u\prime}}$ are sorted. We have $\min(\mathbf{f}, \mathbf{f}') \leq \mathbf{f}, \mathbf{f}'$ because we take pairwise minima and do pairwise comparisons. Similarly, $\min(\mathbf{t}, \mathbf{t}') \leq \mathbf{t}, \mathbf{t}'$ as well as $\max(\mathbf{f^u}, \mathbf{f^{u\prime}}) \geq \mathbf{f^u}, \mathbf{f^{u\prime}}$ and $\max(\mathbf{t^u}, \mathbf{t^{u\prime}}) \geq \mathbf{t^u}, \mathbf{t^{u\prime}}$. Lastly, $\min\{g, g'\} \leq g, g'$ holds. □

The nodes with the highest partial objective are merged, breaking ties by (high) *slack* of the last scheduled job. The slack is the difference between due time and completion time.

4. Experiments

We evaluate the relaxed DD formulation using the total tardiness objective function. Our implementation is written in C# (x64, .NET 4.7). Whenever we compare against an upper bound, we use the best upper bound found by any of the solvers. Also, in our implementation, we used parallelization for building and merging: each thread builds and merges part of each layer. Recall that we merge nodes with highest partial objective.

In all experiments, we supplied all solvers with an initial solution provided by an adaptation of [13] (dealing with partition and precedence constraints). This algorithm quickly gives a very good upper bound.

4.1. Comparison to more natural formulation

We investigate to what extent our formulation improves upon the following more natural extension of the single-machine scheduling formulation by [10]: instead of using permutations, use a list of n (job, machine) pairs. Each job is scheduled on the machine it is paired to, and the order of jobs on each machine is given by the order of the list. Thus, the choices do not define just the job but also the machine. The costs, feasible sets and transition functions are defined accordingly. In this formulation the vector $\mathbf{f}$ is not sorted, and we do not apply the permutation optimization.

For the relaxation of both this and our new DD formulation, a width needs to be selected. We construct DDs iteratively with increasing width as follows: first we use the adaptation of [13] to obtain a schedule (upper bound). Then we compute lower bounds for widths 2^{12} and 2^{15}, and extrapolate these linearly on the logarithm to determine the width for which the lower bound is likely equal to the upper bound. We then repeat this process with this extrapolated width, or a width increased by a factor of 2^3, whichever is smaller.

Regarding the instances, we used the parameters and distributions based on those used by [10] for single-machine scheduling. The processing times $\mathbf{p}$ were integers drawn from $\{10, \ldots, 16\}$, the release times $\mathbf{r}$ were drawn from $\{0, \ldots, 5n\}$, and the due times $\mathbf{d}$ were drawn from $\{r_{P(j)} + (p_j + x)/m \mid x = 4 \cdot 10, \ldots, 4 \cdot 16\}$. We added the divisor m to model the fact that with m machines, we can roughly finish m times as quickly. The jobs were partitioned as evenly as possible, and each job had a random partition assigned. We set $n = 10$, $m = 2$, $k = 3$.

$$\begin{aligned}
&\text{Minimize} && \sum_j \max\{0, \text{StartOf}(I_j) + p_j - d_j\} \\
&\text{s.t.} && I_{j,M} \in \text{Intervals}([r_{P(j)}, \infty), p_j) && (j \in \mathcal{J},\ M \in \mathcal{M}) \\
&&& I_j \in \{I_{j,M} \mid 1 \leq M \leq m\} && (j \in \mathcal{J}) \\
&&& \text{NoOverlap}(\{I_{j,M} \mid 1 \leq j \leq n\}) && (M \in \mathcal{M}) \\
&&& \text{NoOverlap}(\{I_j \mid P(j) = i\}) && (i \in \mathcal{P}) \\
&&& \text{EndBeforeStart}(i, j) && (i, j \in \mathcal{J},\ i \to j)
\end{aligned}$$

Fig. 2. Constraint program.

$$\begin{aligned}
&C_j - C_i + L(1 - Z_{i,j}) \geq p_i && (P(i) = P(j)) \\
&C_i - C_j + LZ_{i,j} \geq p_j && (P(i) = P(j)) \\
&C_j \geq r_{P(j)} + p_j && (\text{replaces } (6)) \\
&Z_{i,j} = 1 && (i \to j) \\
&Z_{i,j} = 0 && (j \to i)
\end{aligned}$$

Fig. 3. Extra constraints for the MIP by [2], with $i, j \in \mathcal{J}$ with $i < j$, and the $Z_{i,j}$ variables are binary.

$$\sum_{\substack{j \\ P(j)=i}} \sum_{t'=g(t,j)}^{t} x_{j,t'} \leq 1 \qquad (i \in \mathcal{P},\ 0 \leq t < H)$$

$$\sum_{i \in \omega^{-1}(j)} \sum_{t'=0}^{t} x_{i,t'} \leq (t+1) \cdot |\omega^{-1}(j)| \cdot (1 - x_{j,t}) \qquad (j \in \mathcal{J},\ 0 \leq t < H)$$

Fig. 4. Extra constraints for the MIP by [3]. We define $g(t, j) = \max\{0, t - p_j + 1\}$ and $\omega^{-1}(j) = \{i \mid j \to i\}$.

For half of the instances, we set precedence constraints as follows. Consider a partition with p jobs $j_i, j_{i+1}, \ldots, j_{i+p-1}$. We added precedence constraints $j_i \to j_{i+1}, j_{i+2} \to j_{i+3}, \ldots$, ignoring the last job if p is odd. The reason we used these precedence constraints is that adding more may cause the search space to be reduced to a rather large extent.

We used 1500 of such instances. This experiment was performed on an Intel Core i7-3537U (2.00 GHz, 4 threads), 12 GB RAM, Windows 10, with a maximum width of 2^{22}. Fig. 1 shows the decrease in gap (using the upper bound by [13]) for both our "real" DD formulation and the more natural formulation. We observe that with more time, and thus DDs with incrementally larger widths, the average gaps for both formulations decreases significantly. Moreover, we see that our formulation is clearly superior. For the more natural formulation, the median time for a non-zero lower bound was about 0.35 s, and for only 465 instances it gave a final lower bound equal to the upper bound. To compare, the new formulation we propose in this paper gave a lower bound equal to the upper bound for 1473 instances, and the median time to solve these instances to optimality was about 0.1 s.

We conclude that extending [10] to multi-machine scheduling is not trivial, and that the relaxation we introduced in this paper is essential to obtain decent performance.

4.2. Comparison to CP and MIP

We also compare our DD formulation to a constraint program (CP) and two mixed integer programs (MIPs). This experiment was run on an Intel Xeon E5-1620 v2 (3.70 GHz, 8 threads), 8 GB RAM, Windows 7.

The CP (Fig. 2) was solved by IBM ILOG CP Optimizer 12.8 and is based on one of the samples provided with the solver. We

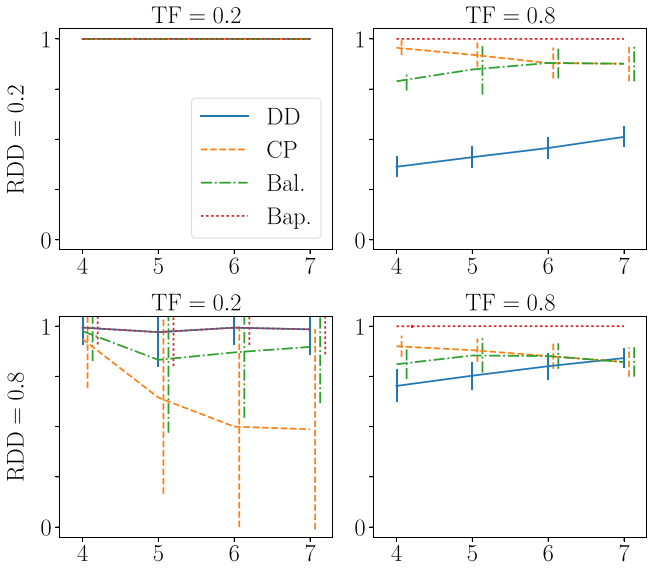

Fig. 5. Average gap with standard deviation after 50 ms for TF/RDD instances as a function of k, with $n = 4k$, $m = k - 2$. The number of instances per plot varies between 510 and 551. Bal. (Bap.) refers to the MIP based on [2,3]. The error bars are shifted based on the solver so as not to let them overlap.

enabled presolving and used the "restart" method, which gives decent performance as well as intermediate lower bounds.

The MIPs were solved by both IBM ILOG CPLEX 12.8 and Gurobi 8.0.1 with default parameters; for each instance, the best result is reported. The first MIP is adapted from [2], which has few variables. See Fig. 3. The second MIP is based on a time-indexed one given by [3]. It has a binary variable for each (job, time) pair, where "time" is any time step a job could be scheduled. For this MIP, we used the makespan of a random schedule as horizon H. See Fig. 4. We used this MIP itself as well as its LP relaxation (as suggested in [3]), both with and without presolving, again reporting only the best result. We start measuring time after having built the model.

We created instances based on the TF/RDD model [4,5] adapted to multiple machines. Given TF, RDD $\in [0..1]$, the due times were drawn uniformly from the integers between $\lfloor r_j + \sum_j p_j(1 - \text{TF} - \text{RDD}/2)/m \rfloor$ and $\lfloor r_j + \sum_j p_j(1 - \text{TF} + \text{RDD}/2)/m \rfloor$ inclusive, with negative values removed. Thus, TF (RDD) controls the expectation (variance) of the due times. The processing times were drawn from $\{75, \ldots, 150\}$. These are rather large because for much smaller times the MIP by [3] outperforms all other formulations. The release times are drawn from $\{0, \ldots, 111\}$. We set $n = 4k$, $m = k-2$, let each partition contain 4 jobs, and set precedence constraints for half of the instances like before.

We fixed a width of $w = 1000$ and gave each solver a time limit of 15 s. Fig. 5 shows the performance for these instances after 50 ms (our formulation was done within this time for all but two instances). We see that DD performs well if the due times are tight on average (large TF) and have a small variance (small RDD). The reason is that such due times cause relatively high tardiness, so that the merge heuristic is well-informed. In contrast, DD do not work well for TF $= 0.2$.

If more time is available, the CP, LP and MIP solvers give (much) better bounds. For example, after about 350 ms, the average gap of CP and the MIP by [2] are better than DD for TF $= 0.8$, RDD $= 0.2$, $k = 7$, except if we fix $m = 2$. After about 3 s, the LP relaxation of [3] gives average gaps close to zero for TF $= 0.8$, and after about 5.5 s, its averages are all less than those of DD.

However, for TF $= 0.8$, RDD $= 0.2$, and $m = 2$, DD outperforms all other models. Only after running about 20 to 100 times longer (1 s to 5 s), the model based on [3] eventually gives smaller average gaps than DD does. Moreover, this formulation uses many variables and constraints, and it takes time and memory to build these (which we did not take into account for the runtime). In particular, if the time scale is larger than what we used, this model may not be practical, while the runtime and memory of our DD formulation virtually do not depend on the time scale.

5. Discussion and future work

When aiming for better bounds, the memory usage in our formulation may become a bottleneck. Each node in our formulation uses $O(m+n)$ memory for its state, so that the total memory usage is $O(nw(m+n))$. We found empirically that one has to double w to get a constant increase in the lower bound, so the memory is exponential in the quality. Although under specific circumstances, an exponential DD is inevitable with our model (see Appendix A in [17]), a more sophisticated merge heuristic may reduce w significantly and is thus an interesting direction for further research.

Given that the DD formulation proposed in this paper is so effective if a small amount of runtime is available, it furthermore is very promising to embed it in algorithms that have worked with DDs successfully in the past, such as branch and bound [7, Chapter 6], logic-based Benders decomposition [9], constraint programming filtering techniques [8], and Lagrangian methods [6].

Finally, we may tackle other multi-machine scheduling problems using similar DDs, such as for unrelated machines.

Acknowledgments

The authors would like to thank Koos van der Linden for proofreading the article, and the anonymous reviewer for constructive comments.

Declaration of interest

When this work was carried out, the first author had a paid internship, and the second author was a visiting scientist at the Dutch Railways (NS).

Role of funding source

The NS had no involvement in the conduct of research, this article, or the decision to submit for publication.

References

[1] M.O. Adamu, A.O. Adewumi, A survey of single machine scheduling to minimize weighted number of tardy jobs, J. Ind. Manag. Optim. 10 (1) (2014) 219–241.
[2] N. Balakrishnan, J.J. Kanet, V. Sridharan, Early/tardy scheduling with sequence dependent setups on uniform parallel machines, Comput. Oper. Res. 26 (2) (1999) 127–141.
[3] P. Baptiste, A. Jouglet, D. Savourey, Lower bounds for parallel machine scheduling problems, Int. J. Oper. Res. 3 (6) (2008) 643–664.
[4] J.E. Beasely, Weighted tardiness (OR library), 2018, http://people.brunel.ac.uk/~mastjjb/jeb/orlib/wtinfo.html, originally described in [5], (Online Accessed 26 February 2018).
[5] J.E. Beasley, OR-Library: distributing test problems by electronic mail, J. Oper. Res. Soc. 41 (11) (1990) 1069–1072.
[6] D. Bergman, A.A. Cire, W.-J. van Hoeve, Lagrangian bounds from decision diagrams, Constraints 20 (3) (2015) 346–361.
[7] D. Bergman, A.A. Cire, W.-J. van Hoeve, J. Hooker, Decision Diagrams for Optimization, Springer, 2016.
[8] A.A. Cire, W.-J. van Hoeve, Multivalued decision diagrams for sequencing problems, Oper. Res. 61 (6) (2013) 1411–1428.
[9] A.A. Ciré, J.N. Hooker, The separation problem for binary decision diagrams, in: Proceedings of the International Symposium on Artificial Intelligence and Mathematics, 2014.

[10] J.N. Hooker, Job sequencing bounds from decision diagrams, in: Proceedings of the International Conference on Principles and Practice of Constraint Programming, Springer, 2017, pp. 565–578.
[11] A. Jouglet, D. Savourey, Dominance rules for the parallel machine total weighted tardiness scheduling problem with release dates, Comput. Oper. Res. 38 (9) (2011) 1259–1266.
[12] M.L. Pinedo, Scheduling: Theory, Algorithms, and Systems, fourth ed., Springer, 2016.
[13] R. Rodrigues, A. Pessoa, E. Uchoa, M. Poggi de Aragão, Heuristic algorithm for the parallel machine total weighted tardiness scheduling problem, Rel. Pesquisa Eng. Prod. 8 (10) (2008) 1–11.
[14] J.M.J. Schutten, List scheduling revisited, Oper. Res. Lett. 18 (4) (1996) 167–170.
[15] B. Simons, Multiprocessor scheduling of unit-time jobs with arbitrary release times and deadlines, SIAM J. Comput. 12 (2) (1983) 294–299.
[16] M. Slusky, Integrating Relaxations for Combinatorial Optimization (Ph.D. thesis), Carnegie Mellon University, 2015.
[17] P. van den Bogaerdt, Multi-Machine Scheduling Lower Bounds Using Decision Diagrams (Master's thesis), Delft University of Technology, The Netherlands, 2018.
[18] F. Yalaoui, C. Chu, New exact method to solve the $Pm/r_j/\sum C_j$ schedule problem, Int. J. Prod. Econ. 100 (1) (2006) 168–179.