Lower Bounds for Uniform Machine Scheduling Using Decision Diagrams

van den Bogaerdt, Pim; de Weerdt, Mathijs

DOI
10.1007/978-3-030-19212-9_38

Publication date
2019

Document Version
Final published version

Published in
Integration of Constraint Programming, Artificial Intelligence, and Operations Research

Citation (APA)
van den Bogaerdt, P., & de Weerdt, M. (2019). Lower Bounds for Uniform Machine Scheduling Using Decision Diagrams. In L.-M. Rousseau, & K. Stergiou (Eds.), *Integration of Constraint Programming, Artificial Intelligence, and Operations Research* (pp. 565-580). (Lecture Notes in Computer Science; Vol. 11494). Springer. https://doi.org/10.1007/978-3-030-19212-9_38

Lower Bounds for Uniform Machine Scheduling Using Decision Diagrams

Pim van den Bogaerdt(✉) and Mathijs de Weerdt

Faculty of Electrical Engineering, Mathematics and Computer Science,
Delft University of Technology, Van Mourik Broekmanweg 6,
2628 XE Delft, The Netherlands
P.vandenBogaerdt@tudelft.nl

Abstract. We propose a relaxed decision diagram (DD) formulation for obtaining lower bounds on uniform machine scheduling instances, based on *separators* to separate jobs on different machines. Experiments on the total tardiness for instances with tight due times show that for obtaining nontrivial bounds, it is important to partition the DD nodes on a layer based on their machine finishing time. When the number of jobs is small, DDs provide stronger bounds in less time than a time-indexed LP relaxation.

Keywords: Multi-machine scheduling · Uniform machines · Lower bounds · Decision diagrams

1 Introduction

We consider machine scheduling on uniform machines with release times and sequence-dependent setup times. This problem models an environment where machines have different speeds, time is incurred between jobs depending on the pair of jobs and their machine assignment, and each job may only be scheduled after a given time depending on the job. This is an abstract model of, for example, scheduling a production factory (see [17, pp. 1–2]).

The aim is to find a schedule that minimizes a given objective function. This problem is hard in many cases, for example it is NP-complete if there are two machines with equal speed, setup times and release times are not present, and the objective is the maximum completion time [15]. We propose a decision diagram formulation for a uniform machine scheduling problem that provides lower bounds on an objective for a given instance.

Decision diagrams (DDs) have proven useful as a tool in optimization (see, e.g., [7]). A particular successful application of decision diagrams has been single-machine scheduling (see, e.g., [10,12]) as well as multi-machine scheduling where the machines are considered identical [9]. To the best of our knowledge, there has so far been no extension to more general multi-machine scheduling problems where the machines are not exchangeable.

L.-M. Rousseau and K. Stergiou (Eds.): CPAIOR 2019, LNCS 11494, pp. 565–580, 2019.
https://doi.org/10.1007/978-3-030-19212-9_38

In this paper, we propose a DD formulation for uniform multi-machine scheduling; that is, each machine has a speed with which it processes a job. More precisely, we provide a formulation for a relaxed DD, which gives lower bounds on the optimal objective for instances of this problem. Lower bounds can be used to appreciate the quality of a feasible schedule, and may help in search algorithms such as branch-and-bound.

We also propose a merge heuristic based on partitioning nodes on their machine finishing time. An experiment for the total tardiness objective on instances with tight due times shows that this is important for obtaining non-trivial bounds. We also show that, at least when the number of jobs is small, bounds given by DDs are stronger and computed faster than those given by the linear programming relaxation of a time-indexed mixed integer program that [6] is based on, solved by IBM ILOG CPLEX.

The remainder of this paper is structured as follows. In Sect. 2, we introduce our problem formally and explain the basics of DDs. In Sect. 3, we briefly review previous work in the literature on DDs for single-machine scheduling by formulating a single-machine DD of our problem. In Sect. 4, we present our DD formulation for the multi-machine problem, and in Sect. 5, we elaborate on improving the bounds provided by our DD formulation. In Sect. 6, we present computational results. Finally, in Sect. 7, we conclude and present directions for future work.

2 Background

2.1 Problem Formulation

In a uniform machine scheduling problem, a number of *machines* is available that can each process one *job* at a time. The set of m machines is $\mathcal{M} = \{M_1, \ldots, M_m\}$ and the set of n jobs is $\mathcal{J} = \{j_1, \ldots, j_n\}$. Each job j has a *processing time* $p_{i,j}$ on machine i. We assume the ratio $p_{i,j}/p_{i',j}$ is constant over all jobs j for each pair of machines i, i'; that is, each machine has a *speed* with which it processes jobs.

Each job has a *release time* r_j and can only start after that time. We also allow a sequence-dependent *setup time* $\sigma_{M,j,j'}$ between each pair of jobs j, j' and for each machine M, which means job j' can only start $\sigma_{M,j,j'}$ time after job j has ended if both jobs are scheduled on M. The first job on a machine may have a setup time as well; let $\sigma_{M,\vdash,j}$ denote such a setup time for job j on machine M. Here, we let a "dummy job" $\vdash$ denote there is no job before j.

We assume the setup times satisfy the triangle inequality, that is, $\sigma_{M,j,j'} \leq \sigma_{M,j,j''} + \sigma_{M,j'',j'}$ for all jobs $j \in \mathcal{J} \cup \{\vdash\}, j', j'' \in \mathcal{J}$ and machines $M \in \mathcal{M}$. This is a reasonable assumption because a direct setup should not take more time than the time it takes performing the setup indirectly. The setup times need not be symmetric.

Release times look like setup times involving the dummy job, but are not superfluous. Namely, the release times are not involved in the above triangle

inequality. A slight generalization of this problem has been formulated (but not in the context of lower bounds) in [16].

In a *schedule*, each job $j \in \mathcal{J}$ has a machine assignment $m(j)$, start time $s(j)$ and completion (end) time $e(j) = s(j) + p_{m(j),j}$. We seek a schedule that minimizes an *objective function* $z(e(j_1), \ldots, e(j_n))$. We assume that the objective function is non-decreasing in each completion time, and that it can be written as a sum $\sum_j z_j(e(j))$ of functions, each depending on the completion time of only one job. In this article, we let each job have a *due time* d_j, and consider the *total tardiness* $\sum_j \max\{0, e(j) - d_j\}$ as the objective function. This is a measure of the total delay of the jobs in a schedule.

2.2 Decision Diagrams

A *decision diagram* (DD) is a directed acyclic graph for which all edges go from one layer to the next. (See [7] for an elaborate introduction to decision diagrams.) The first and last layer consist of a single node, called the *root* and *terminal*, respectively. One can model the search space of a minimization problem in n variables as a DD of $n + 1$ layers: each layer (except the last) corresponds to a variable in the sense that each outgoing edge of that layer corresponds to a choice of that variable. By setting an appropriate cost to each edge, the smallest root-terminal path represents the optimal solution of the problem. Such a DD is called *exact*.

Exact DDs may be of exponential size. To ensure tractability, we consider *relaxed DDs* with a bounded number of nodes. Such DDs may also represent infeasible solutions, and each root-terminal path is only required to be a *lower bound* on the objective value of the corresponding solution. These two properties together imply that the shortest root-terminal path in a relaxed DD is a lower bound on the optimal solution. To bound the number of nodes, we let each layer contain at most $w \geq 1$ nodes, where w is a fixed parameter.

To build an exact DD, one can keep track of what choices have been made by means of a *state* S in each node. This state can be used to define the set $F(S)$ of outgoing edges (i.e., choices) of this node as well as the cost $c(S, j)$ of each such edge. The state of a node after following an edge for choice j is defined as $\varphi(S, j)$. This algorithm of building a DD is called *top-down compilation*.

To build a relaxed DD, one needs to additionally ensure that each layer contains at most w nodes. This is done by *merging* nodes. The state of a merged node is defined through an associative *merge operator* $\oplus$. A merged node represents multiple paths from the root but only has a single state. This state should "underapproximate" these paths in some sense. Intuitively, the shortest root-terminal path then has a length that is an underapproximation (i.e., lower bound) of the optimum.

The merge operator needs to be *valid*, i.e., a DD resulting from applying it must be relaxed. In [12], a theorem is presented to prove the validity of a merge operator. We present it in the following formulation, like [9].

Definition 1. *A binary operator $\preceq$ on states is a* state relaxation relation *if it adheres to 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 [12]). *Suppose a state relaxation relation $\preceq$ is given. If $S \oplus S' \preceq S, S'$ for all states S, S' for which $\oplus$ is defined, then $\oplus$ is a valid merge operator.*

This theorem as presented here is actually slightly stronger than [12] because we only consider pairs for which $S \oplus S'$ is defined. The same proof as in [12] is applicable, however. The purpose of this addition becomes clear when we prove our DD formulation in Sect. 4.

3 DDs for Single-Machine Scheduling

DDs have been used for single-machine scheduling before (see, e.g., [10,12]). In this section, we review existing work by modelling the single-machine variant of our problem as a DD. In the next section, we extend this model to uniform machines.

A single-machine schedule can be considered as a permutation of the jobs. A permutation can be transformed into a schedule by considering the jobs iteratively, scheduling each job as soon as possible. The set of schedules generated this way includes an optimal schedule because we assume the objective function is non-decreasing in each of the job completion times. It is thus reasonable to let the DD represent permutations (see, e.g., [10,12]). We now explain the details of this formulation, which we then extend to multiple machines in the next section.

In each node, we keep track of a set V of jobs we have certainly already scheduled; that is, the jobs that appear on all paths from the root to the node. These jobs are removed from the feasible set in the node, because all paths through the edge for this job would contain that job twice, and so would not represent a feasible solution. The root has $V = \varnothing$ and the transition for job j is $\overline{V} = V \cup \{j\}$. (We use a bar to denote the state variable after a transition.) The merge operator is the intersection: given two nodes v_1, v_2, the jobs that occur on all paths from the root to the merged node are precisely the jobs that appear on all paths from the root to v_1 and to v_2.

When we schedule a next job, we need (a relaxation of) the finishing time of the machine, so that we know when the job can start. To this end, we keep a number f in each node that represents this finishing time [12]. The root has $f = 0$; the transition will be discussed below. The merge operator is the minimum. Intuitively, by underestimating the finishing time of the machine, we also underestimate the end time of jobs and hence their costs, thereby obtaining a valid relaxation on the objective.

To incorporate setup times, we use a set L of jobs that may have been scheduled last (see [7, p. 143][1]). If we make a transition for job j, we need to take into account the setup time between j and the previous job (or the dummy job if j is the first job). A valid relaxation (underestimate) of this setup time is $\min_{j' \in L} \sigma_{j',j}$. Hence, we could define the transition for f as $f + \min_{j' \in L} \sigma_{j',j} + p_j$. Rather, to take into account the release time, we define

$$\overline{f} = \max\{r_j, f + \min_{j' \in L} \sigma_{j',j}\} + p_j.$$

Note that in an exact DD, L is a singleton set, and we simply take the setup time between that job and j. The root has $L = \{\vdash\}$ and the transition for job j is $\overline{L} = \{j\}$. The merge operator is the union.

4 DD Formulation for Uniform Machines

In this section, we propose a DD formulation for uniform machines, based on the single-machine DD formulation discussed in the previous section.

We represent a schedule by a list of n jobs and $m - 1$ *separators*, inspired by [11]. The schedule represented by such a list of $n + m - 1$ elements is as follows: the i'th machine has the jobs between the $(i-1)$'th and i'th separator, in the order of the list. (We assume there is an implicit separator before and after the list.) Between two separators, we thus consider a single machine. We let ‡ denote a separator.

See Fig. 1 for an example with $n = 5$ jobs, $m = 2$ machines of equal speed, no release and setup times, and processing times $p_{j_i} = i$.

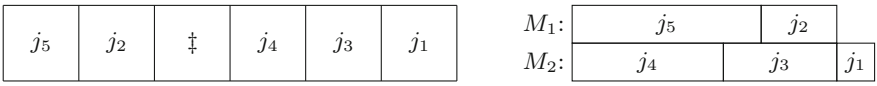

Fig. 1. Example of a list representation (left) of a schedule (right)

Our DD formulation is based on this list representation and has $n+m$ layers, where the outgoing edges of the i'th layer, $1 \leq i \leq n + m - 1$, correspond to making a choice for the i'th item in the list. Between two separators, the formulation is essentially the single-machine DD formulation described in the previous section. We keep track of the current machine i in each node, which is considered as a parameter for the single-machine formulation (for example, so that we can use $p_{i,j}$ as the processing time for job j on this machine). Also, we do not merge nodes with different values of i.

Figure 2 contains a sketch of a possible DD of our formulation for $n = 3$ jobs and $m = 2$ machines. Each of the two columns of nodes corresponds to a machine.

[1] We use a slightly simpler definition, where $L \subseteq \mathcal{J}$.

The four edges between the columns correspond to choosing a separator, whereas the other edges correspond to choosing a job. The DD has $n+m=5$ layers, and the number of separators in any root-terminal path is $m-1=1$.

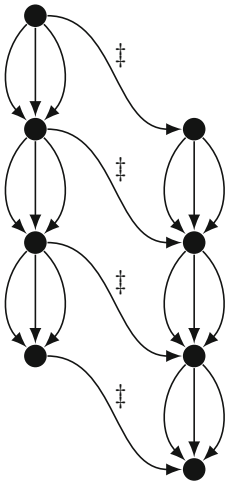

Fig. 2. Example decision diagram with three jobs and two machines. The separator edges are labeled as such.

We now consider our formulation formally. The state in a node is a tuple $S=(i,V,L,f)$ where V,L are sets and i,f are numbers. The root has state $(1,\varnothing,\{\vdash\},0)$. Recall that i is the current machine, V is the set of jobs that are certainly already scheduled, L is the set of jobs that may have been scheduled last, and f is a lower bound on the finishing time of the machine.

To define the feasible set $F(S)$, we first define $X=\mathcal{J}-V$. Then, we define $F(S)$ to be X, additionally with a separator if we are not on the last machine:

$$F(S)=\begin{cases} X\cup\{\ddagger\} & \text{if } i<m \\ X & \text{otherwise.}\end{cases}$$

When making a transition from a node, we define the new state $\overline{S}=(\overline{i},\overline{V},\overline{L},\overline{f})$ as follows. If we choose a job j, we use definitions based on single-machine scheduling:

$$\overline{i}=i,\qquad \overline{V}=V\cup\{j\},\qquad \overline{L}=\{j\},\qquad \overline{f}=\max\{r_j,f+\min_{j'\in L}\sigma_{i,j',j}\}+p_{i,j}$$

If we instead choose a separator, we let:

$$\overline{i}=i+1,\qquad \overline{V}=V,\qquad \overline{L}=\{\vdash\},\qquad \overline{f}=0$$

We pass V along as we still should not schedule these jobs on the new machine. Also, we reset $\overline{f}$ to 0 because the new machine does not contain any jobs yet.

The cost of an edge corresponding to job j is

$$c(S, j) = z_j(\max\{r_j, f + \min_{j' \in L} \sigma_{i,j',j}\} + p_{i,j}),$$

as the operand of z_j is the time job j finishes according to the state information. For a separator, we set $c(S, \ddagger) = 0$. In an exact DD (built top-down without merging), the cost of any root-terminal path is therefore equal to the objective value of the corresponding schedule.

We proceed with proving the validity of the merge operator using Theorem 1. Parts of the ideas below are similar to the single-machine case and can (to some extent) be found in [12].

We define a state relaxation relation $\preceq$ on states S, S' for which $i = i'$ as follows: $S' \preceq S$ means $V' \subseteq V \ \wedge \ L' \supseteq L \ \wedge \ f' \leq f$.

Theorem 2. *This relation satisfies the conditions of being a state relaxation relation (Definition 1).*

Proof. We show each of (R1)–(R4). Let S, S' be states such that $S' \preceq S$. In particular, assume $i = i'$.

For (R1), reflexivity and transitivity follow from that of $\subseteq, \supseteq, \leq$.

For (R2), we need to show $F(S') \supseteq F(S)$. Since $i = i'$, it suffices to show $X' \supseteq X$, where X' denotes the set used in the definition of $F(S')$. Since $V' \subseteq V$, indeed $\mathcal{J} - V' \supseteq \mathcal{J} - V$.

For (R3), let j be a job or separator in $F(S)$. We need to show that $c(S', j) \leq c(S, j)$. If j is a separator, both costs are zero, so the inequality holds. So assume j is a job.

Since the z_j are non-decreasing, we need to show that

$$\max\{r_j, f' + \min_{j' \in L'} \sigma_{j',j}\} + p_{i',j} \leq \max\{r_j, f + \min_{j' \in L} \sigma_{j',j}\} + p_{i,j}.$$

Since $i = i'$, also $p_{i,j} = p_{i',j}$ and hence we only need to show

$$\max\{r_j, f' + \min_{j' \in L'} \sigma_{j',j}\} \leq \max\{r_j, f + \min_{j' \in L} \sigma_{j',j}\}.$$

In turn, it is sufficient to show

$$f' + \min_{j' \in L'} \sigma_{j',j} \leq f + \min_{j' \in L} \sigma_{j',j}.$$

First, we have $f' \leq f$. Also, $L' \supseteq L$, so the minimum is taken over a superset. Hence the inequality holds.

For (R4), define $\overline{S} = \varphi(S, j)$ and $\overline{S'} = \varphi(S', j)$. We have to prove $\overline{S'} \preceq \overline{S}$, which we do by showing each of the relations that $\preceq$ entails.

First, consider the case where we select a job j in $F(S)$.

- $\overline{i'} = i' = i = \overline{i}$ so it is meaningful to prove the claim.
- $\overline{V'} = V' \cup \{j\} \subseteq V \cup \{j\} = \overline{V}$ because $V' \subseteq V$.
- $\overline{L'} = \{j\} = \overline{L}$, so also $L' \supseteq L$.
- $\overline{f'} \leq \overline{f}$ can be shown using the same calculation as for the costs (R3).

Next, consider the case where we select a separator in $F(S)$.

- $\overline{i'} = i' + 1 = i + 1 = \overline{i}$ so it is meaningful to prove the claim.
- $\overline{V'} = V'$ and $\overline{V} = V$ so by assumption, $\overline{V'} \subseteq \overline{V}$.
- $\overline{L'} = \{\vdash\} = \overline{L}$ so also $\overline{L'} \supseteq \overline{L}$.
- $\overline{f'} = 0 = \overline{f}$ so also $\overline{f'} \leq \overline{f}$.

□

We define the merge operator $\oplus$ for states S, S' as for the single-machine case, but we only define it for $i = i'$. In that case,

$$S \oplus S' = (i, V \cap V', L \cup L', \min\{f, f'\}).$$

Using the state relaxation relation $\preceq$, we can utilize Theorem 1 to show that this merge operator is valid.

Theorem 3. *The merge operator $\oplus$ is valid.*

Proof. Consider states S, S' for which $S \oplus S'$ is defined, that is, $i = i'$. We need to show $S \oplus S' \preceq S, S'$. The merged state also has this value for i, so it is meaningful to prove the two $\preceq$ claims.

We indeed have $V \cap V' \subseteq V, V'$ and $L \cup L' \supseteq L, L'$, as well as $\min\{f, f'\} \leq f, f'$. □

5 Merging Heuristics

In the previous section, we have proven the correctness of the proposed DD formulation for uniform machines. The *performance*, i.e., the quality of the bounds, depends on the choice of nodes to merge. Recall that, for correctness, we only merge nodes with the same value of i, i.e., corresponding to the same machine. Among such nodes, however, we have the freedom to choose what nodes to merge, impacting the performance. In this section, we first explain two existing merging heuristics (based on f or the partial objective) and their weakness in our model. We then explain the new merging heuristic that we use.

Ideally, one wants to merge nodes which are going to have identical subtrees, since this way no information is lost, and so would give the best bound (i.e., the optimal objective). However, identifying such nodes is NP-hard [20].

Instead, at least for single-machine scheduling, it seems reasonable to merge nodes which have a large f or a large partial objective [12]. Merging nodes with high partial objective likely does not affect the shortest path (i.e., the bound). Similarly, nodes with a relatively high f may correspond to the machine having

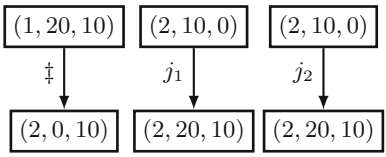

Fig. 3. Part of a DD with two consecutive layers and three nodes in each layer. The state variables depicted are (i, f, z) where z is the partial objective.

much idle time, and so the optimal solution likely does not go through such a node.

In our model, however, one may expect these heuristics to work poorly. Recall that a transition corresponding to a separator sets $\overline{f} = 0$ in the new node. Thus, merging based on f may give priority to such nodes, which does not necessarily make sense.

For example, a path consisting of separators on each of the first few layers would go through several nodes with $f = 0$, which would not get merged. However, selecting multiple separators after each other, thereby leaving machines empty, likely does not result in a good schedule.

Merging based on the partial objective alone may also not be the best choice. Namely, we might merge nodes for which $f = 0$ and nodes for which f is large. In the merged node, the f value is set to the minimum of these f values, which can result in a weak relaxation. We illustrate this with an example.

In Fig. 3, a layer with three nodes is depicted, and a transition from each of these nodes, resulting in three nodes in the next layer. Suppose we need to decide which nodes of this latter layer to merge. If we consider the partial objective values (z), then, because all these three values are equal (and all states have the same value for i), we might decide to merge the first two nodes. The merged node then has $(i, f, z) = (2, 0, 10)$. In particular, the f value is $0 = \min\{0, 20\}$, so that the (relaxed) cost of scheduling a job in this state is likely small. This may result in a weak bound.

Therefore, it seems reasonable to merge nodes with similar values of f. We do this by *partitioning* the nodes on a given layer such that we only merge nodes within the same partition. In the example, the first node (with $f = 0$) should be in a different partition than the other two (with $f = 20$). In that case, we would merge the last two nodes, resulting in state $(2, 20, 10)$, which has a stronger relaxation of f: it is $20 = \min\{20, 20\}$. Within each partition, we merge the nodes with the highest partial objective.

There are two factors influencing the partitioning: the number of partitions, and the range of f values of each partition. We let each partition have a range of equal size; hence, the latter factor reduces to finding a approximation of the type of values of f that may occur in the DD.

We use the following heuristic value for this parameter: construct a schedule and select the largest start time over all jobs. This value is a heuristic estimate of what order of values of f may occur in the DD. For example, if the time scale is multiplied by a factor, this value should also increase with that factor.

More precisely, we use an adaptation of the algorithm by [18], which is a heuristic to find a good schedule by transforming permutations of jobs into schedules. This heuristic is designed for a different scheduling problem; we use the idea by [19] by iteratively selecting the machine for which a job *completes* (rather than starts) earliest. We set $N = 1$ in the heuristic because we do not need the algorithm to spend much time on finding a good solution. Nodes with an f value above the approximation are put in the last partition; the last partition therefore has a larger range than the others.

6 Experiments

In this section, we investigate the effect of partitioning on the quality of the bound. We also compare the bound given by the DDs to that by a linear programming (LP) relaxation of a mixed integer program (MIP). We use the total tardiness $\sum_j \max\{0, e(j) - d_j\}$ as the objective function.

When we compute a gap (that is, the difference between an upper and a lower bound, divided by the upper bound), we use the upper bound of a constraint program (CP, see Fig. 4) found by IBM ILOG CP Optimizer 12.8 after 5 s with default settings. The CP is based on one of the samples provided with this solver. We supply the solution we use for partitioning (the adaptation of [18]) as a warm start to CP Optimizer. Our DD implementation was written in C# (x64, .NET 4.7) and the experiments were run on an Intel Xeon E5-1620 v2 (3.70 GHz, 8 threads), 8 GB RAM, Windows 7. We parallelized our DD implementation, where each thread builds and merges part of each layer, like [9]. We thus also partition the nodes based on the thread they are created in.

$$
\begin{aligned}
&\text{Minimize} \quad \sum_j \max\{0, \text{EndOf}(I_j) - d_j\} \\
&\text{s.t. } I_{j,M} \in \text{Intervals}([\max\{r_j, \sigma_{M,\vdash,j}\}, \infty), p_{j,M}) && (j \in \mathcal{J},\ M \in \mathcal{M}) \\
&\quad I_j \in \{I_{j,M} \mid M \in \mathcal{M}\} && (j \in \mathcal{J}) \\
&\quad \text{NoOverlap}(\{I_{j,M} \mid j \in \mathcal{J}\}, \sigma_{M,\cdot,\cdot}) && (M \in \mathcal{M})
\end{aligned}
$$

Fig. 4. Constraint program

6.1 Instances

We first describe the instances we created. We construct processing times by first drawing an integer uniformly between 5 and 10 inclusive, and then multiplying this integer with $m - 1$. We set the speed of machine M, where $1 \leq M \leq m$, to

$1/(1+q(M-1)/(m-1))$, so that the speeds are between $1/(1+q)$ and 1. (A speed of $1/2$ means all jobs are processed twice as slowly compared to a speed of 1.) The parameter q controls the difference in processing speeds between the machines, and the factor $m-1$ in the processing times ensures that all $p_{i,j}$ are integer.

Setup times are initially drawn uniformly between 0 and $10s$ where s is a parameter. The setup times are then modified by the Floyd-Warshall algorithm to make them satisfy the triangle inequality, as proposed in [14, p. 113]. We use the factor s to increase setup times if the Floyd-Warshall algorithm makes them small.

Release times are drawn uniformly between 0 and $10s$ inclusive for half of the jobs; for the other half, we set them to 0 (so as to make use of the dummy job setup times). For due times, we use the TF/RDD model [3,4], adapted to multiple machines much like [9]. Given $\mathrm{TF}, \mathrm{RDD} \in [0..1]$, due times are drawn uniformly integer between $\lfloor R_j + (1 - \mathrm{TF} - \mathrm{RDD}/2)S \rfloor$ and $\lfloor R_j + (1 - \mathrm{TF} + \mathrm{RDD}/2)S \rfloor$ inclusive (with negative values removed). Here, we let $R_j = \max\{r_j, \sum_M \sigma_{M,\vdash,j}/m\}$ be an approximation of when j can start, and $S = \sum_j p_j / \sum_M \mathrm{speed}_M$ is the total processing time corrected for the speeds of the machines. For a machine M, speed_M is the speed of machine M as described above.

We set $\mathrm{TF} = 0.8, \mathrm{RDD} = 0.2$. The latter two parameters were chosen so that the partial objectives are likely often nonzero, thereby likely improving the merge heuristic (like [9]).

6.2 Partitioning

We first consider the performance of the partitioning based on f. Figure 5 shows the average gap of 25 instances with $n = 60$ jobs and $m = 5$ machines. We set $q = 3$, and $s = 7$ so that the average setup time is about 3.5. Further, we set the width to 8 threads $\cdot$ 5 machines $\cdot$ 500 $= 20,000$. If the width is not divisible by the number of partitions, we divided the width as evenly as possible, giving priority to partitions with a smaller value of f.

We see that partitioning is important: at least 8 partitions are necessary to obtain a nonzero bound. Increasing the number of partitions after that gives better bounds. In particular, not using partitions (i.e., using one partition) gives a zero bound. However, the gap is not monotonically decreasing in the number of partitions.

6.3 Comparison to LPs

We also compare our formulation to the LP relaxation of a time-indexed MIP, solved by IBM ILOG CPLEX 12.8 with default settings. The MIP is based on the one given in [6]. It has a binary variable for each tuple of job, machine and time step, indicating whether a job starts on a machine at a time step. See Fig. 6. We modified the MIP to incorporate setup times; the release times constraint is adapted from [2].

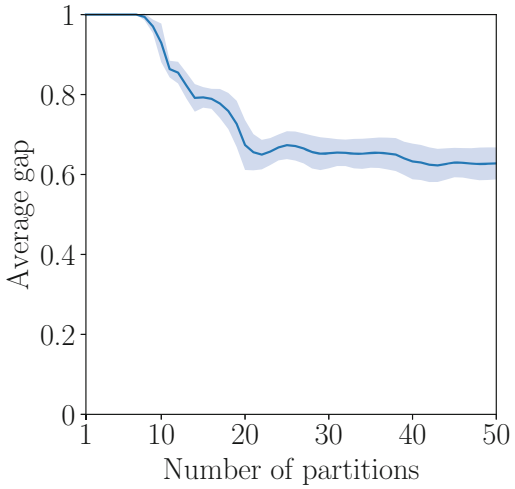

Fig. 5. Average gap with standard deviation of 25 instances as a function of the number of partitions, for $n = 60, m = 5, q = 3, s = 7$

The authors of [6] propose two improvements to the plain LP relaxation of the MIP: preprocessing and cutting planes. However, both require multiple LPs to be solved, whereas our DD model is a single relaxation. We therefore did not use these additions.

Instead, to improve the LP relaxation, we added constraint $(*)$, which is used in the identical machine MIP in [2]. Conceptually, this constraint states that at most m jobs can be processed at any time. While this constraint follows from the others in the MIP used here, we found that it improves the optimum of the LP.

The horizon H in the LP needs to be sufficiently large: if it is too small, the model may be infeasible because not all jobs can be scheduled within the time frame $\{0, \ldots, H-1\}$. However, choosing H too large may yield bad performance because of the large number of variables and constraints. We are not aware of a method to compute the best value of H for our problem.

However, there is a lower bound on H of 1, and an upper bound is induced by the schedule where all jobs are scheduled after each other on the slowest machine, starting at the latest release time (or dummy setup time). We increase horizons between these bounds (divided in 100 steps); when the LP for a horizon becomes feasible, its solution *might* be a lower bound for the MIP (i.e., the scheduling problem), but is not if the horizon is too small. Nevertheless, a larger horizon is guaranteed to result in a smaller LP optimum because the feasible set for such a horizon is a superset of that for a smaller horizon. Thus, any horizon for which the LP is feasible gives an *upper bound* on the lower bound that LP is able to give for the scheduling problem. Equivalently, it gives a lower bound on the gap.

$$\text{Minimize} \quad \sum_j \sum_M \sum_t x_{j,M,t} \max\{0, t + p_{j,M} - d_j\}$$

$$\text{s.t.} \quad \sum_M \sum_t x_{j,M,t} = 1 \qquad (\forall j)$$

$$x_{j,M,t} = 0 \qquad (\forall j,\ \forall M,\ \forall t : t < \max\{r_j, \sigma_{M,\vdash,j}\})$$

$$x_{j,M,t} + \sum_{t \le s < \min\{t + p_{j,M} + \sigma_{M,j,j'}, H\}} x_{j',M,s} \le 1 \qquad (\forall j, j' : j \ne j',\ \forall M,\ \forall t)$$

$$\sum_j \sum_M \sum_{\max\{0, t - p_{j,M} + 1\} \le s \le t} x_{j,M,s} \le m \qquad (\forall t) \qquad (*)$$

$$x_{j,M,t} \ge 0 \qquad (\forall j,\ \forall M,\ \forall t)$$

Fig. 6. LP relaxation of the time-indexed MIP. The notation t denotes a time step $0 \le t < H$.

We report the gap and solve time of the LP with the first (i.e., smallest) horizon we tried that makes the LP feasible.

We tested instances with $n \in \{10, 15, 20\}, m \in \{2, 5\}, q \in \{3, 6\}$. The values of s for the various n are respectively $2, 2.5, 3$, so that the average setup time is about 4.4. For each parameter combination we generated 10 instances. We set a width of 8 threads $\cdot\, m$ machines $\cdot\, 500$ and fix the number of partitions to 100. We used relatively small values of n because the size of the model becomes too large otherwise. We measured time using the `System.Diagnostics.Stopwatch` class so that the LPs and DDs are measured in the same way; we did not count the time of building the LP models. We used the solver with and without presolving, reporting the best result. For the DDs, we also included the time of determining the value on which we base the partitioning, i.e., the time of running the adaptation of [18].

The average gap with standard deviation, with and without constraint $(*)$, is shown in Table 1. We see that the gap of the DDs is smaller than that of the LP. In particular, constraint $(*)$ improves the bound but requires quite some more time. We also provide the bound given by CP Optimizer that we used for obtaining the upper bound (i.e., after 5 s); these gaps are typically between LP without and with constraint $(*)$. It seems that the bound provided by the DDs becomes weaker as the instance grows, a phenomenon also mentioned by [12] (although they consider a harder problem).

We additionally compared to LP relaxations of non-time-indexed MIPs, based instead on binary variables denoting the relative order of pairs of jobs on a machine. Specifically, we used the LP relaxation of MIPs adapted from [13,16] and [1]. While these were solved very quickly, the bounds were always worse than that of the time-indexed LP.

Table 1. Average gap and time (ms) with standard deviation

(n, m, q)	DD		Time-indexed LP without $(*)$		Time-indexed LP with $(*)$		CP
	Gap	Time	Gap	Time	Gap	Time	Gap
$(10, 2, 3)$	0.148 (0.079)	8 (1)	0.869 (0.027)	11 (4)	0.710 (0.041)	42 (9)	0.540 (0.376)
$(10, 2, 6)$	0.142 (0.050)	5 (0)	0.875 (0.037)	18 (3)	0.746 (0.036)	44 (13)	0.566 (0.311)
$(10, 5, 3)$	0.228 (0.053)	6 (0)	0.791 (0.035)	32 (7)	0.622 (0.036)	137 (46)	0.811 (0.031)
$(10, 5, 6)$	0.275 (0.054)	4 (0)	0.864 (0.021)	31 (3)	0.751 (0.033)	145 (20)	0.851 (0.017)
$(15, 2, 3)$	0.309 (0.070)	30 (3)	0.973 (0.014)	47 (14)	0.753 (0.038)	181 (70)	0.959 (0.008)
$(15, 2, 6)$	0.319 (0.063)	19 (1)	0.975 (0.011)	63 (16)	0.825 (0.030)	189 (67)	0.938 (0.005)
$(15, 5, 3)$	0.386 (0.072)	22 (10)	0.927 (0.014)	121 (47)	0.704 (0.022)	1125 (284)	0.935 (0.013)
$(15, 5, 6)$	0.366 (0.057)	35 (3)	0.957 (0.014)	140 (18)	0.848 (0.018)	790 (140)	0.946 (0.010)
$(20, 2, 3)$	0.451 (0.060)	67 (7)	0.990 (0.003)	115 (18)	0.742 (0.045)	701 (142)	0.978 (0.004)
$(20, 2, 6)$	0.458 (0.064)	39 (2)	0.994 (0.006)	146 (40)	0.818 (0.042)	982 (115)	0.966 (0.005)
$(20, 5, 3)$	0.369 (0.058)	138 (10)	0.971 (0.013)	362 (100)	0.739 (0.025)	7998 (1911)	0.969 (0.008)
$(20, 5, 6)$	0.393 (0.045)	109 (23)	0.989 (0.005)	423 (53)	0.882 (0.020)	3198 (935)	0.977 (0.004)

7 Conclusion, Discussion, and Future Work

We proposed a DD formulation for finding lower bounds on a uniform machine scheduling problem, based on separators to identify the machine assignment of jobs. Additionally, we proposed a merge heuristic based on partitioning nodes on a layer, and showed this is important for obtaining a nontrivial bound. Further, we compared to the LP relaxation of a time-indexed MIP, and found the DDs give stronger bounds in less time. We thus conclude that using DDs to obtain lower bounds can contribute to shorter solve times of uniform machine scheduling problems.

The DD formulation proposed in this paper schedules on an "assignment first" basis: we first decide which jobs to schedule on machine M_1, then which jobs to schedule on M_2, etc. In contrast, [9] schedules on a "time first" basis (and the machine assignment is implicit). The latter approach may have advantages, such as being able to model precedence constraints. In our formulation, a precedence constraint $i \rightarrow j$ can be enforced if i is scheduled on an earlier machine than j, but we are unsure how to model precedence constraints in general.

In order to use an efficient "time first" DD formulation, one ideally wants a machine dispatching rule, such as the SPTF rule as used in [9]. An interesting direction for future research is devising such a dispatching rule for uniform machines (i.e., one that is guaranteed to generate an optimal schedule for some permutation of jobs), if one exists.

Another possibility is applying the proposed DD formulation to *unrelated machines*, where the processing times $p_{i,j}$ can depend on i and j in an arbitrary way, and need not necessarily be factored by means of a machine speed. Our DD formulation is directly applicable to this setting; however, the performance is yet to be investigated.

In further performance analysis, it may be interesting to consider alternatives to the LP-relaxation benchmark, such as solving a MIP rather than its LP

(e.g., using branch-and-bound). We expect that this will give better bounds, but will require significantly more computation time than the LP relaxations.

Given that DDs have proven useful in branch-and-bound scheme [8], incorporating our DD formulation in such a search procedure may also improve their performance.

Finally, our DD model may also be applicable to vehicle routing problems, which show similarities to machine scheduling problems [5].

Compliance with Ethical Standards

Declaration of interest: When this work was carried out, the first author had a paid PhD position, 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. Balakrishnan, N., Kanet, J.J., Sridharan, V.: Early/tardy scheduling with sequence dependent setups on uniform parallel machines. Comput. Oper. Res. **26**(2), 127–141 (1999)
2. Baptiste, P., Jouglet, A., Savourey, D.: Lower bounds for parallel machine scheduling problems. Int. J. Oper. Res. **3**(6), 643–664 (2008)
3. Beasley, J.E.: Weighted tardiness (OR library) (2018). http://people.brunel.ac.uk/~mastjjb/jeb/orlib/wtinfo.html, originally described in [4]. Accessed 26 Feb 2018
4. Beasley, J.E.: OR-Library: distributing test problems by electronic mail. J. Oper. Res. Soc. **41**(11), 1069–1072 (1990)
5. Beck, J.C., Prosser, P., Selensky, E.: Vehicle routing and job shop scheduling: what's the difference? In: ICAPS, pp. 267–276 (2003)
6. Berghman, L., Spieksma, F., T'Kindt, V.: Solving a time-indexed formulation by preprocessing and cutting planes (2014). https://doi.org/10.2139/ssrn.2437371, Available at SSRN (First appeared as extended abstract in the 6th Multidisciplinary International Scheduling Conference: Theory & Applications)
7. Bergman, D., Cire, A.A., van Hoeve, W.J., Hooker, J.: Decision Diagrams for Optimization. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-42849-9
8. Bergman, D., Cire, A.A., van Hoeve, W.J., Hooker, J.N.: Discrete optimization with decision diagrams. INFORMS J. Comput. **28**(1), 47–66 (2016)
9. van den Bogaerdt, P., de Weerdt, M.M.: Multi-machine scheduling lower bounds using decision diagrams. Oper. Res. Lett. **46**(6), 616–621 (2018)
10. Cire, A.A., Van Hoeve, W.J.: Multivalued decision diagrams for sequencing problems. Oper. Res. **61**(6), 1411–1428 (2013)
11. Driessel, R., Mönch, L.: Variable neighborhood search approaches for scheduling jobs on parallel machines with sequence-dependent setup times, precedence constraints, and ready times. Comput. Ind. Eng. **61**(2), 336–345 (2011)
12. Hooker, J.N.: Job sequencing bounds from decision diagrams. In: Beck, J.C. (ed.) CP 2017. LNCS, vol. 10416, pp. 565–578. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-66158-2_36
13. Jain, V., Grossmann, I.E.: Algorithms for hybrid MILP/CP models for a class of optimization problems. INFORMS J. Comput. **13**(4), 258–276 (2001)

14. Jordan, C.: Batching and Scheduling: Models and Methods for Several Problem Classes. Lecture Notes in Economics and Mathematical Systems, vol. 437. Springer, Heidelberg (1996). https://doi.org/10.1007/978-3-642-48403-2
15. Lenstra, J.K., Kan, A.R., Brucker, P.: Complexity of machine scheduling problems. Ann. Discrete Math. **1**, 343–362 (1977)
16. Lin, Y.K., Hsieh, F.Y.: Unrelated parallel machine scheduling with setup times and ready times. Int. J. Prod. Res. **52**(4), 1200–1214 (2014)
17. Pinedo, M.L.: Scheduling: Theory, Algorithms, and Systems, 4th edn. Springer, New York (2016). https://doi.org/10.1007/978-3-319-26580-3
18. Rodrigues, R., Pessoa, A., Uchoa, E., Poggi de Aragão, M.: Heuristic algorithm for the parallel machine total weighted tardiness scheduling problem. Relat. Pesquisa Engenh. Produçao **8**(10), 1–11 (2008)
19. Schutten, J.M.J.: List scheduling revisited. Oper. Res. Lett. **18**(4), 167–170 (1996)
20. Van Hoeve, W.J.: Decision diagrams for sequencing and scheduling (2016). http://icaps16.icaps-conference.org/proceedings/tutorials/tutorial5.pdf, Tutorial ICAPS 2016. Accessed 8 Nov 2018