# Chapter 11
# Sequencing and Single-Machine Scheduling

**Abstract** In this chapter we provide an in-depth study of representing and handling single-machine scheduling and sequencing problems with decision diagrams. We provide exact and relaxed MDD representations, together with MDD filtering algorithms for various side constraints, including time windows, precedence constraints, and sequence-dependent setup times. We extend a constraint-based scheduling solver with these techniques, and provide an experimental evaluation for a wide range of problems, including the traveling salesman problem with time windows, the sequential ordering problem, and minimum-tardiness sequencing problems. The results demonstrate that MDD propagation can improve a state-of-the-art constraint-based scheduler by orders of magnitude in terms of solving time.

## 11.1 Introduction

Sequencing problems are among the most widely studied problems in operations research. Specific variations of sequencing problems include single-machine scheduling, the traveling salesman problem with time windows, and precedence-constrained machine scheduling. Sequencing problems are those where the best order for performing a set of tasks must be determined, which in many cases leads to an NP-hard problem [71, Section A5]. Sequencing problems are prevalent in manufacturing and routing applications, including production plants where jobs should be processed one at a time on an assembly line, and in mail services where packages must be scheduled for delivery on a vehicle. Industrial problems that involve multiple facilities may also be viewed as sequencing problems in certain scenarios, e.g., when



D. Bergman et al., *Decision Diagrams for Optimization*, Artificial Intelligence: Foundations, Theory, and Algorithms, DOI 10.1007/978-3-319-42849-9_11

a machine is the bottleneck of a manufacturing plant [128]. Existing methods for sequencing problems either follow a dedicated heuristic for a specific problem class or utilize a generic solving methodology such as integer programming or constraint programming.

In this chapter we present a new approach for solving sequencing problems, based on MDDs. We argue that relaxed MDDs can be particularly useful as a discrete relaxation of the feasible set of sequencing problems. We focus on a broad class of sequencing problems where jobs should be scheduled on a single machine and are subject to precedence and time window constraints, and in which setup times can be present. It generalizes a number of single-machine scheduling problems and variations of the traveling salesman problem (TSP). The relaxation provided by the MDD, however, is suitable for any problem where the solution is defined by a permutation of a fixed number of tasks, and it does not directly depend on particular constraints or on the objective function.

The structure of this chapter is as follows: We first introduce a representation of the feasible set of a sequencing problem as an MDD, and show how we can obtain a relaxed MDD. We then show how the relaxed MDD can be used to compute bounds on typical objective functions in scheduling, such as the makespan and total tardiness. Moreover, we describe how to derive more structured sequencing information from the relaxed MDD, in particular a valid set of precedence relations that must hold in any feasible solution.

We also propose a number of techniques for strengthening the MDD relaxation, which take into account the precedence and time window constraints. We demonstrate that these generic techniques can be used to derive a polynomial-time algorithm for a particular TSP variant introduced by [14] by showing that the associated MDD has polynomial size.

To demonstrate the use of relaxed MDDs in practice, we apply our techniques to *constraint-based scheduling* [17]. Constraint-based scheduling plays a central role as a general-purpose methodology in complex and large-scale scheduling problems. Examples of commercial applications that apply this methodology include yard planning of the Singapore port and gate allocation of the Hong Kong airport [68], Brazilian oil-pipeline scheduling [112], and home healthcare scheduling [136]. We show that, by using the relaxed MDD techniques described here, we can improve the performance of the state-of-the-art constraint-based schedulers by orders of magnitude on single-machine problems without losing the generality of the method.

## 11.2 Problem Definition

As mentioned above, we focus on generic sequencing problems, presented here in terms of 'unary machine' scheduling. Note that a machine may refer to any resource capable of handling at most one activity at a time.

Let $\mathscr{J} = \{j_1, \ldots, j_n\}$ be a set of $n$ jobs to be processed on a machine that can perform at most one job at a time. Each job $j \in \mathscr{J}$ has an associated *processing time* $p_j$, which is the number of time units the job requires from the machine, and a *release date* $r_j$, the time from which job $j$ is available to be processed. For each pair of distinct jobs $j, j' \in \mathscr{J}$ a *setup time* $t_{j,j'}$ is defined, which indicates the minimum time that must elapse between the end of $j$ and the beginning of $j'$ if $j'$ is the first job processed after $j$ finishes. We assume that jobs are *non-preemptive*, i.e., we cannot interrupt a job while it is being processed on the machine.

We are interested in assigning a *start time* $s_j \geq r_j$ for each job $j \in \mathscr{J}$ such that job processing intervals do not overlap, the resulting schedule observes a number of constraints, and an objective function $f$ is minimized. Two types of constraints are considered in this chapter: *precedence constraints*, requiring that $s_j \leq s_{j'}$ for certain pairs of jobs $(j, j') \in \mathscr{J} \times \mathscr{J}$, which we equivalently write $j \ll j'$; and *time window constraints*, where the *completion time* $c_j = s_j + p_j$ of each job $j \in \mathscr{J}$ must be such that $c_j \leq d_j$ for some *deadline* $d_j$. Furthermore, we study three representative objective functions in scheduling: the *makespan*, where we minimize the completion time of the schedule, or $\max_{j \in \mathscr{J}} c_j$; the *total tardiness*, where we minimize $\sum_{j \in \mathscr{J}} (\max\{0,\ c_j - \delta_j\})$ for given *due dates* $\delta_j$; and the *sum of setup times*, where we minimize the value obtained by accumulating the setup times $t_{j,j'}$ for all consecutive jobs $j, j'$ in a schedule. Note that for these objective functions we can assume that jobs should always be processed as early as possible (i.e., idle times do not decrease the value of the objective function).

Since jobs are processed one at a time, any solution to such scheduling problem can be equivalently represented by a total ordering $\pi = (\pi_1, \pi_2, \ldots, \pi_n)$ of $\mathscr{J}$. The start time of job $j$ implied by $\pi$ is given by $s_j = r_j$ if $j = \pi_1$, and $s_j = \max\{r_j,\ s_{\pi_{i-1}} + p_{\pi_{i-1}} + t_{\pi_{i-1},j}\}$ if $j = \pi_i$ for some $i \in \{2, \ldots, n\}$. We say that an ordering $\pi$ of $\mathscr{J}$ is *feasible* if the implied job times observe the precedence and time window constraints, and *optimal* if it is feasible and minimizes $f$.

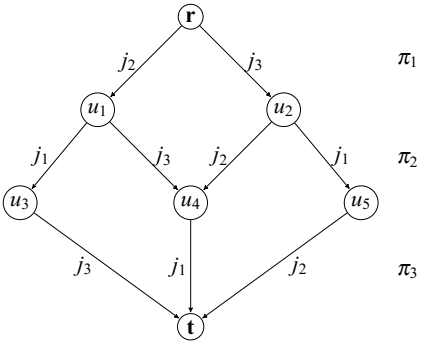
*Job parameters*

| Job | Release ($r_j$) | Deadline ($d_j$) | Processing ($p_j$) |
|---|---|---|---|
| $j_1$ | 2 | 20 | 3 |
| $j_2$ | 0 | 14 | 4 |
| $j_3$ | 1 | 14 | 2 |

*Setup times*

| | $j_1$ | $j_2$ | $j_3$ |
|---|---|---|---|
| $j_1$ | - | 3 | 2 |
| $j_2$ | 3 | - | 1 |
| $j_3$ | 1 | 2 | - |

(a) Instance data.




(b) MDD.

**Fig. 11.1** Example of an MDD for a sequencing problem.

## 11.3 MDD Representation

We have already seen examples of single-machine scheduling in Section 3.8 and Section 8.3, and we will follow a similar MDD representation in this chapter. That is, we define an MDD $\mathscr{M}$ whose paths represent the feasible orderings of $\mathscr{J}$. The set of nodes of $\mathscr{M}$ are partitioned into $n+1$ layers $L_1,\ldots,L_{n+1}$, where layer $L_i$ corresponds to the $i$-th position $\pi_i$ of the feasible orderings encoded by $\mathscr{M}$, for $i=1,\ldots,n$. Layers $L_1$ and $L_{n+1}$ are singletons representing the root **r** and the terminal **t**, respectively. In this chapter, an arc $a=(u,v)$ of $\mathscr{M}$ is always directed from a source node $u$ in some layer $L_i$ to a target node $v$ in the *subsequent* layer $L_{i+1}$, $i\in\{1,\ldots,n\}$. We write $\ell(a)$ to indicate the layer of the source node $u$ of the arc $a$ (i.e., $u\in L_{\ell(a)}$).

With each arc $a$ of $\mathscr{M}$ we associate a label $d(a)\in\mathscr{J}$ that represents the assignment of the job $d(a)$ to the $\ell(a)$-th position of the orderings identified by the paths traversing $a$. Hence, an arc-specified path $(a_1,\ldots,a_n)$ from **r** to **t** identifies the ordering $\pi=(\pi_1,\ldots,\pi_n)$, where $\pi_i=d(a_i)$ for $i=1,\ldots,n$. Every feasible ordering is identified by some path from **r** to **t** in $\mathscr{M}$, and conversely every path from **r** to **t** identifies a feasible ordering.

*Example 11.1.* We provide an MDD representation for a sequencing problem with three jobs $j_1$, $j_2$, and $j_3$. The instance data are presented in Fig. 11.1(a), and the associated MDD $\mathscr{M}$ is depicted in Fig. 11.1(b). No precedence constraints are considered. There are four feasible orderings in total, each identified by a path from **r** to **t** in $\mathscr{M}$. In particular, the path traversing nodes **r**, $u_2$, $u_4$, and **t** represents a

solution where jobs $j_3$, $j_2$, and $j_1$ are performed in this order. The completion times for this solution are $c_{j_1} = 15$, $c_{j_2} = 9$, and $c_{j_3} = 3$. Note that we can never have a solution where $j_1$ is first on the machine, otherwise either the deadline of $j_2$ or $j_3$ would be violated. Hence, there is no arc $a$ with $d(a) = j_1$ directed out of $\mathbf{r}$.

We next show how to compute the orderings that yield the optimal makespan and the optimal sum of setup times in polynomial time in the size of $\mathscr{M}$. For the case of total tardiness and other similar objective functions, we are able to provide a lower bound on its optimal value also in polynomial time in $\mathscr{M}$.

- *Makespan*. For each arc $a$ in $\mathscr{M}$, define the *earliest completion time* of $a$, or $ect_a$, as the minimum completion time of the job $d(a)$ among all orderings that are identified by the paths in $\mathscr{M}$ containing $a$. If the arc $a$ is directed out of $\mathbf{r}$, then $a$ assigns the first job that is processed in such orderings, thus $ect_a = r_{d(a)} + p_{d(a)}$. For the remaining arcs, recall that the completion time $c_{\pi_i}$ of a job $\pi_i$ depends only on the completion time of the previous job $\pi_{i-1}$, the setup time $t_{\pi_{i-1},\pi_i}$, and on the specific job parameters; namely, $c_{\pi_i} = \max\{r_{\pi_i},\ c_{\pi_{i-1}} + t_{\pi_{i-1},\pi_i}\} + p_{\pi_i}$. It follows that the earliest completion time of an arc $a = (u,v)$ can be computed by the relation

$$ect_a = \max\{r_{d(a)},\ \min\{ect_{a'} + t_{d(a'),d(a)} : a' \in in(u)\}\} + p_{d(a)}. \tag{11.1}$$

  The minimum makespan is given by $\min_{a \in in(\mathbf{t})} ect_a$, as the arcs directed to $\mathbf{t}$ assign the last job in all orderings represented by $\mathscr{M}$. An optimal ordering can be obtained by recursively retrieving the minimizer arc $a' \in in(u)$ in the "min" of (11.1).
- *Sum of Setup Times*. The minimum sum of setup times is computed analogously: For an arc $a = (u,v)$, let $st_a$ represent the minimum sum of setup times up to job $d(a)$ among all orderings that are represented by the paths in $\mathscr{M}$ containing $a$. If $a$ is directed out of $\mathbf{r}$, we have $st_a = 0$; otherwise,

$$st_a = \min\{st_{a'} + t_{d(a'),d(a)} : a' \in in(u)\}. \tag{11.2}$$

  The minimum sum of setup times is given by $\min_{a \in in(\mathbf{t})} st_a$.
- *Total Tardiness*. The *tardiness* of a job $j$ is defined by $\max\{0,\ c_j - \delta_j\}$ for some due date $\delta_j$. Unlike the previous two cases, the tardiness value that a job attains in an optimal solution depends on the sequence of all activities, not only on its individual contribution or the value of its immediate predecessor. Nonetheless, as

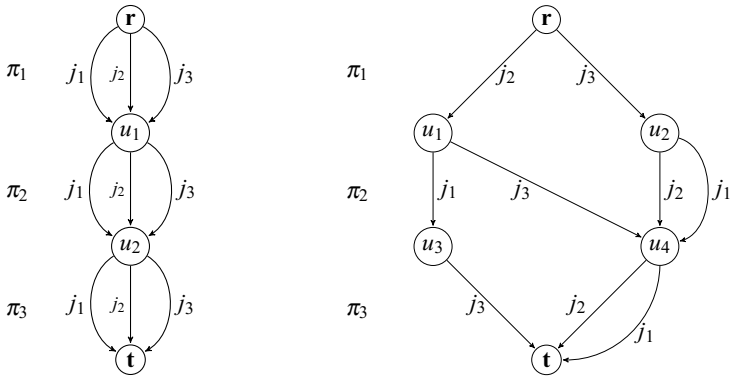
the tardiness function for a job is nondecreasing in its completion time, we can utilize the earliest completion time as follows: For any arc $a = (u, v)$, the value $\max\{0,\ ect_a - \delta_{d(a)}\}$ yields a lower bound on the tardiness of the job $d(a)$ among all orderings that are represented by the paths in $\mathscr{M}$ containing $a$. Hence, a lower bound on the total tardiness is given by the length of the shortest path from $\mathbf{r}$ to $\mathbf{t}$, where the length of an arc $a$ is set to $\max\{0,\ ect_a - \delta_{d(a)}\}$. Observe that this bound is tight if the MDD is composed by a single path.

We remark that valid bounds for many other types of objective in the scheduling literature can be computed in an analogous way as above. For example, suppose the objective is to minimize $\sum_{j\in\mathscr{J}} f_j(c_j)$, where $f_j$ is a function defined for each job $j$ and which is nondecreasing in the completion time $c_j$. Then, as in total tardiness, the value $f_{d(a)}(ect_a)$ for an arc $a = (u, v)$ yields a lower bound on the minimum value of $f_{d(a)}(c_{d(a)})$ among all orderings that are identified by the paths in $\mathscr{M}$ containing $a$. Using such bounds as arc lengths, the shortest path from $\mathbf{r}$ to $\mathbf{t}$ represents a lower bound on $\sum_{j\in\mathscr{J}} f_j(c_j)$. This bound is tight if $f_j(c_j) = c_j$, or if $\mathscr{M}$ is composed by a single path. Examples of such objectives include weighted total tardiness, total square tardiness, sum of (weighted) completion times, and number of late jobs.

*Example 11.2.* In the instance depicted in Fig. 11.1, we can apply the recurrence relation (11.1) to obtain $ect_{\mathbf{r},u_1} = 4$, $ect_{\mathbf{r},u_2} = 3$, $ect_{u_1,u_3} = 10$, $ect_{u_1,u_4} = 7$, $ect_{u_2,u_4} = 9$, $ect_{u_2,u_5} = 7$, $ect_{u_3,\mathbf{t}} = 14$, $ect_{u_4,\mathbf{t}} = 11$, and $ect_{u_5,\mathbf{t}} = 14$. The optimal makespan is $\min\{ect_{u_3,\mathbf{t}}, ect_{u_4,\mathbf{t}}, ect_{u_5,\mathbf{t}}\} = ect_{u_4,\mathbf{t}} = 11$; it corresponds to the path $(\mathbf{r}, u_1, u_4, \mathbf{t})$, which identifies the optimal ordering $(j_2, j_3, j_1)$. The same ordering also yields the optimal sum of setup times with a value of 2.

Suppose now that we are given due dates $\delta_{j_1} = 13$, $\delta_{j_2} = 8$, and $\delta_{j_3} = 3$. The length of an arc $a$ is given by $l_a = \max\{0,\ ect_a - \delta_{d(a)}\}$, as described earlier. We have $l_{u_1,u_4} = 4$, $l_{u_2,u_4} = 1$, $l_{u_3,\mathbf{t}} = 11$, and $l_{u_5,\mathbf{t}} = 6$; all remaining arcs $a$ are such that $l_a = 0$. The shortest path in this case is $(\mathbf{r}, u_2, u_4, \mathbf{t})$ and has a value of 1. The minimum tardiness, even though it is given by the ordering identified by this same path, $(j_3, j_2, j_1)$, has a value of 3.

The reason for this gap is that the ordering with minimum tardiness does not necessarily coincide with the schedule corresponding to the earliest completion time. Namely, we computed $l_{u_4,\mathbf{t}} = 0$ considering $ect_{u_4,\mathbf{t}} = 11$, since the completion time of the job $d(u_4, \mathbf{t}) = j_1$ is 11 in $(j_2, j_3, j_1)$. However, in the optimal ordering $(j_3, j_2, j_1)$ for total tardiness, the completion time of $j_1$ would be 15; this solution yields a better cost than $(j_2, j_3, j_1)$ due to the reduction in the tardiness of $j_3$.




(a) 1-width relaxation. (b) 2-width relaxation.

**Fig. 11.2** Two relaxed MDDs for the sequencing problem in Fig. 11.1.

## 11.4 Relaxed MDDs

We next consider the compilation of relaxed MDDs for sequencing problems, which represent a superset of the feasible orderings of $\mathscr{J}$. As an illustration, Fig. 11.2(a) and 11.2(b) present two examples of a relaxed MDD with maximum width $W = 1$ and $W = 2$, respectively, for the problem depicted in Fig. 11.1. In particular, the MDD in Fig. 11.2(a) encodes all the orderings represented by permutations of $\mathscr{J}$ with repetition, hence it trivially contains the feasible orderings of any sequencing problem. It can be generally constructed as follows: We create one node $u_i$ for each layer $L_i$ and connect the pair of nodes $u_i$ and $u_{i+1}$, $i = 1, \ldots, n$, with arcs $a_1, \ldots, a_n$ such that $d(a_l) = j_k$ for each job $j_k$.

It can also be verified that the MDD in Fig. 11.2(b) contains all the feasible orderings of the instance in Fig. 11.1. However, the rightmost path going through nodes **r**, $u_2$, $u_4$, and **t** identifies an ordering $\pi = (j_3, j_1, j_1)$, which is infeasible as job $j_1$ is assigned twice in $\pi$.

The procedures in Section 11.3 for computing the optimal makespan and the optimal sum of setup times now yield a lower bound on such values when applied to a relaxed MDD, since all feasible orderings of $\mathscr{J}$ are encoded in the diagram. Moreover, the lower bounding technique for total tardiness remains valid.

Considering that a relaxed MDD $\mathscr{M}$ can be easily constructed for any sequencing problem (e.g., the 1-width relaxation of Fig. 11.2(a)), we can now apply the techniques presented in Section 4.7 and Chapter 9 to incrementally modify $\mathscr{M}$ in order to strengthen the relaxation it provides, while observing the maximum width

$W$. Under certain conditions, we obtain the reduced MDD representing exactly the feasible orderings of $\mathscr{J}$, provided that $W$ is sufficiently large.

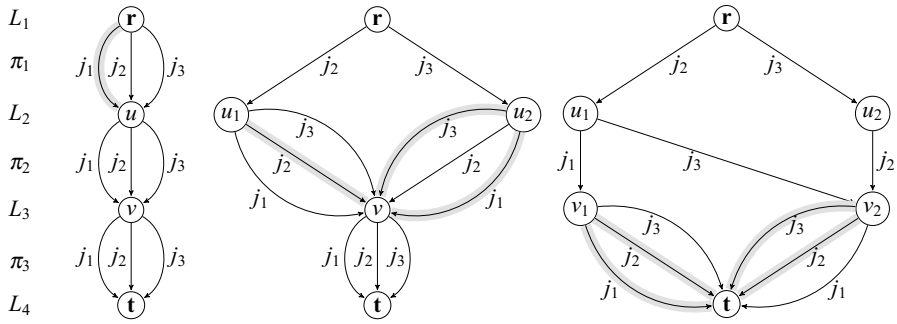
Recall that we modify a relaxed MDD $\mathscr{M}$ by applying the operations of *filtering* and *refinement*, which aim at approximating $\mathscr{M}$ to an *exact* MDD, i.e., one that exactly represents the feasible orderings of $\mathscr{J}$. We revisit these concepts below, and describe them in the context of sequencing problems:

- *Filtering.* An arc $a$ in $\mathscr{M}$ is *infeasible* if all the paths in $\mathscr{M}$ containing $a$ represent orderings that are not feasible. Filtering consists of identifying infeasible arcs and removing them from $\mathscr{M}$, which would hence eliminate one or more infeasible orderings that are encoded in $\mathscr{M}$. We will provide details on the filtering operation in Section 11.5.
- *Refinement.* A relaxed MDD can be intuitively perceived as a diagram obtained by merging nonequivalent nodes of an exact MDD for the problem. Refinement consists of identifying these nodes in $\mathscr{M}$ that are encompassing multiple equivalence classes, and *splitting* them into two or more new nodes to represent such classes more accurately (as long as the maximum width $W$ is not violated). In particular, a node $u$ in layer $L_i$ can be split if there exist two partial orderings $\pi'_1$, $\pi'_2$ identified by paths from $\mathbf{r}$ to $u$ such that, for some $\pi^* = (\pi_i, \ldots, \pi_n)$, $(\pi'_1, \pi^*)$ is a feasible ordering while $(\pi'_2, \pi^*)$ is not. If this is the case, then the partial paths in $\mathscr{M}$ representing such orderings must end in different nodes of the MDD, which will be necessarily nonequivalent by definition. We will provide details on the refinement operation in Section 11.7.

Observe that, if a relaxed MDD $\mathscr{M}$ does not have any infeasible arcs and no nodes require splitting, then by definition $\mathscr{M}$ is exact. However, it may not necessarily be reduced.

As mentioned in Chapter 9, filtering and refinement are independent operations that can be applied to $\mathscr{M}$ in any order that is suitable for the problem at hand. In this chapter we assume a top-down approach: We traverse layers $L_2, \ldots, L_{n+1}$ one at a time in this order. At each layer $L_i$, we first apply filtering to remove infeasible arcs that are directed to the nodes in $L_i$. After the filtering is complete, we perform refinement to split the nodes in layer $L_i$ as necessary, while observing the maximum width $W$.

*Example 11.3.* Figure 11.3 illustrates the top-down application of filtering and refinement for layers $L_2$ and $L_3$. Assume a scheduling problem with three jobs $\mathscr{J} = \{j_1, j_2, j_3\}$ and subject to a single precedence constraint stating that job $j_2$




(a) Initial relaxation. (b) After processing $L_2$. (c) After processing $L_3$.

**Fig. 11.3** Example of filtering and refinement. The scheduling problem is such that job $j_2$ must precede $j_1$ in all feasible orderings. Shaded arrows represent infeasible arcs detected by the filtering.

must precede job $j_1$. The initial relaxed MDD is a 1-width relaxation, depicted in Fig. 11.3(a). Our maximum width is set to $W = 2$.

We start by processing the incoming arcs at layer $L_2$. The filtering operation detects that the arc $a \in in(u)$ with $d(a) = j_1$ is infeasible, otherwise we will have an ordering starting with job $j_1$, violating the precedence relation. Refinement will split node $u$ into nodes $u_1$ and $u_2$, since for any feasible ordering starting with job $j_2$, i.e., $(j_2, \pi')$ for some $\pi'$, the ordering $(j_3, \pi')$ is infeasible as it will necessarily assign job $j_3$ twice. The resulting MDD is depicted in Fig. 11.3(b). Note that, when a node is split, we replicate its outgoing arcs to each of the new nodes.

We now process the incoming arcs at layer $L_3$. The filtering operation detects that the arc with label $j_2$ directed out of $u_1$ and the arc with label $j_3$ directed out of $u_2$ are infeasible, since the corresponding paths from **r** to $v$ would yield orderings that assign some job twice. The arc with label $j_1$ leaving node $u_2$ is also infeasible, since we cannot have any ordering with prefix $(j_3, j_1)$. Finally, refinement will split node $v$ into nodes $v_1$ and $v_2$; note in particular that the feasible orderings prefixed by $(j_2, j_3)$ and $(j_3, j_2)$ have the same completions, namely $(j_1)$, therefore the corresponding paths end at the same node $v_1$. The resulting MDD is depicted in Fig. 11.3(c). We can next process the incoming arcs at layer $L_4$, and remove arcs with labels $j_1$ and $j_2$ out of $v_1$, and arcs with labels $j_2$ and $j_3$ out of $v_2$.

## 11.5 Filtering

We next apply the methodology developed in Chapter 9 to sequencing problems. That is, for each constraint type of our problem, we identify necessary conditions for the infeasibility of an arc in $\mathscr{M}$. To this end, for each constraint type $\mathscr{C}$, we equip the nodes and arcs of $\mathscr{M}$ with state information that is specific to $\mathscr{C}$.

We take care that the conditions for infeasibility can be tested in polynomial time in the size of the relaxed MDD $\mathscr{M}$. Namely, we restrict our state definitions to have size $O(|\mathscr{J}|)$ and to be Markovian, in that they only depend on the states of the nodes and arcs in the adjacent layers. Thus, the states can be computed simultaneously with the filtering and refinement operations during the top-down approach described in Section 11.4. We also utilize additional state information that is obtained through an extra bottom-up traversal of the MDD and that, when combined with the top-down states, leads to stronger tests.

### *11.5.1 Filtering Invalid Permutations*

The feasible orderings of any sequencing problem are permutations of $\mathscr{J}$ without repetition, which can be enforced with the constraint $\textsc{AllDifferent}(\pi_1,\ldots,\pi_n)$. Hence, we can directly use the filtering conditions for this constraint described in Section 4.7.1, based on the states $All_u^{\downarrow} \subseteq \mathscr{J}$ and $Some_u^{\downarrow} \subseteq \mathscr{J}$ for each node $u$ of $\mathscr{M}$. Recall that the state $All_u^{\downarrow}$ is the set of arc labels that appear in *all* paths from the root node $\mathbf{r}$ to $u$, while the state $Some_u^{\downarrow}$ is the set of arc labels that appear in *some* path from the root node $\mathbf{r}$ to $u$. As presented in Section 4.7.1, an arc $a=(u,v)$ is infeasible if either $d(a) \in All_u^{\downarrow}$ (condition 4.6) or $|Some_u^{\downarrow}| = \ell(a)$ and $d(a) \in Some_u^{\downarrow}$ (condition 4.7).

We also equip the nodes with additional states that can be derived from a bottom-up perspective of the MDD. Namely, we define two new states $All_u^{\uparrow} \subseteq \mathscr{J}$ and $Some_u^{\uparrow} \subseteq \mathscr{J}$ for each node $u$ of $\mathscr{M}$. They are similar to the states $All_u^{\downarrow}$ and $Some_u^{\downarrow}$, but now they are computed with respect to the paths from $\mathbf{t}$ to $u$ instead of the paths from $\mathbf{r}$ to $u$, and analogously computed recursively.

It follows from Section 4.7.1 that an arc $a=(u,v)$ is infeasible if we have either $d(a) \in All_v^{\uparrow}$ (condition 4.10), $|Some_v^{\uparrow}| = n-\ell(a)$ and $d(a) \in Some_v^{\uparrow}$ (condition 4.11), or $|Some_u^{\downarrow} \cup \{d(a)\} \cup Some_v^{\uparrow}| < n$ (condition 4.12).

### 11.5.2 Filtering Precedence Constraints

We next consider filtering a given set of precedence constraints, where we write $j \ll j'$ if a job $j$ should precede job $j'$ in any feasible ordering. We assume the precedence relations are not trivially infeasible, i.e., there are no cycles of the form $j \ll j_1 \ll \cdots \ll j_m \ll j$. We can apply the same states defined for the ALLDIFFERENT constraint in Section 11.5.1 for this particular case.

**Lemma 11.1.** *An arc $a = (u,v)$ with label $d(a)$ is infeasible if either of the following conditions hold:*

$$\exists\, j \in (\mathscr{J} \setminus Some_u^{\downarrow}) \;\; s.t. \;\; j \ll d(a), \tag{11.3}$$

$$\exists\, j \in (\mathscr{J} \setminus Some_v^{\uparrow}) \;\; s.t. \;\; d(a) \ll j. \tag{11.4}$$

*Proof.* Let $\pi'$ be any partial ordering identified by a path from $\mathbf{r}$ to $u$, and consider (11.3). By definition of $Some_u^{\downarrow}$, we have that any job $j$ in the set $\mathscr{J} \setminus Some_u^{\downarrow}$ is not assigned to any position in $\pi'$. Thus, if any such job $j$ must precede $d(a)$, then all orderings prefixed by $(\pi', d(a))$ will violate this precedence constraint, and the arc is infeasible. The condition (11.4) is the symmetrical version of (11.3). □

### 11.5.3 Filtering Time Window Constraints

Consider now that a deadline $d_j$ is imposed for each job $j \in \mathscr{J}$. With each arc $a$ we associate the state $ect_a$ as defined in Section 11.3: It corresponds to the minimum completion time of the job in the $\ell(a)$-th position among all orderings that are identified by paths in $\mathscr{M}$ containing the arc $a$. As in relation (11.1), the state $ect_a$ for an arc $a = (u,v)$ is given by the recurrence

$$ect_a = \begin{cases} r_{d(a)} + p_{d(a)} & \text{if } a \in out(\mathbf{r}), \\ \max\{r_{d(a)},\, \min\{ect_{a'} + t_{d(a'),d(a)} : a' \in in(u), d(a) \neq d(a')\}\} + p_{d(a)} & \\ & \text{otherwise.} \end{cases}$$

Here we added the trivial condition $d(a) \neq d(a')$ to strengthen the bound on $ect_a$ in the relaxed MDD $\mathscr{M}$. We could also include the condition $d(a) \not\ll d(a')$ if precedence constraints are imposed over $d(a)$.

We next consider a symmetrical version of $ect_a$ to derive a necessary infeasibility condition for time window constraints. Namely, with each arc $a$ we associate the state $lst_a$, which represents the *latest start time* of $a$: For all orderings that are identified by paths in $\mathscr{M}$ containing the arc $a$, the value $lst_a$ corresponds to an upper bound on the maximum start time of the job in the $\ell(a)$-th position so that no deadlines are violated in such orderings. The state $lst_a$ for an arc $a = (u,v)$ is given by the following recurrence, which can be computed through a single bottom-up traversal of $\mathscr{M}$:

$$lst_a = \begin{cases} d_{d(a)} - p_{d(a)} & \text{if } a \in in(\mathbf{t}), \\ \min\{d_{d(a)},\ \max\{lst_{a'} - t_{d(a),d(a')} : a' \in out(v), d(a) \neq d(a')\}\} - p_{d(a)} & \\ & \text{otherwise.} \end{cases}$$

We combine $ect_a$ and $lst_a$ to derive the following rule:

**Lemma 11.2.** *An arc $a = (u,v)$ is infeasible if*

$$ect_a > lst_a + p_{d(a)}. \tag{11.5}$$

*Proof.* The value $lst_a + p_{d(a)}$ represents an upper bound on the maximum time the job $d(a)$ can be completed so that no deadlines are violated in the orderings identified by paths in $\mathscr{M}$ containing $a$. Since $ect_a$ is the minimum time that job $d(a)$ will be completed among all such orderings, no feasible ordering identified by a path traversing $a$ exists if rule (11.5) holds. □

### *11.5.4 Filtering Objective Function Bounds*

As described in Section 9.2, in constraint programming systems the objective function is treated as a constraint $z \leq z^*$, where $z$ represents the objective function, and $z^*$ is an upper bound of the objective function value. The upper bound typically corresponds to the best feasible solution found during the search for an optimal solution.

Below we describe filtering procedures for the 'objective constraint' for sequencing problems. Given $z^*$, an arc $a$ is infeasible with respect to the objective if all paths in $\mathscr{M}$ that contain $a$ have objective value greater than $z^*$. However, the associated filtering method depends on the form of the objective function. We consider here

three types of objectives: minimize makespan, minimize the sum of setup times, and minimize total (weighted) tardiness.

### Minimize Makespan

If the objective is to minimize makespan, we can replace the deadline $d_j$ by $d'_j = \min\{d_j, z^*\}$ for all jobs $j$ and apply the same infeasibility condition as in Lemma 11.2.

### Minimize Sum of Setup Times

If $z^*$ represents an upper bound on the sum of setup times, we proceed as follows: For each arc $a = (u,v)$ in $\mathscr{M}$, let $st_a^{\downarrow}$ be the minimum possible sum of setup times incurred by the partial orderings represented by paths from $\mathbf{r}$ to $v$ that contain $a$. We recursively compute

$$st_a^{\downarrow} = \begin{cases} 0, & \text{if } a \in out(\mathbf{r}), \\ \min\{t_{d(a'),d(a)} + st_{a'}^{\downarrow} : a' \in in(u), d(a) \neq d(a')\}, & \text{otherwise.} \end{cases}$$

Now, for each arc $a = (u,v)$ let $st_a^{\uparrow}$ be the minimum possible sum of setup times incurred by the partial orderings represented by paths from $u$ to $\mathbf{t}$ that contain $a$. The state $st_a^{\uparrow}$ can be recursively computed through a bottom-up traversal of $\mathscr{M}$, as follows:

$$st_a^{\uparrow} = \begin{cases} 0, & \text{if } a \in in(\mathbf{t}), \\ \min\{t_{d(a),d(a')} + st_{a'}^{\uparrow} : a' \in out(v), d(a) \neq d(a')\}, & \text{otherwise.} \end{cases}$$

**Lemma 11.3.** *An arc a is infeasible if*

$$st_a^{\downarrow} + st_a^{\uparrow} > z^*. \tag{11.6}$$

*Proof.* It follows directly from the definitions of $st_a^{\downarrow}$ and $st_a^{\uparrow}$. □

**Minimize Total Tardiness**

To impose an upper bound $z^*$ on the total tardiness, assume $ect_a$ is computed for each arc $a$. We define the length of an arc $a$ as $l_a = \max\{0, ect_a - \delta_{d(a)}\}$. For a node $u$, let $sp_u^{\downarrow}$ and $sp_u^{\uparrow}$ be the shortest path from $\mathbf{r}$ to $u$ and from $\mathbf{t}$ to $u$, respectively, with respect to the lengths $l_a$. That is,

$$sp_u^{\downarrow} = \begin{cases} 0, & \text{if } u = \mathbf{r}, \\ \min\{l_a + sp_v^{\downarrow} : a = (v,u) \in in(u)\}, & \text{otherwise} \end{cases}$$

and

$$sp_u^{\uparrow} = \begin{cases} 0, & \text{if } u = \mathbf{t}, \\ \min\{l_a + sp_v^{\uparrow} : a = (u,v) \in out(u)\}, & \text{otherwise.} \end{cases}$$

**Lemma 11.4.** *A node u should be removed from $\mathscr{M}$ if*

$$sp_u^{\downarrow} + sp_u^{\uparrow} > z^*. \tag{11.7}$$

*Proof.* Length $l_a$ represents a lower bound on the tardiness of job $d(a)$ with respect to solutions identified by $\mathbf{r}$–$\mathbf{t}$ paths that contain $a$. Thus, $sp_u^{\downarrow}$ and $sp_u^{\uparrow}$ are a lower bound on the total tardiness for the partial orderings identified by paths from $\mathbf{r}$ to $u$ and $\mathbf{t}$ to $u$, respectively, since the tardiness of a job is nondecreasing in its completion time. □

## 11.6 Inferring Precedence Relations from Relaxed MDDs

Given a set of precedence relations for a problem (e.g., that were possibly derived from other relaxations), we can use the filtering rules (11.3) and (11.4) from Section 11.5.2 to strengthen a relaxed MDD. In this section, we show that a converse relation is also possible. Namely, given a relaxed MDD $\mathscr{M}$, we can deduce all precedence relations that are satisfied by the partial orderings represented by $\mathscr{M}$ in polynomial time in the size of $\mathscr{M}$. To this end, assume that the states $All_u^{\downarrow}$, $All_u^{\uparrow}$, $Some_u^{\downarrow}$, and $Some_u^{\uparrow}$ as described in Section 11.5.1 are computed for all nodes $u$ in $\mathscr{M}$. We have the following results:

**Theorem 11.1.** *Let $\mathscr{M}$ be an MDD that exactly identifies all the feasible orderings of $\mathscr{J}$. A job $j$ must precede job $j'$ in any feasible ordering if and only if either $j' \notin All_u^{\downarrow}$ or $j \notin All_u^{\uparrow}$ for all nodes $u$ in $\mathscr{M}$.*

*Proof.* Suppose there exists a node $u$ in layer $L_i$, $i \in \{1,\dots,n+1\}$, such that $j' \in All_u^{\downarrow}$ and $j \in All_u^{\uparrow}$. By definition, there exists a path $(\mathbf{r},\dots,u,\dots,\mathbf{t})$ that identifies an ordering where job $j'$ starts before job $j$. This is true if and only if job $j$ does not precede $j'$ in any feasible ordering. □

**Corollary 11.1.** *The set of all precedence relations that must hold in any feasible ordering can be extracted from $\mathscr{M}$ in $O(n^2|\mathscr{M}|)$.*

*Proof.* Construct a digraph $G^* = (\mathscr{J}, E^*)$ by adding an arc $(j,j')$ to $E^*$ if and only if there exists a node $u$ in $\mathscr{M}$ such that $j' \in All_u^{\downarrow}$ and $j \in All_u^{\uparrow}$. Checking this condition for all pairs of jobs takes $O(n^2)$ for each node in $\mathscr{M}$, and hence the time complexity to construct $G^*$ is $O(n^2|\mathscr{M}|)$. According to Theorem 11.1 and the definition of $G^*$, the complement graph of $G^*$ contains an edge $(j,j')$ if and only if $j \ll j'$. □

As we are mainly interested in relaxed MDDs, we derive an additional corollary of Theorem 11.1.

**Corollary 11.2.** *Given a relaxed MDD $\mathscr{M}$, an activity $j$ must precede activity $j'$ in any feasible solution if $(j' \notin Some_u^{\downarrow})$ or $(j \notin Some_u^{\uparrow})$ for all nodes $u$ in $\mathscr{M}$.*

*Proof.* It follows from the state definitions that $All_u^{\downarrow} \subseteq Some_u^{\downarrow}$ and $All_u^{\uparrow} \subseteq Some_u^{\uparrow}$. Hence, if the conditions for the relation $j \ll j'$ from Theorem 11.1 are satisfied by $Some_u^{\downarrow}$ and $Some_v^{\uparrow}$, they must be also satisfied by any MDD which only identifies feasible orderings. □

By Corollary 11.2, the precedence relations implied by the solutions of a relaxed MDD $\mathscr{M}$ can be extracted by applying the algorithm in Corollary 11.1 to the states $Some_v^{\downarrow}$ and $Some_v^{\uparrow}$. Since $\mathscr{M}$ has at most $O(nW)$ nodes and $O(nW^2)$ arcs, the time to extract the precedences has a worst-case complexity of $O(n^3W^2)$ by the presented algorithm. These precedences can then be used for guiding search or communicated to other methods or relaxations that may benefit from them.

## 11.7 Refinement

As in Section 4.7.1.2, we will develop a refinement procedure based on the permutation structure of the jobs, represented by the ALLDIFFERENT constraint. The goal of our heuristic refinement is to be as precise as possible with respect to the equivalence classes that refer to jobs with a higher *priority*, where the priority of a

job follows from the problem data. More specifically, we will develop a heuristic for refinement that, when combined with the infeasibility conditions for the permutation structure described in Section 11.5.1, yields a relaxed MDD where the jobs with a high priority are represented exactly with respect to that structure; that is, these jobs are assigned to exactly one position in all orderings encoded by the relaxed MDD. We also take care that a given maximum width $W$ is observed when creating new nodes in a layer.

Thus, if higher priority is given to jobs that play a greater role in the feasibility or optimality of the sequencing problem at hand, the relaxed MDD may represent more accurately the feasible orderings of the problem, providing, e.g., better bounds on the objective function value. For example, suppose we wish to minimize the makespan on an instance where certain jobs have very large release dates and processing times in comparison with other jobs. If we construct a relaxed MDD where these longer jobs are assigned exactly once in all orderings encoded by the MDD, the bound on the makespan would be potentially tighter with respect to the ones obtained from other possible relaxed MDDs for this same instance. Examples of job priorities for other objective functions are presented in Section 11.9. Recall from Section 4.7.1.2 that the refinement heuristic requires a *ranking* of jobs $\mathscr{J}^* = \{j_1^*, \ldots, j_n^*\}$, where jobs with smaller index in $\mathscr{J}^*$ have higher priority.

We note that the refinement heuristic also yields a *reduced* MDD $\mathscr{M}$ for certain structured problems, given a sufficiently large width. The following corollary, stated without proof, is directly derived from Lemma 4.4 and Theorem 4.3.

**Corollary 11.3.** *Assume $W = +\infty$. For a sequencing problem having only precedence constraints, the relaxed MDD $\mathscr{M}$ that results from the constructive proof of Theorem 4.3 is a reduced MDD that exactly represents the feasible orderings of this problem.*

Lastly, recall that equivalence classes corresponding to constraints other than the permutation structure may also be taken into account during refinement. Therefore, if the maximum width $W$ is not met in the refinement procedure above, we assume that we will further split nodes by arbitrarily partitioning their incoming arcs. Even though this may yield false equivalence classes, the resulting $\mathscr{M}$ is still a valid relaxation and may provide a stronger representation.

## 11.8 Encoding Size for Structured Precedence Relations

The actual constraints that define a problem instance greatly impact the size of an MDD. If these constraints carry a particular structure, we may be able to compactly represent that structure in an MDD, perhaps enabling us to bound its width.

In this section we present one such case for a problem class introduced by [14], in which jobs are subject to *discrepancy* precedence constraints: For a fixed parameter $k \in \{1,\dots,n\}$, the relation $j_p \ll j_q$ must be satisfied for any two jobs $j_p, j_q \in \mathscr{J}$ if $q \geq p+k$. This precedence structure was motivated by a real-world application in steel rolling mill scheduling. The work by [15] also demonstrates how solution methods to this class of problems can serve as auxiliary techniques in other cases, for example, as heuristics for the TSP and vehicle routing with time windows.

We stated in Corollary 11.3 that we are able to construct the reduced MDD $\mathscr{M}$ when only precedence constraints are imposed and a sufficiently large $W$ is given. We have the following results for $\mathscr{M}$ if the precedence relations satisfy the discrepancy structure for a given $k$:

**Lemma 11.5.** *We have* $All_v^{\downarrow} \subseteq \{j_1,\dots,j_{\min\{m+k-1,n\}}\}$ *for any given node* $v \in L_{m+1}$, $m = 1,\dots,n$.

*Proof.* If $m+k-1 > n$ we obtain the redundant condition $All_u^{\downarrow} \subseteq \mathscr{J}$, therefore assume $m+k-1 \leq n$. Suppose there exists $j_l \in All_v^{\downarrow}$ for some $v \in L_{m+1}$ such that $l > m+k-1$. Then, for any $i = 1,\dots,m$, we have $l-i \geq m+k-i \geq m+k-m = k$. This implies $\{j_1,\dots,j_m\} \subset All_v^{\downarrow}$, since job $j_l$ belongs to a partial ordering $\pi$ only if all jobs $j_i$ for which $l-i \geq k$ are already accounted for in $\pi$. But then $|All_v^{\downarrow}| \geq m+1$, which is a contradiction since $v \in L_{m+1}$ implies that $|All_v^{\downarrow}| = m$, as any partial ordering identified by a path from $\mathbf{r}$ to $v$ must contain $m$ distinct jobs. □

**Theorem 11.2.** *The width of* $\mathscr{M}$ *is* $2^{k-1}$.

*Proof.* Let us first assume $n \geq k+2$ and restrict our attention to layer $L_{m+1}$ for some $m \in \{k,\dots,n-k+1\}$. Also, let $\mathscr{F} := \{All_u^{\downarrow} : u \in L_{m+1}\}$. It can be shown that, if $\mathscr{M}$ is reduced, no two nodes $u, v \in L_{m+1}$ are such that $All_u^{\downarrow} = All_v^{\downarrow}$. Thus, $|\mathscr{F}| = |L_{m+1}|$.

We derive the cardinality of $\mathscr{F}$ as follows: Take $All_v^{\downarrow} \in \mathscr{F}$ for some $v \in L_{m+1}$. Since $|All_v^{\downarrow}| = m$, there exists at least one job $j_i \in All_v^{\downarrow}$ such that $i \geq m$. According to Lemma 11.5, the maximum index of a job in $All_v^{\downarrow}$ is $m+k-1$. So consider the jobs indexed by $m+k-1-l$ for $l = 0,\dots,k-1$; at least one of them is necessarily contained in $All_v^{\downarrow}$. Due to the discrepancy precedence constraints, $j_{m+k-1-l} \in All_v^{\downarrow}$ implies that any $j_i$ with $i \leq m-l-1$ is also contained in $All_v^{\downarrow}$ (if $m-l-1 > 0$).

Now, consider the sets in $\mathscr{F}$ which contain a job with index $m+k-1-l$, but do *not* contain any job with index greater than $m+k-1-l$. Any such set $All_u^{\downarrow}$ contains the jobs $j_1,\ldots,j_{m-l-1}$ according to Lemma 11.5. Hence, the remaining $m-(m-l-1)-1=l$ job indices can be freely chosen from the sequence $m-l,\ldots,m+k-l-2$. Notice there are no imposed precedences on these remaining $m+k-l-2-(m-l)+1=k-1$ elements; thus, there exist $\binom{k-1}{l}$ such subsets. But these sets define a partition of $\mathscr{F}$. Therefore

$$|\mathscr{F}| = |L_{m+1}| = \sum_{l=0}^{k-1} \binom{k-1}{l} = \binom{k-1}{0} + \cdots + \binom{k-1}{k-1} = 2^{k-1}.$$

We can use an analogous argument for the layers $L_{m+1}$ such that $m < k$ or $m > n-k+1$, or when $k = n-1$. The main technical difference is that we have fewer than $k-1$ possibilities for the new combinations, and so the maximum number of nodes is strictly less than $2^{k-1}$ for these cases. Thus the width of $\mathscr{M}$ is $2^{k-1}$. □

According to Theorem 11.2, $\mathscr{M}$ has $O(n\,2^{k-1})$ nodes as it contains $n+1$ layers. Since arcs only connect nodes in adjacent layers, the MDD contains $O(n\,2^{2k-2})$ arcs (assuming a worst-case scenario where all nodes in a layer are adjacent to all nodes in the next layer, yielding at most $2^{k-1}\cdot 2^{k-1} = 2^{2k-2}$ arcs directed out of a layer). Using the recursive relation (11.2) in Section 11.3, we can compute, e.g., the minimum sum of setup times in worst-case time complexity of $O(n^2\,2^{2k-2})$. The work by [14] provides an algorithm that minimizes this same function in $O(n\,k^2\,2^{k-2})$, but that is restricted to this particular objective.

## 11.9 Application to Constraint-Based Scheduling

We next describe how the techniques of the previous sections can be added to IBM ILOG CP Optimizer (CPO), a state-of-the-art general-purpose constraint programming solver. In particular, it contains dedicated syntax and associated propagation algorithms for sequencing and scheduling problems. Given a sequencing problem as considered in this chapter, CPO applies a depth-first branch-and-bound search where jobs are recursively appended to the end of a partial ordering until no jobs are left unsequenced. At each node of the branching tree, a number of sophisticated propagation algorithms are used to reduce the possible candidate jobs to be appended to

the ordering. Examples of such propagators include *edge-finding*, *not-first/not-last rules*, and *deductible precedences*; details can be found in [17] and [154].

We have implemented our techniques by introducing a new user-defined constraint type to the CPO system, representing a generic sequencing problem. We maintain a relaxed MDD for this constraint type, and we implemented the filtering and refinement techniques in the associated propagation algorithm. The constraint participates in the constraint propagation cycle of CPO; each time it is activated it runs one round of top-down filtering and refinement. In particular, the filtering operation takes into account the search decisions up to that point (i.e., the jobs that are already fixed in the partial ordering) and possible precedence constraints that are deduced by CPO. At the end of a round, we use the relaxed MDD to reduce the number of candidate successor jobs (by analyzing the arc labels in the appropriate layers) and to communicate new precedence constraints as described in Section 11.6, which may trigger additional propagation by CPO. Our implementation follows the guidelines from [101].

In this section we present computational results for different variations of single-machine sequencing problems using the MDD-based propagator. Our goal is twofold. First, we want to analyze the sensitivity of the relaxed MDD with respect to the width and refinement strategy. Second, we wish to provide experimental evidence that combining a relaxed MDD with existing techniques for sequencing problems can improve the performance of constraint-based solvers.

### *11.9.1 Experimental Setup*

Three formulations were considered for each problem: a CPO model with its default propagators, denoted by `CPO`; a CPO model containing only the MDD-based propagator, denoted by `MDD`; and a CPO model with the default and MDD-based propagators combined, denoted by `CPO+MDD`. The experiments mainly focus on the comparison between `CPO` and `CPO+MDD`, as these indicate whether incorporating the MDD-based propagator can enhance existing methods.

We have considered two heuristic strategies for selecting the next job to be appended to a partial schedule. The first, denoted by *lex search*, is a static method that always tries to first sequence the job with the smallest index, where the index of a job is fixed per instance and defined by the order in which it appears in the input. This allows for a more accurate comparison between two propagation methods,

since the branching tree is fixed. In the second strategy, denoted by *dynamic search*, the CPO engine automatically selects the next job according to its own state-of-the-art scheduling heuristics. The purpose of the experiments that use this search is to verify how the MDD-based propagator is influenced by strategies that are known to be effective for constraint-based solvers. The dynamic search is only applicable to `CPO` and `CPO+MDD`.

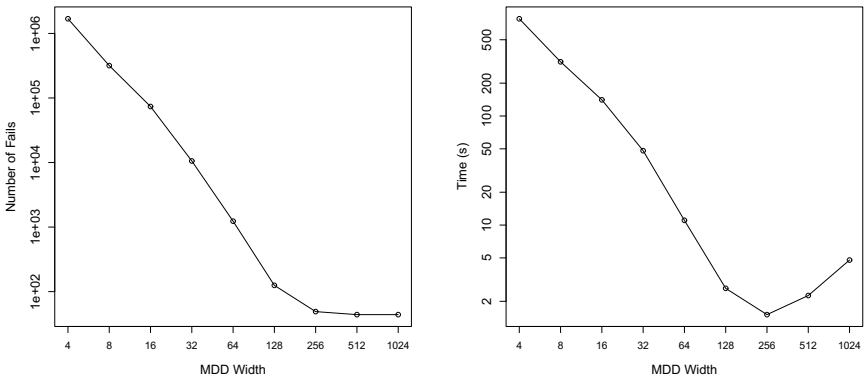
We measure two performance indicators: the total solving time and the *number of fails*. The number of fails corresponds to the number of times during search that a partial ordering was detected to be infeasible, i.e., either some constraint is violated or the objective function is greater than a known upper bound. The number of fails is proportional to the size of the branching tree and, hence, to the total solving time of a particular technique.

The techniques presented here do not explore any additional problem structure that was not described in this chapter, such as specific search heuristics, problem relaxations, or dominance criteria (except only if such structure is already explored by CPO). More specifically, we used the same MDD-based propagator for all problems, which dynamically determines what node state and refinement strategy to use according to the input constraints and the objective function.

The experiments were performed on a computer equipped with an Intel Xeon E5345 at 2.33 GHz with 8 GB RAM. The MDD code was implemented in C++ using the CPO callable library from ILOG CPLEX Academic Studio V.12.4.01. We set the following additional CPO parameters for all experiments: `Workers=1`, to use a single computer core; `DefaultInferenceLevel=Extended`, to use the maximum possible propagation available in CPO; and `SearchType=DepthFirst`.

### *11.9.2 Impact of the MDD Parameters*

We first investigate the impact of the maximum width and refinement on the number of fails and total solving time for the MDD approaches. As a representative test case, we consider the traveling salesman problem with time windows (TSPTW). The TSPTW is the problem of finding a minimum-cost tour in a weighted digraph starting from a selected vertex (the depot), visiting each vertex within a given time window, and returning to the original vertex. In our case, each vertex is a job, the release dates and deadlines are defined according to the vertex time windows, and



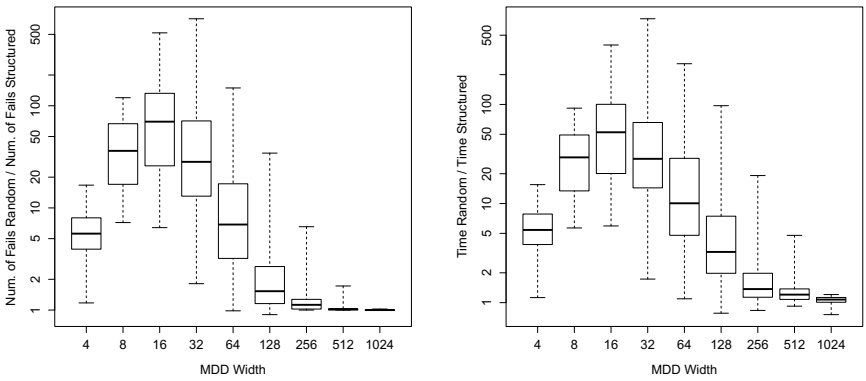

(a) Number of fails. (b) Time.

**Fig. 11.4** Impact of the MDD width on the number of fails and total time for the TSPTW instance `n20w200.001` from the Gendreau class. The axes are in logarithmic scale.

travel distances are perceived as setup times. The objective function is to minimize the sum of setup times.

We selected the instance `n20w200.001` from the well-known Gendreau benchmark proposed by [72], as it represents the typical behavior of an MDD. It consists of a 20-vertex graph with an average time window width of 200 units. The tested approach was the `MDD` model with lex search. We used the following job ranking for the refinement strategy described in Section 11.7: The first job in the ranking, $j_1^*$, was set as the first job of the input. The $i$-th job in the ranking, $j_i^*$, is the one that maximizes the sum of the setup times to the jobs already ranked, i.e., $j_i^* = \arg\max_{p \in \mathscr{J} \setminus \{j_1^*, \ldots, j_{i-1}^*\}} \{\sum_{k=1}^{i-1} t_{j_k^*, p}\}$ for the setup times $t$. The intuition is that we want jobs with largest travel distances to be exactly represented in $\mathscr{M}$.

The number of fails and total time to find the optimal solution for different MDD widths are presented in Fig. 11.4. Due to the properties of the refinement technique in Theorem 4.3, we consider only powers of 2 as widths. We note from Fig. 11.4(a) that the number of fails is decreasing rapidly as the width increases, up to a point where it becomes close to constant (from 512 to 1024). This indicates that, at a certain point, the relaxed MDD is very close to an actual exact representation of the problem, and hence no benefit is gained from any increment of the width. The number of fails has a direct impact on the total solving time, as observed in Fig. 11.4(b). Namely, the times decrease accordingly as the width increases. At




(a) Number of fails ratio. (b) Time ratio.

**Fig. 11.5** Performance comparison between random and structured refinement strategies for the TSPTW instance `n20w200.001`. The axes are in logarithm scale.

the point where the relaxed MDD is close to exact, larger widths only introduce additional overhead, thus increasing the solving time.

To analyze the impact of the refinement, we generated 50 job rankings uniformly at random for the refinement strategy described in Section 11.7. These rankings were compared with the *structured* one for setup times used in the previous experiment. To make this comparison, we solved the `MDD` model with lex search for each of the 51 refinement orderings, considering widths from 4 to 1024. For each random order, we divided the resulting number of fails and time by the ones obtained with the structured refinement for the same width. Thus, this ratio represents how much better the structured refinement is over the random strategies. The results are presented in the box-and-whisker plots of Fig. 11.5. For each width the horizontal lines represent, from top to bottom, the maximum observed ratio, the upper quartile, the median ratio, the lower quartile, and the minimum ratio.

We interpret Fig. 11.5 as follows: An MDD with very small width captures little of the jobs that play a more important role in the optimality or feasibility of the problem, in view of Theorem 4.3. Thus, distinct refinement strategies are not expected to differ much on average, as shown, e.g., in the width-4 case of Fig. 11.5(a). As the width increases, there is a higher chance that these crucial jobs are better represented by the MDD, leading to a good relaxation, but also a higher chance that little of their structure is captured by a random strategy, leading in turn to a weak relaxation. This yields a larger variance in the refinement performance.

Finally, for sufficiently large widths, we end up with an almost exact representation of the problem and the propagation is independent of the refinement order (e.g., widths 512 and 1024 of Fig. 11.5(a)). Another aspect we observe in Fig. 11.5(b) is that, even for relatively small widths, the structured refinement can be orders of magnitude better than a random one. This emphasizes the importance of applying an appropriate refinement strategy for the problem at hand.

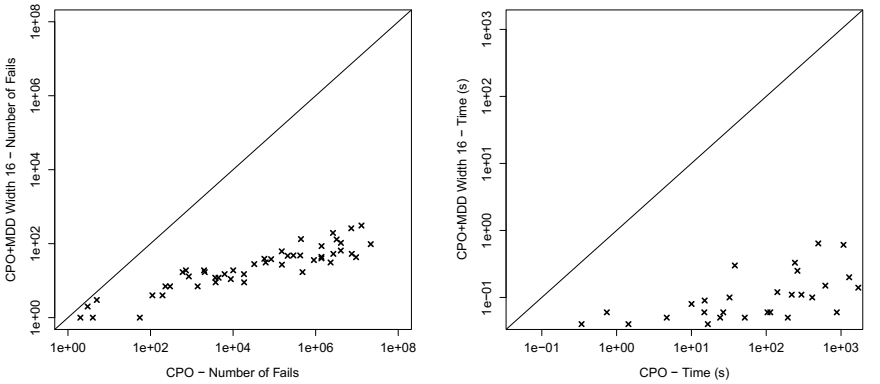
### *11.9.3 Traveling Salesman Problem with Time Windows*

We first evaluate the relative performance of `CPO` and `CPO+MDD` on sequencing problems with time window constraints, and where the objective is to minimize the sum of setup times. We considered a set of well-known TSPTW instances defined by the Gendreau, Dumas, and Ascheuer benchmark classes, which were proposed by [72], [56], and [10], respectively. We selected all instances with up to 100 jobs, yielding 388 test cases in total. The `CPO` and the `CPO+MDD` models were initially solved with lex search, considering a maximum width of 16. A time limit of 1,800 seconds was imposed for all methods, and we used the structured job ranking described in Section 11.9.2.

The `CPO` approach was able to solve 26 instances to optimality, while the `CPO+MDD` approach solved 105 instances to optimality. The number of fails and solution times are presented in the scatter plots of Fig. 11.6, where we only considered instances solved by both methods. The plots provide a strong indication that the MDD-based propagator can greatly enhance the CPO inference mechanism. For example, `CPO+MDD` can reduce the number of fails from over 10 million (`CPO`) to less than 100 for some instances.

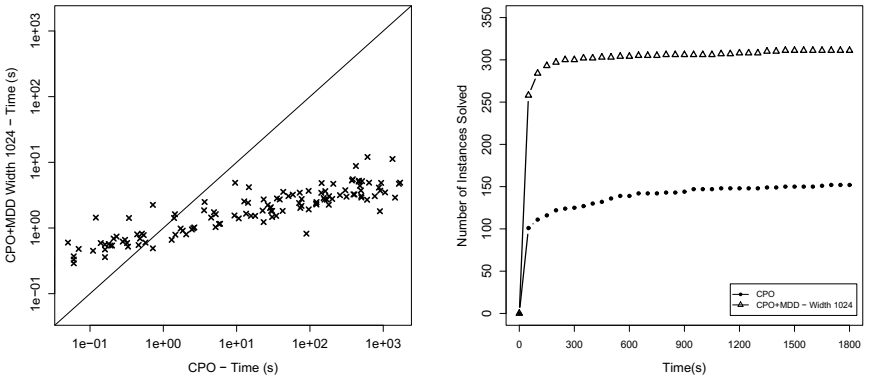
In our next experiment we compared `CPO` and `CPO+MDD` considering a maximum width of 1024 and applying instead a dynamic search, so as to verify if we could still obtain additional gains with the general-purpose scheduling heuristics provided by CPO. A time limit of 1,800 seconds was imposed for all approaches.

With the above configuration, the `CPO` approach solved to optimality 184 out of the 388 instances, while the `CPO+MDD` approach solved to optimality 311 instances. Figure 11.7(a) compares the times for instances solved by both methods, while Fig. 11.7(b) depicts the performance plot. In particular, the overhead introduced by the MDD is only considerable for small instances (up to 20 jobs). In the majority of the cases, `CPO+MDD` is capable of proving optimality much quicker.




(a) Number of fails. (b) Time.

**Fig. 11.6** Performance comparison between `CPO` and `CPO+MDD` for minimizing sum of setup times on Dumas, Gendreau, and Ascheuer TSPTW classes with lex search. The vertical and horizontal axes are in logarithmic scale.




(a) Scatter plot. (b) Performance plot.

**Fig. 11.7** Performance comparison between `CPO` and `CPO+MDD` for minimizing sum of setup times on Dumas, Gendreau, and Ascheuer TSPTW classes using default depth-first CPO search. The horizontal and vertical axes in (a) are in logarithmic scale.

### *11.9.4 Asymmetric Traveling Salesman Problem with Precedence Constraints*

We next evaluate the performance of `CPO` and `CPO+MDD` on sequencing problems with precedence constraints, while the objective is again to minimize the sum of

setup times. As benchmark problem, we consider the asymmetric traveling salesman problem with precedence constraints (ATSPP), also known as the *sequential ordering problem*. The ATSPP is a variation of the asymmetric TSP where precedence constraints must be observed. Namely, given a weighted digraph $D = (V,A)$ and a set of pairs $P = V \times V$, the ATSPP is the problem of finding a minimum-weight Hamiltonian tour $T$ such that vertex $v$ precedes $u$ in $T$ if $(v,u) \in P$.

The ATSPP has been shown to be extremely challenging for exact methods. In particular, a number of instances with fewer than 70 vertices from the well-known [147] benchmark, proposed initially by [11], are still open. We refer to the work of [6] for a more detailed literature review of exact and heuristic methods for the ATSPP.

We applied the `CPO` and `CPO+MDD` models with dynamic search and a maximum width of 2048 for 16 instances of the ATSPP from the TSPLIB benchmark. A time limit of 1,800 seconds was imposed, and we used the structured job ranking described in Section 11.9.2. The results are reported in Table 11.1. For each instance we report the size (number of vertices) and the current best lower and upper bound from the literature.[1] The column 'Best' corresponds to the best solution found by a method, and the column 'Time' corresponds to the computation time in which the solution was proved optimal. A value TL indicates that the time limit was reached.

We were able to close three of the unsolved instances with our generic approach, namely p43.2, p43.3, and ry48p.4. In addition, instance p43.4 was solved before with more than 22 hours of CPU time by [92] (for a computer approximately 10 times slower than ours), and by more than 4 hours by [76] (for an unspecified machine), while we could solve it in less than 90 seconds. The presence of more precedence constraints (indicated for these instances by a larger suffix number) is more advantageous to our MDD approach, as shown in Table 11.1. On the other hand, less constrained instances are better suited to approaches based on mixed integer linear programming; instances p43.1 and ry48p.1 are solved by a few seconds in [11].

As a final observation, we note that the bounds for the p43.1-4 instances reported in the TSPLIB are inconsistent. They do not match any of the bounds from existing works we are aware of or the ones provided by [11], where these problems were proposed. This includes the instance p43.1 which was solved in that work.

[1] Since the TSPLIB results are not updated on the TSPLIB website, we report updated bounds obtained from [92], [76], and [6].

**Table 11.1** Results on ATSPP instances. Values in bold represent instances solved for the first time. TL represents that the time limit (1,800 s) was reached.

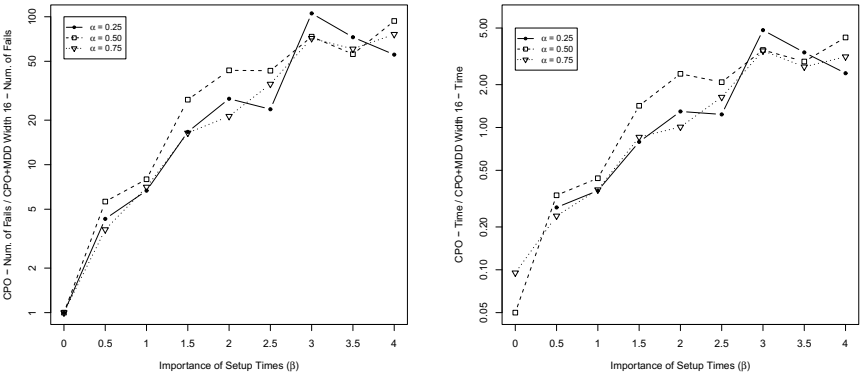
| Instance | Vertices | Bounds | CPO | | CPO+MDD width 2048 | |
|---|---|---|---|---|---|---|
| | | | Best | Time (s) | Best | Time (s) |
| br17.10 | 17 | 55 | 55 | 0.01 | 55 | 4.98 |
| br17.12 | 17 | 55 | 55 | 0.01 | 55 | 4.56 |
| ESC07 | 7 | 2125 | 2125 | 0.01 | 2125 | 0.07 |
| ESC25 | 25 | 1681 | 1681 | TL | 1681 | 48.42 |
| p43.1 | 43 | 28140 | 28205 | TL | 28140 | 287.57 |
| p43.2 | 43 | [28175, 28480] | 28545 | TL | **28480** | **279.18** |
| p43.3 | 43 | [28366, 28835] | 28930 | TL | **28835** | **177.29** |
| p43.4 | 43 | 83005 | 83615 | TL | 83005 | 88.45 |
| ry48p.1 | 48 | [15220, 15805] | 18209 | TL | 16561 | TL |
| ry48p.2 | 48 | [15524, 16666] | 18649 | TL | 17680 | TL |
| ry48p.3 | 48 | [18156, 19894] | 23268 | TL | 22311 | TL |
| ry48p.4 | 48 | [29967, 31446] | 34502 | TL | **31446** | **96.91** |
| ft53.1 | 53 | [7438, 7531] | 9716 | TL | 9216 | TL |
| ft53.2 | 53 | [7630, 8026] | 11669 | TL | 11484 | TL |
| ft53.3 | 53 | [9473, 10262] | 12343 | TL | 11937 | TL |
| ft53.4 | 53 | 14425 | 16018 | TL | 14425 | 120.79 |

### *11.9.5 Makespan Problems*

Constraint-based solvers are known to be particularly effective when the objective is to minimize makespan, which is largely due to specialized domain propagation techniques that can be used in such cases; see, e.g., [17].

In this section we evaluate the performance of `CPO` and `CPO+MDD` on sequencing problems with time window constraints and where the objective is to minimize makespan. Our goal is to test the performance of such procedures on makespan problems, and verify the influence of setup times on the relative performance. In particular, we will empirically show that the MDD-based propagator makes schedulers more robust for makespan problems, especially when setup times are present.

To compare the impact of setup times between methods, we performed the following experiment: Using the scheme from [41], we first generated three random instances with 15 jobs. The processing times $p_i$ are selected uniformly at random from the set $\{1, 100\}$, and release dates are selected uniformly at random from the set $\{0, \ldots, \alpha \sum_i p_i\}$ for $\alpha \in \{0.25, 0.5, 0.75\}$. No deadlines are considered. For each of the three instances above, we generated additional random instances where we add a setup time for all pairs of jobs $i$ and $j$ selected uniformly at random from the set



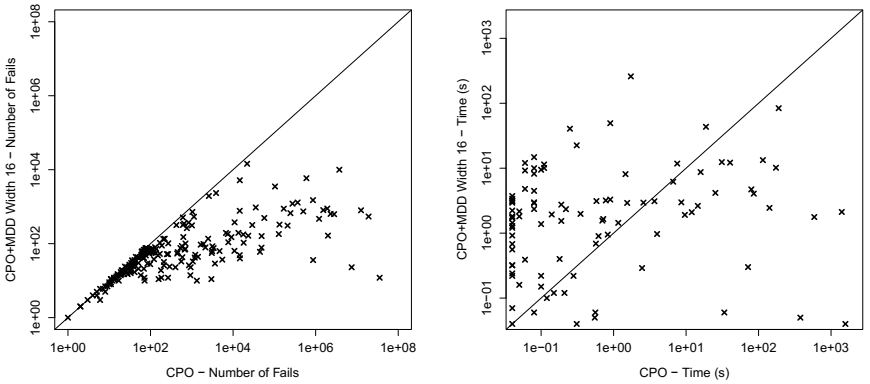

(a) Number of fails ratio. (b) Time ratio.

**Fig. 11.8** Comparison between CPO and CPO+MDD for minimizing makespan on three instances with randomly generated setup times. The vertical axes are in logarithmic scale.

$\{0,\ldots,(50.5)\beta\}$, where $\beta \in \{0, 0.5, 1, \ldots, 4\}$. In total, 10 instances are generated for each $\beta$. We computed the number of fails and total time to minimize the makespan using CPO and CPO+MDD models with a maximum width of 16, applying a lex search in both cases. We then divided the CPO results by the CPO+MDD results, and computed the average ratio for each value of $\beta$. The job ranking for refinement is done by sorting the jobs in decreasing order according to the value obtained by summing their release dates with their processing times. This forces jobs with larger completion times to have higher priority in the refinement.

The results are presented in Fig. 11.8. For each value of $\alpha$, we plot the ratio of CPO and CPO+MDD in terms of the number of fails (Fig. 11.8(a)) and time (Fig. 11.8(b)). The plot in Fig. 11.8(a) indicates that the CPO+MDD inference becomes more dominant in comparison with CPO for larger values of $\beta$, that is, when setup times become more important. The MDD introduces a computational overhead in comparison with the CPO times (around 20 times slower for this particular problem size). This is compensated as $\beta$ increases, since the number of fails for the CPO+MDD model becomes orders of magnitude smaller in comparison with CPO. The same behavior was observed on average for other base instances generated under the same scheme.

To evaluate this on structured instances, we consider the TSPTW instances defined by the Gendreau and Dumas benchmark classes, where we changed the objective function to minimize makespan instead of the sum of setup times. We




(a) Number of fails. (b) Time.

**Fig. 11.9** Performance comparison between `CPO` and `CPO+MDD` for minimizing makespan on Dumas and Gendreau TSPTW classes. The vertical and horizontal axes are in logarithmic scale.

selected all instances with up to 100 jobs, yielding 240 test cases in total. We solved the `CPO` and the `CPO+MDD` models with lex search, so as to compare the inference strength for these problems. A maximum width of 16 was set for `CPO+MDD`, and a time limit of 1,800 seconds was imposed for both cases. The job ranking is the same as in the previous experiment.

The `CPO` approach was able to solve 211 instances to optimality, while the `CPO+MDD` approach solved 227 instances to optimality (including all the instances solved by `CPO`). The number of fails and solving time are presented in Fig. 11.9, where we only depict instances solved by both methods. In general, for easy instances (up to 40 jobs or with a small time window width), the reduction of the number of fails induced by `CPO+MDD` was not significant, and thus did not compensate the computational overhead introduced by the MDD. However, we note that the MDD presented better performance for harder instances; the lower diagonal of Fig. 11.9(b) is mostly composed by instances from the Gendreau class with larger time windows, for which the number of fails was reduced by five and six orders of magnitude. We also note that the result for the makespan objective is less pronounced than for the sum of setup times presented in Section 11.9.3.

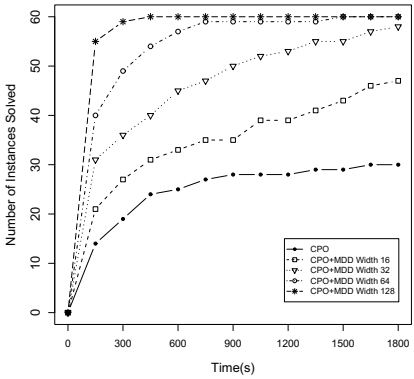
### 11.9.6 Total Tardiness

Constraint-based schedulers are usually equipped with specific filtering techniques for minimizing total tardiness, which are based on the propagation of a piecewise-linear function as described by [17]. For problems without any constraints, however, the existing schedulers are only capable of solving small instances, and heuristics end up being more appropriate as the propagators are not sufficiently strong to deduce good bounds.

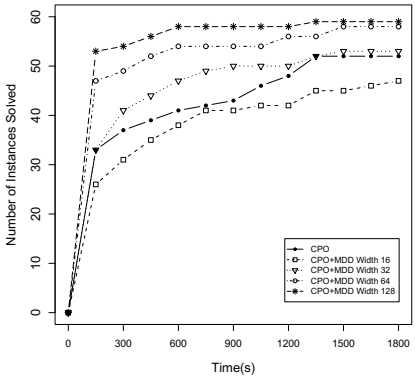
In this section we evaluate the performance of `CPO` and `CPO+MDD` on sequencing problems where the objective is to minimize the total tardiness. Since we are interested in evaluating the inference strength of the objective function bounding mechanism, we do not take into account any additional side constraints and we limit our problem size to 15 jobs. Moreover, jobs are only subject to a release date, and no setup time is considered.

We have tested the total tardiness objective using random instances, again generated with the scheme of [41]. The processing times $p_i$ are selected uniformly at random from the set $\{1,10\}$, the release dates $r_i$ are selected uniformly at random from the set $\{0,\ldots,\alpha\sum_i p_i\}$, and the due dates are selected uniformly at random from the set $\{r_i+p_i,\ldots,r_i+p_i+\beta\sum_i p_i\}$. To generate a good diversity of instances, we considered $\alpha \in \{0,0.5,1.0,1.5\}$ and $\beta \in \{0.05,0.25,0.5\}$. For each random instance generated, we create a new one with the same parameters but where we assign tardiness weights selected uniformly at random from the set $\{1,\ldots,10\}$. We generated 5 instances for each configuration, hence 120 instances in total. A time limit of 1,800 seconds was imposed for all methods. The ranking procedure for refinement is based on sorting the jobs in decreasing order of their due dates.

We compared the `CPO` and the `CPO+MDD` models for different maximum widths, and lex search was applied to solve the models. The results for unweighted total tardiness are presented in Fig. 11.10(a), and the results for the weighted total tardiness instances are presented in Fig. 11.10(b). We observe that, even for relatively small widths, the `CPO+MDD` approach was more robust than `CPO` for unweighted total tardiness; more instances were solved in less time even for a width of 16, which is a reflection of a great reduction of the number of fails. On the other hand, for weighted total tardiness `CPO+MDD` required larger maximum widths to provide a more significant benefit with respect to `CPO`. We believe that this behavior may be due to a weaker refinement for the weighted case, which may require larger widths to capture the set of activities that play a bigger role in the final solution cost.




(a) Total tardiness.




(b) Weighted total tardiness.

**Fig. 11.10** Performance comparison between `CPO` and `CPO+MDD` for minimizing total tardiness on randomly generated instances with 15 jobs.

In all cases, a minimum width of 128 would suffice for the MDD propagation to provide enough inference to solve all the considered problems.