Towards Bound Consistency for the No-Overlap Constraint Using MDDs

Anonymous author
Anonymous affiliation

Anonymous author
Anonymous affiliation

Anonymous author
Anonymous affiliation

Anonymous author
Anonymous affiliation

Abstract

Achieving Bound-Consistency (BC) for the No-Overlap constraint is known to be NP-complete. Therefore, several polynomial-time tightening techniques, such as edge finding, not-first-not-last reasoning, and energetic reasoning, have been introduced for this constraint. In this work, we derive the first BC algorithm for the No-Overlap constraint. By building on the No-Overlap MDD defined by Ciré and van Hoeve, we extract bounds of the time window of the jobs, allowing us to tighten start and end times in time polynomial in the number of nodes of the MDD. Similarly, to bound the size and complexity, we limit the width of the MDD to a threshold, creating a relaxed MDD that can also be used to relax the BC filtering. Through experiments on a sequencing problem with time windows and a just-in-time objective ($1 \mid r_j, d_j, \bar{d}_j \mid \sum E_j + \sum T_j$), we observe that the proposed filtering, even with a threshold on the width, achieves a stronger reduction in the number of nodes visited in the search tree compared to the previously proposed precedence-detection algorithm of Ciré and van Hoeve. The new filtering also appears to be complementary to classical propagation methods for the No-Overlap constraint, allowing a substantial reduction in both the number of nodes and the solving time on several instances.

2012 ACM Subject Classification Replace ccsdesc macro with valid one

Keywords and phrases MDD, No-Overlap, Disjunctive, Time windows

1 Introduction

Let $J = \{1, \ldots, n\}$, a set of n jobs. Each job $i \in J$ has a release date r_i (i.e., earliest time the task can start), a processing time $p_i > 0$, and a strict deadline $\bar{d}_i > r_i$ (i.e., latest time the task have to end). The disjunctive scheduling problem is the problem of determining start times so that all jobs execute within their time windows without overlapping[1]. This problem is NP-complete [9]. Disjunctive scheduling problems have already been widely studied in the Constraint Programming (CP) community [3, 14, 15, 16]. In CP, this problem is traditionally modeled using start-time variables $s_i \in [r_i, \bar{d}_i - p_i]$ and end-time variables $e_i = s_i + p_i$, or by using interval variables [17]. The non-overlapping requirement is enforced by a set of constraints that together define the semantics of the global No-Overlap (also known as Disjunctive) constraint ($\forall i, j \in J$, s.t. $i \neq j$, $e_i \leq s_j \ \vee \ e_j \leq s_i$).

Propagators prune inconsistent values by updating the earliest start $\underline{s_i}$ (i.e., lower bound of s_i) and latest completion times $\overline{e_i}$ (i.e., upper bound of e_i), tightening the time windows. Achieving bound consistency is at least as hard as solving the problem itself, which is NP-complete. Therefore, many polynomial-time algorithms (most of them running in $O(n^2)$ or $O(n \log(n))$) update the bounds using relaxed reasoning. The most common approaches are based on edge-finding, not-first/not-last, or detectable precedence reasoning (a non-exhaustive list of such algorithms includes [2, 4, 7, 8, 21]).

[1] Notations are borrowed from [13].

Domains	r_i	$\bar{d}_i$	p_i
t_1	4	6	2
t_2	0	10	3
t_3	0	9	2
t_4	7	19	6

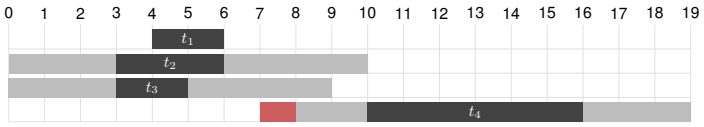


Figure 1 Job domains and timeline visualization with tightened start time for t_4.

▶ **Example 1.** A small No-Overlap instance is given in Fig. 1. Job t_4 cannot start at time 7, as t_2 and t_3 cannot both be scheduled before t_1, nor can they be after t_4. Ordering $\langle t_2, t_1, t_3, t_4 \rangle$ allows for the earliest feasible start time of 8 for t_4. None of the common filtering techniques can detect that t_4 cannot start at time 7. In contrast, a bound-consistent filtering algorithm would detect it.

To the best of our knowledge, a guaranteed bound-consistent (BC) filtering has not yet been developed. Despite a worst-case exponential time complexity, it could be valuable. First, it would allow researchers to check filtering properties, such as in [12] and measure the filtering gap of the existing polynomial-time algorithm to guide future research on filtering algorithms for the No-Overlap. It would also allow testing whether, for a given problem, increasing its filtering to BC significantly reduces the number of nodes (e.g., using the replay of [19]).

Ciré et al. [5] introduced a Multi-valued Decision Diagram (MDD) based on job sequencing, denoted NO-MDD hereafter. Each edge is labeled with a job. Consequently, a node at layer k corresponds to a state where a prefix of k jobs has been sequenced. Since the width of the NO-MDD can grow rapidly (proportional to the number of permutations), they propose to bound it to W via node-merging (i.e., creating a relaxed NO-MDD). They also infer job precedence relations from it to filter start times. Given a built NO-MDD, detecting all precedences takes $O(n^3W)$ time.

The purpose of this paper, building on the NO-MDD *[5], is threefold. It offers stronger propagation when working with exact* MDDs*. It proposes a practical polynomial filtering based on relaxed* MDDs*. And it delivers an empirical evaluation.* Specifically, it first describes a BC filtering that inspects all edges of the exact NO-MDD in $\mathcal{O}(|\mathcal{E}|)$ time (with $\mathcal{E}$, the set of edges) for a filtering stronger than [5]. It then presents a *polynomial-time* propagator using a relaxed NO-MDD running in $\mathcal{O}(n^2W)$, whose implementation takes advantage of the dynamic refinements found in Haddock [11]. Finally, it presents an empirical evaluation where the new propagator is a redundant constraint. The evaluation assesses the filtering strength relative to the state of the art and the precedence-based method [5].

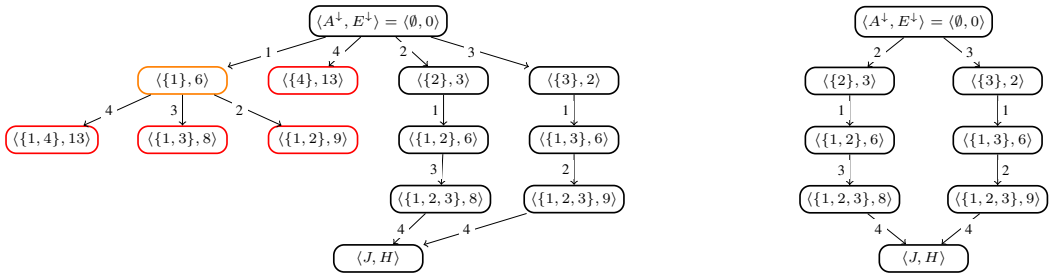
2 Bound-Consistent No-Overlap Filtering

We now recall the No-Overlap MDD from [5] and proceed to the first contribution: achieving BC when filtering start times with an exact MDD.

2.1 The No-Overlap Exact MDD

An MDD can be specified by a set of *states* $\mathcal{S}$, a set of possible *labels* $\mathcal{U}$, a *label* generating function $\lambda : \mathcal{S} \to \mathcal{U}$ (i.e., possible edges from a state), and a *state transition* function $\tau : \mathcal{S} \times \mathcal{U} \to \mathcal{S}$ computing the new state for a state and a label. For the NO-MDD, these are:

- A state is a tuple $\langle A^{\downarrow}, E^{\downarrow} \rangle$, where $A^{\downarrow} \subseteq J$ is the set of jobs already placed, and $E^{\downarrow}$ is the earliest time at which the next job can be placed. The initial state (*root*) is $\langle \emptyset, 0 \rangle$, i.e., no activities sequenced yet. The target state (*sink*) is defined as $\langle J, H \rangle$, i.e., the unique state where all the jobs have been sequenced, and H is an upper-bound on the latest completion time (horizon).
- Labels consist of the jobs to be scheduled (i.e., $\mathcal{U} = J$).
- $\lambda(\langle A^{\downarrow}, E^{\downarrow} \rangle) = J \setminus (A^{\downarrow} \cup \{i \in J \mid \max(E^{\downarrow}, \underline{s}_i) + p_i > \overline{e}_i\})$. It filters out jobs that have already been sequenced and those that cannot be scheduled while satisfying their deadlines.



(a) Exact NO-MDD with dead ends nodes yet to be pruned

(b) Exact NO-MDD

Figure 2 Construction process of an exact NO-MDD

- The state transition fuction τ is defined as: If $A^{\downarrow} \cup i = J$, $\tau(\langle A^{\downarrow}, E^{\downarrow}\rangle, i) = \langle J, H\rangle$, otherwise, $\tau(\langle A^{\downarrow}, E^{\downarrow}\rangle, i) = \langle A^{\downarrow} \cup i, \max(E^{\downarrow}, \underline{s}_i) + p_i\rangle$. This function ensures that job i is never scheduled earlier than the earliest feasible time of the originating state, $E^{\downarrow}$, nor than its own.

The NO-MDD is compiled top-down layer-wise, ensuring that each state is unique in each layer. One can note that applying λ to the *sink* node yields an empty set, ensuring, by construction, an MDD of at most $|J|$ layers of edges, leading to sequences of at most $|J|$ jobs. Since solutions are, by design, permutations of the J jobs, one knows that nodes on a path that does not reach the *sink* are not part of solutions. This is why a bottom-up pass is then applied to remove every node that does not belong to at least one path to the target. Let $\mathcal{E}$ denote the set of edges of the NO-MDD. Every directed $e = (u, v) \in \mathcal{E}$ is labeled by job ϕ_e. We further define $e.orig = u$ and $e.dest = v$.

▶ **Example 2.** The construction process of the exact NO-MDD is shown in Fig. 2. Figure 2a represents the top-down computations of each layer of the exact NO-MDD. Nodes display state $\langle A^{\downarrow}, E^{\downarrow}\rangle$. In red, one can see the childless nodes that will be removed at the bottom-up pass, and in orange, the ones removed consequently. The final exact NO-MDD is given at Fig. 2b.

2.2 Bound-Consistent Filtering using the NO-MDD

The earliest start time of job $i \in J$ is computed as the minimum of earliest start states among all edges labeled with i in the NO-MDD by rule 1. This is valid because every path from the root to the target node of the NO-MDD encodes a feasible sequencing of all jobs in J and guarantees that, by scheduling them sequentially in that order at their earliest start times, the deadline constraints are also satisfied. Therefore, the formula implicitly takes the earliest start time of job i among all valid sequences. Therefore, these filtering rules ensure bound-consistency of the No-Overlap constraint.

$$\underline{s}_i \leftarrow \max\left(\underline{s}_i, \min_{\{e \in \mathcal{E} \mid \phi_e = i\}} \left(E^{\downarrow}_{e.orig}\right)\right) \tag{1}$$

Note that the filtering can be performed for all jobs in $O(|\mathcal{E}|)$ time, since one can create an array initialized as $\underline{s}' = [+\infty \mid i \in J]$ and update the corresponding entry $\underline{s}'_{\phi_e} \leftarrow \min\left(\underline{s}'_{\phi_e}, E^{\downarrow}_{e.orig}\right)$ in constant time for each edge $e \in \mathcal{E}$. A similar filtering can be applied to tighten the latest end times. Given interval domains for the start- and end-times variables, the fix-point is obtained in one pass.

▶ **Example 3.** Considering the exact NO-MDD in Fig. 2b and job t_4, two edges have this job as a label. Therefore, the earliest start for this job is $\underline{s}_4 \leftarrow \min(8, 9)$.

3 Relaxed Bound-Consistent No-Overlap

The complexity for the BC filtering of the No-Overlap constraint is thus linear in the number of edges of the NO-MDD. However, this number can grow exponentially. To palliate that, [5] suggested

working on a relaxed NO-MDD, by bounding the width of each layer to W, encoding a super-set of all the valid sequences in the exact NO-MDD. This is achieved by merging certain states within layers. This section describes how to build a relaxed NO-MDD [5]. This section also explains how to refine the NO-MDD, following the Haddock framework [11].

3.1 The Relaxed No-Overlap MDD

Given an MDD representing a set of solutions S, its relaxation represents a super set of S. A relaxed MDD can be obtained by merging nodes with different states on a given layer of an MDD. This reduces its size while adding new invalid paths. To construct a top-down relaxed MDD, one needs to extend the definition of a state to enable node merging, as done in [5]. The specification of a relaxed NO-MDD including the node-merging operator $\oplus$ is given next.

- A state is a tuple $\langle A^{\downarrow}, S_u^{\downarrow}, E^{\downarrow}, K\rangle$, where $A^{\downarrow} \subseteq J$ represents the set of jobs in every path leading to this node, $S_u^{\downarrow} \subseteq A^{\downarrow}$ is the set of jobs in at least one path leading to this node, and $E^{\downarrow}$ is a lower-bound on the earliest start for the next job to be scheduled after all the already placed ones, and K is the number of scheduled jobs (ID of the node layer). The initial state is $\langle \emptyset, \emptyset, 0, 0\rangle$, a state where no activities have been sequenced yet, and 0 is assumed to be a lower-bound on the earliest start of all the jobs. The target state is defined as $\langle J, J, H, n\rangle$, the state where all the jobs have been sequenced (H is the horizon, i.e., the upper-bound on the latest completion time).
- $\mathcal{U} = J$.
- $\lambda'(\langle A^{\downarrow}, S^{\downarrow}, E^{\downarrow}, K\rangle) = \lambda(\langle A^{\downarrow}, E^{\downarrow}\rangle) \setminus \left(\mathbf{1}_{\{|S^{\downarrow}|=K\}} S^{\downarrow}\right)$ where $\mathbf{1}$ is the indicator function to express the conditional set subtraction. This function filters out jobs that have already been sequenced earlier and those that cannot be scheduled while satisfying their deadlines.
- The state transition function τ' is defined as: If $K = n-1$, $\tau'(\langle A^{\downarrow}, S^{\downarrow}, E^{\downarrow}, K\rangle, i) = \langle J, J, H, n\rangle$, otherwise $\tau'(\langle A^{\downarrow}, S^{\downarrow}, E^{\downarrow}, K\rangle, i) = \langle A^{\downarrow} \cup \{i\}, S^{\downarrow} \cup \{i\}, \max(E^{\downarrow}, \underline{s}_i) + p_i, K+1\rangle$. It ensures that job i is never scheduled earlier than the earliest feasible time of the originating state, $E^{\downarrow}$, nor earlier than its own earliest start time.
- $\oplus(\langle A_u^{\downarrow}, S_u^{\downarrow}, E_u^{\downarrow}, K\rangle, \langle A_v^{\downarrow}, S_v^{\downarrow}, E_v^{\downarrow}, K\rangle) = \langle \langle A_u^{\downarrow} \cap A_v^{\downarrow}, S_u^{\downarrow} \cup S_v^{\downarrow}, \min(E_u^{\downarrow}, E_v^{\downarrow}), K\rangle$

The top-down compilation of a relaxed MDD is similar to that of an exact one except, after expanding a layer, it needs to reduce its size to W by merging nodes ($\oplus$). We merge nodes by grouping them in buckets [20] based on their $E^{\downarrow}$ property. Buckets are obtained by partitioning the ranges of values of $E^{\downarrow}$ into W sub-intervals and merging nodes belonging to the same sub-interval.

▶ **Example 4.** Fig. 3 shows the top-down compilation of a relaxed NO-MDD. Property K (trivial) is omitted for clarity. Starting at the root, four nodes are created (λ' generates valid outgoing labels, τ' creates new states). This layer is reduced by bucketing (orange nodes merged). On the next layer, seven nodes are generated, bucketed, and merged. The process is repeated until each layer is constructed. Then, the MDD needs to be made sound again by removing the three dead-end nodes (circled in red) and propagating their removal upward (orange nodes removed). Circled green nodes composed the final MDD. One remark: even though the width is limited to 3 (higher than the exact width of 2 in Fig. 2b), the final MDD is not exact. This is due to the introduction of infeasible solutions during merging, leading to relaxed states that cannot remove some incorrect edges. Rule (1) does not tighten the earliest start of t_4 to 8 yet, but, since the full width is not used, we can refine this MDD.

3.2 Incremental update of the relaxed NO-MDD

During the search, along any branch of the search tree, the job time windows can only shrink. This might cause some transitions to become invalid according to λ'. One could recompute the NO-MDD from scratch, but it turns out to be less costly to update it incrementally using a *refinement* procedure.

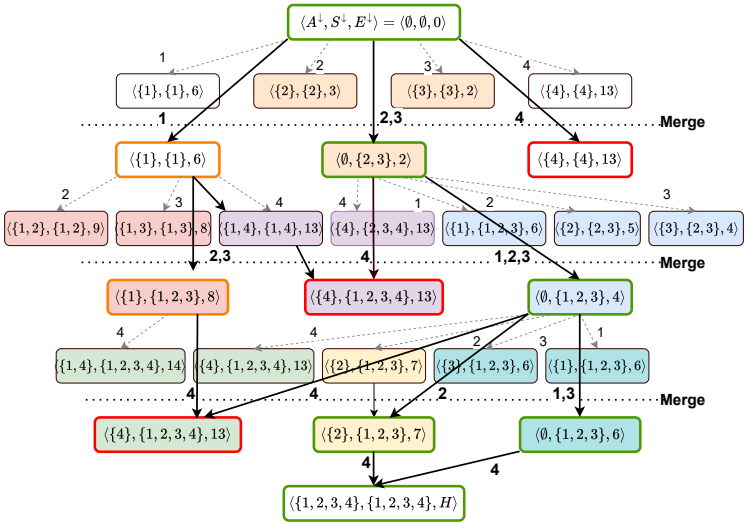


Figure 3 Relaxed NO-MDD generation bound to a width of 3

It first updates all properties in a top-down manner and deletes transitions when necessary. As a result, some nodes might become orphans or dead ends. After their removal, some layer might be left with fewer nodes than the allowed limit W. In such cases, nodes from previous layers are expanded again via the τ' function, and the layer is compressed when required using the merge operator $\oplus$. Such refinement procedures are well described in [5] and [11], which we adapt to our context here.

▶ **Example 5.** Fig. 4 shows the refinement. From top to bottom, a relaxed node is selected, one of its incoming edges is extracted, and a new node is created for it. On the example, the edge labeled 3 is extracted from the node $\langle \emptyset, \{2, 3\}, 2 \rangle$, and the MDD is updated. By rule (1), the value 7 is computed. Then, when extracting the edge labeled 2 and then the edge labeled 3 from the node $\langle \emptyset, \{1, 2, 3\}, 5 \rangle$, the resulting relaxed MDD is now strong enough to tighten the earliest start time of t_4 to 8.

So far, we have described the NO-MDD used to filter the earliest start times of the jobs, and noted that another MDD could be built in reverse order to prune the latest end times as well. In practice, as in [5, 11], we combine these two MDD by extending the state with the corresponding bottom-up values, i.e., $A^{\uparrow}$, $S^{\uparrow}$, $L^{\uparrow}$. The update of the states is thus performed both top-down and bottom-up, allowing for a stronger edge check based on the parts of the sequence before and after the edge.

3.3 BC filtering/precedences extraction on the relaxed NO-MDD

Applying our BC filtering or extracting the precedences on a relaxed NO-MDD follows the same formulas as in the exact ones. However, due to its relaxed nature, the same level of propagation is not guaranteed to be reached. This is why we speak about *Relaxed BC filtering* in that case.

To recall, precedences are extracted from an exact NO-MDD by a combination of top-down $A^{\downarrow}$ and bottom-up $A^{\uparrow}$ properties of a node (Theorem 6 in [5]). A job i precedes a job j (denoted $i \prec j$) if and only if $\forall$ nodes u, $(j \notin A_u^{\downarrow}) \vee (i \notin A_u^{\uparrow})$. Verifying it requires checking each job pair. On relaxed NO-MDD, the $S^{\downarrow}$ and $S^{\uparrow}$ are used to extract the precedence (Corollary 8 in [5]). A job i precedes a job j (denoted $i \prec j$) if and only if $\forall$ nodes u, $(j \notin S_u^{\downarrow}) \vee (i \notin S_u^{\uparrow})$. The time complexity is thus $O(n^2|\mathcal{V}|)$ (with $\mathcal{V}$ the set of nodes of the NO-MDD)[2] or $O(n^3 W)$, given W the width of the MDD.

[2] Can be expressed as $\Omega(n|\mathcal{E}|)$ to ease comparison with the $O(|\mathcal{E}|)$ for the BC filtering

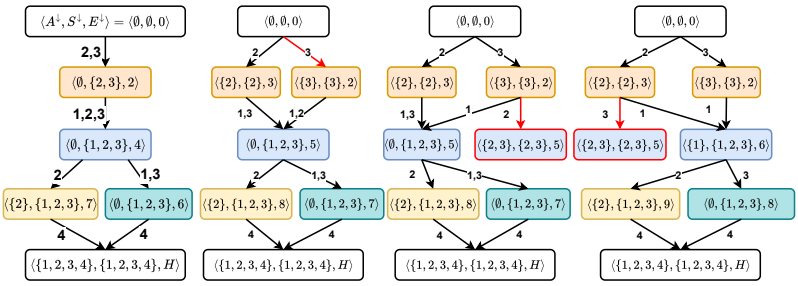


Figure 4 Edges extractions occurring during refinement of the relaxed MDD

▶ **Example 6.** In the running example, using the exact MDD, the precedences extracted are $1 \prec 4$, $2 \prec 4$, $3 \prec 4$. Precedence extraction also fails to tighten the time window of t_4. Therefore, even on an exact NO-MDD, the precedence extraction filtering does not reach the BC one for the No-Overlap.

4 Experiments

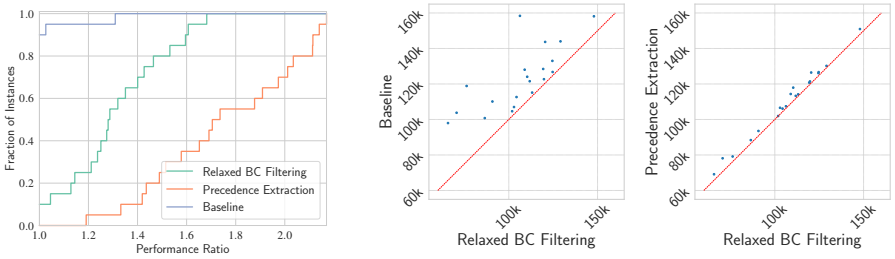
We study experimentally the problem of just-in-time scheduling $(1 \mid r_j, d_j, \bar{d}_j \mid \sum E_j + \sum T_j)$ [13, 1]. This problem involves scheduling n jobs on a single machine, given their release times r_j, processing times p_j, deadlines d_j, and a strict deadline $\bar{d}_j$ (i.e., the latest completion time). Given E_j the earliness of job j (i.e., $E_j = \max(0, d_j - e_j)$) and the tardiness T_j (i.e., $T_j = \max(0, e_j - d_j)$), the goal of the problem is to minimize $\sum E_j + \sum T_j$ while scheduling, in a non-overlapping manner, the jobs within their time window defined by their release time and strict deadline. We generated a given number of instances of various sizes (depending on the experiment). The processing time of each job was randomly selected from 1 to 25. To generate an instance, jobs were placed sequentially, and each job's time window was defined to overlap with the two preceding and two following jobs. The desired deadline was then randomly selected within this time window.

Our code is available online[3] and is implemented as a constraint in MaxiCP [18]. The experiments were run on an Intel Xeon Platinum 8160 @ 2.100GHz with 96 cores and 320 GB of RAM.

We compare four models: (i) model (*baseline*) using only all the filtering algorithms from [21] (i.e., overload-check, detectable precedences, not-first not-last, and edge finding) for the No-Overlap, (ii) model (*Relaxed BC Filtering*) with the NO-MDD, relaxed BC filtering and No-Overlap (redundant), (iii) model (*Precedence Extraction/PE*) with the NO-MDD, precedence extraction and No-Overlap, and (iv) model (*BC Filtering/BC*) with the NO-MDD, exact BC filtering and No-Overlap (redundant). The last model is not used in all experiments (especially with bigger instances) due to its complexity.

To avoid bias arising from discrepancies in programming language, MDD refinement strategies, and the solver used, we implemented precedence extraction within our solver. This allows both MDD-based filtering to compare in a similar environment. Moreover, we use a replay-based search framework [19] to ensure a fair comparison between the models. First, the weakest *baseline* model is used with the *conflict ordering search* [10] with a time limit of 1min. During this, the sequence of branching decisions is recorded, thereby defining a reference search tree (possibly partial if the run times out). This recorded tree is then replayed for the other models, which differ only in their pruning strength. All models are thus forced to explore exactly the same parts of the search tree, and any differences in performance can be attributed solely to the additional filtering they provide by pruning

[3] Released upon acceptance



(a) Time Performance profile **(b)** Plot of the search node counts

Figure 5 Results for instances of size $n = 40$ and relaxed MDDs bounded to $W = 16$

this forced search tree exploration. This avoids confounding effects due to heuristic interactions[4].

4.1 Search Space Reduction and Runtime Comparison

In this experiment, comparing the baseline, the model with relaxed BC filtering, and the model with precedence extraction, twenty instances of the problem were generated for each $n \in \{18, 25, 30, 40\}$. Fig. 5a displays a performance profile [6] (i.e., a cumulative over the instances of the ratio between the method's performance and the best performance overall) of the times of each method. The graph shows that our method outperforms the precedence extraction but is dominated by the baseline. Appendix contains the results for smaller instances. We also empirically confirmed the quadratic time complexity nature of Relaxed BC Filtering, in contrast to the cubic one of precedence extraction, by comparing the time taken at each node of the search tree. Complete results are in the Appendix.

MDD-based methods systematically reduce the number of nodes explored (Fig.5b). In addition, the relaxed BC filtering is even more reducing the search tree than extracting precedence, showing that even in a relaxed context, the effect of the stronger propagation is shown (Fig.5b).

We also varied the MDD size to assess the impact. The results in the Appendix are consistent with those of previous experiments in [5, 11]. The wider the MDD, the greater the reduction in the search tree, but there is an increasing exponential overhead of the operations related to construction and refinement of the MDD. A size for the MDD of 16 has also been shown (as in [5, 11]) to be a good compromise between relaxation quality and time spent.

4.2 Comparison to Bound-Consistency

Having a Bound-Consistent (BC) algorithm at our disposal, even if inefficient in practice for larger instances, allows us to empirically compare, on smaller instances, the pruning capacity of weaker algorithms relative to BC (i.e., how many nodes are unnecessarily explored compared to BC). For this experiment, 20 instances of sizes 12, 14, and 16 were generated. The replay method first records the search tree of the baseline model using a 1-minute timeout. The number of nodes explored, and the search time for the four models, as well as the width for the exact NO-MDD were monitored.

Figure 6 presents a cactus plot for each non-BC method of the search tree expansion between the method and the BC filtering. The search tree expansion g is computed using $g = \frac{Z-Z'}{Z'}$ where Z is the number of nodes explored during the search by the method under consideration, and Z' is the number of nodes explored when using the bound-consistent propagator. The tree expansion denotes

[4] It is well known that stronger propagation combined with first-fail–based heuristics may paradoxically lead to the exploration of larger search trees.

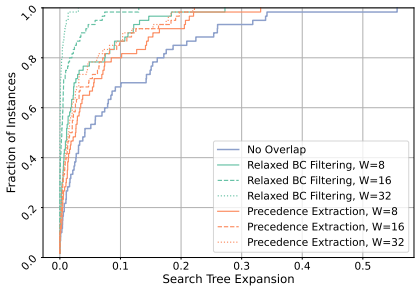


Figure 6 Cactus plot showing, for each method and MDD width, the proportion of replayed searches completed with a given gap g relative to the exact BC method. Smaller search tree expansion indicate behavior closer to BC.

Method	n	W	min (s)	max (s)	mean (s)
BC	12	-	4.6	449.0	82.7
Baseline	12	-	0.7	**18.0**	5.7
Relaxed BC	12	8	0.3	20.4	4.3
Relaxed BC	12	16	**0.2**	18.6	**4.1**
Relaxed BC	12	32	0.3	20.4	4.3
PE	12	8	**0.2**	21.1	4.4
PE	12	16	0.3	21.1	4.2
PE	12	32	0.3	21.2	4.4
BC	14	-	121.8	7141.5	2066.9
Baseline	14	-	4.9	**60.0**	**24.1**
Relaxed BC	14	8	2.6	82.1	27.7
Relaxed BC	14	16	2.5	98.3	29.2
Relaxed BC	14	32	2.2	70.6	26.0
PE	14	8	**2.1**	91.6	30.4
PE	14	16	**2.1**	117.0	32.1
PE	14	32	2.2	89.2	29.9
BC	16	-	1183.8	20630.4	8991.9
Baseline	16	-	**2.7**	**60.0**	**27.4**
Relaxed BC	16	8	3.2	**60.0**	28.1
Relaxed BC	16	16	3.7	64.1	28.5
Relaxed BC	16	32	3.7	61.8	27.7
PE	16	8	3.3	65.4	31.3
PE	16	16	4.0	75.6	32.6
PE	16	32	4.0	71.0	32.2

Table 1 Comparison of search time for different methods, problem size, and NO-MDD width.

the additional percentage of nodes required by the method compared to the BC search tree. As all the compared methods have a weaker consistency than BC, the tree expansions measured here are all positives. Closer to a zero tree expansion means closer to a bound consistent pruning.

The results, comparing the four models (baseline, relaxed BC filtering, precedence extraction, and BC filtering), show that the classical relaxation used in the baseline is relatively far from BC, requiring at least 10% more nodes on 30% of the tested instances, and with only 20% of the instances close to BC. The precedence extraction behaves a bit better, requiring at least 10% more nodes for 10 to 20% of the instance (depending on the W). The relaxed BC model shows the best results empirically, requiring at least 10% more nodes for 0 to 7% of the instance. With $W = 32$, it even shows a maximum of 5% tree expansion. For both models with NO-MDD, a larger width led to smaller tree expansion, demonstrating the expected improved quality of a less relaxed MDD.

Although the instances considered are small, the exact NO-MDD already attains an average width of 20 000 nodes during construction. This significantly limits scalability, as larger or real-world instances would quickly exceed practical memory constraints, resulting in prohibitive search times. Relax MDDs allows avoiding such an explosion of required memory by bounding the width.

Table 1 reports the minimum, maximum, and average replay times for each method. For the baseline, precedence extraction and relaxed BC, the values follows similar trends as for bigger instances (see Fig.5). For the model with the BC filtering, the results shows the drastic increase (several order of magnitude higher) of time resulting from the exponential behavior of the exact algorithm. This demonstrates that, for a fraction of the search time and a fraction of the NO-MDD width required, our method is able to reach performance close to bound-consistency.

To summarize the findings of this experiment, our relaxed BC algorithm exhibits the lowest search tree expansion among the relaxed algorithms, while maintaining control over memory usage.

5 Conclusion

This paper proposes the first BC algorithm to tighten the time window of the jobs in the No-Overlap constraint based on the No-Overlap MDD introduced in [5]. A relaxed polynomial-time version of it is obtained by relaxing the MDD. Experiments on a single-machine just-in-time scheduling problem showed that the new filtering method provides additional pruning compared to the standard filtering for the no-overlap constraint. The experiments also show that, compared to the classical filtering only, using the Relaxed BC filtering redundantly achieves filtering closer to bound-consistency.

References

1 Philippe Baptiste, Marta Flamini, and Francis Sourd. Lagrangian bounds for just-in-time job-shop scheduling. *Computers & Operations Research*, 2008.

2 Philippe Baptiste, Claude Le Pape, and Wim Nuijten. *Constraint-based scheduling: applying constraint programming to scheduling problems*. Springer Science & Business Media, 2001.

3 Nicolas Blais, Alexis Remartini, Claude-Guy Quimper, Nadia Lehoux, and Jonathan Gaudreault. Disjunctive scheduling with setup times: Optimizing a food factory. In *International Conference on Information Systems, Logistics and Supply Chain*, 2020.

4 Jacques Carlier and Eric Pinson. Adjustment of heads and tails for the job-shop problem. *European Journal of Operational Research*, 1994.

5 Andre Cire and Willem-jan Van Hoeve. Mdd propagation for disjunctive scheduling. *Proceedings of the International Conference on Automated Planning and Scheduling*, 2012. URL: http://dx.doi.org/10.1609/icaps.v22i1.13521, doi:10.1609/icaps.v22i1.13521.

6 Elizabeth D Dolan and Jorge J Moré. Benchmarking optimization software with performance profiles. *Mathematical programming*, 2002.

7 Hamed Fahimi, Yanick Ouellet, and Claude-Guy Quimper. Linear-time filtering algorithms for the disjunctive constraint and a quadratic filtering algorithm for the cumulative not-first not-last. *Constraints*, 2018.

8 Hamed Fahimi and Claude-Guy Quimper. Linear-time filtering algorithms for the disjunctive constraint. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2014.

9 D. S. Garey, M. R.; Johnson. Computers and intractability. *A guide to the theory of NP-Completeness*, 1979.

10 Steven Gay, Renaud Hartert, Christophe Lecoutre, and Pierre Schaus. *Conflict Ordering Search for Scheduling Problems*. 2015. doi:10.1007/978-3-319-23219-5_10.

11 Rebecca Gentzel, Laurent Michel, and W.-J. van Hoeve. *HADDOCK: A Language and Architecture for Decision Diagram Compilation*. 2020. doi:10.1007/978-3-030-58475-7_31.

12 Xavier Gillard, Pierre Schaus, and Yves Deville. Solvercheck: Declarative testing of constraints. In *International Conference on Principles and Practice of Constraint Programming*, pages 565–582. Springer, 2019.

13 Ronald Lewis Graham, Eugene Leighton Lawler, Jan Karel Lenstra, and AHG Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey. In *Annals of discrete mathematics*, volume 5, pages 287–326. Elsevier, 1979.

14 Adam Green, J Christopher Beck, and Amanda Coles. Using constraint programming for disjunctive scheduling in temporal ai planning. In *Proceedings of the Thirtieth Conference on Principles and Practice of Constraint Programming CP 2024*. Springer Berlin Heidelberg, 2024.

15 Diarmuid Grimes and Emmanuel Hebrard. Solving variants of the job shop scheduling problem through conflict-directed search. *INFORMS Journal on Computing*, 2015.

16 Emmanuel Hebrard. Disjunctive scheduling in tempo. In *31st International Conference on Principles and Practice of Constraint Programming (CP 2025)*, 2025.

17 Philippe Laborie and Jerome Rogerie. Reasoning with conditional time-intervals. In *FLAIRS*, 2008.

18 Pierre Schaus, Guillaume Derval, Augustin Delecluse, Laurent Michel, and Pascal Van Hentenryck. Maxicp: A constraint programming solver for scheduling and vehicle routing, 2024. URL: https://github.com/aia-uclouvain/maxicp.

19 Sascha Van Cauwelaert, Michele Lombardi, and Pierre Schaus. *Understanding the Potential of Propagators*. 2015. doi:10.1007/978-3-319-18008-3_29.

20 Hélène Verhaeghe, Roger Kameugne, Christophe Lecoutre, and Pierre Schaus. Improved filtering of scheduling problems using redundant table constraints. In *20th workshops on Constraint Modelling and Reformulation (ModRef2021)*, 2021.

21 Petr Vilím, Roman Barták, and Ondřej Čepek. Extension of o(n log n) filtering algorithms for the unary resource constraint to optional activities. *Constraints*, (4), 2005. doi:10.1007/s10601-005-2814-0.