# Vers la cohérence de borne pour la contrainte de non-chevauchement à l'aide des diagrammes de décision

**Amaury Guichard**[1*] **Laurent Michel**[2]
**Hélène Verhaeghe**[1] **Pierre Schaus**[1]
[1] UCLouvain, Louvain-La-Neuve, Belgique
[2] UCONN, Storrs, Connecticut, Etats-Unis d'Amérique
{amaury.guichard,helene.verhaeghe,pierre.schaus}@uclouvain.be
laurent.michel@uconn.edu

**Résumé**

Atteindre la cohérence de bornes pour la contrainte de non-chevauchement est NP-complet. Par conséquent, plusieurs techniques de filtrage plus faibles mais en temps polynomial, telles que la détection de précédences, le raisonnement pas-premier/pas-dernier et le raisonnement énergétique, ont été introduites pour cette contrainte.

Dans ce travail, nous définissons le premier algorithme assurant la cohérence de bornes pour la contrainte de non-chevauchement. En nous appuyant sur le diagramme de décision de la contrainte de non-chevauchement défini par Ciré et van Hoeve, des bornes sur les fenêtres temporelles des tâches sont extraites, permettant de resserrer les dates de début et de fin en un temps polynomial en fonction du nombre de nœuds du diagramme.

De manière similaire, afin de borner la taille et la complexité temporelle, la largeur du diagramme est limitée à une largeur seuil maximale, créant ainsi un diagramme relaxé. Celui-ci peut également être utilisé pour relaxer le filtrage assurant la cohérence de borne.

Une évaluation empirique sur un problème de séquencement avec fenêtres temporelles et un objectif juste à temps $(1 \mid r_j, d_j, \bar{d}_j \mid \sum E_j + \sum T_j)$ permet de montrer, même avec un seuil sur la largeur, une réduction du nombre de nœuds visités dans l'arbre de recherche par rapport à l'algorithme de détection de précédences proposé précédemment par Ciré et van Hoeve.

Ce nouveau filtrage apparaît également comme complémentaire aux méthodes classiques de propagation pour la contrainte de non-chevauchement, permettant une réduction substantielle à la fois du nombre de nœuds explorés et du temps de résolution sur plusieurs instances.

## 1 Introduction

Soit $J = \{1, \ldots, n\}$ un ensemble de $n$ tâches. Chaque tâche $i \in J$ possède une date de disponibilité $r_i$ (c.-à-d. le plus tôt où la tâche peut commencer), une durée d'exécution $p_i > 0$, et une échéance stricte $\bar{d}_i > r_i$ (c.-à-d. le plus tard où la tâche doit se terminer). Le problème d'ordonnancement disjonctif consiste à déterminer des temps de début tels que toutes les tâches s'exécutent dans leurs fenêtres temporelles sans chevauchement [1].Ce problème est NP-complet [9]. Ces problèmes ont déjà été largement étudiés dans la communauté de la programmation par contraintes (PPC) [3, 14, 15, 16, 18, 21, 22]. En PPC, il est traditionnellement modélisé à l'aide de variables de début $s_i \in [r_i, \bar{d}_i - p_i]$ et de variables de fin $e_i = s_i + p_i$, ou en utilisant des variables d'intervalle [17]. La contrainte de non-chevauchement est imposée par un ensemble de contraintes qui définissent conjointement la sémantique de la contrainte globale Non-Chevauchement (également appelée Disjonctive) :

$$\forall i, j \in J, \text{ t.q. } i \neq j,\ e_i \leq s_j \ \vee \ e_j \leq s_i$$

Les propagateurs éliminent les valeurs incohérentes en mettant à jour la date de début au plus tôt, $\underline{s_i}$ (c.-à-d. la borne inférieure de $s_i$), et la date de fin au plus tard, $\overline{e_i}$ (c.-à-d. la borne supérieure de $e_i$), resserrant ainsi les fenêtres

*Papier doctorant : Amaury Guichard[1] est auteur principal.

1. Les notations sont empruntées à [13].

temporelles. Atteindre la cohérence de bornes est au moins aussi difficile que de résoudre le problème lui-même, qui est NP-complet. Par conséquent, de nombreux algorithmes en temps polynomial ont été développés (la plupart en $O(n^2)$ ou en $O(n \log n)$), mettant à jour les bornes à l'aide de raisonnements relaxés. Les approches les plus courantes reposent sur l'edge-finding, le *pas-premier/pas-dernier* (i.e., *not-first/not-last*), ou le raisonnement par précédence détectable (liste non exhaustive : [2, 4, 7, 8, 25]).

**Exemple 1.** *Une petite instance de Non-Chevauchement est présentée à la Fig. 1. La tâche $t_4$ ne peut pas commencer au temps 7, car $t_2$ et $t_3$ ne peuvent pas toutes deux être planifiées avant $t_1$, ni après $t_4$. L'ordre $\langle t_2, t_1, t_3, t_4 \rangle$ permet de planifier le plus tôt possible $t_4$, soit 8. Aucune des techniques de filtrage usuelles ne peut détecter que $t_4$ ne peut pas commencer à l'instant 7. En revanche, un algorithme de filtrage cohérent en bornes le détecterait.*

À notre connaissance, aucun filtrage garantissant la cohérence de bornes (CB) n'a encore été développé pour la contrainte de Non-Chevauchement. Malgré une complexité exponentielle dans le pire des cas, un tel algorithme pourrait être utile. Premièrement, il permettrait aux chercheurs de vérifier les propriétés de filtrage, comme dans [12], et de mesurer l'écart de filtrage entre les algorithmes de filtrage polynomiaux existants afin de guider les travaux futurs sur ces algorithmes. Il permettrait également de tester si, pour un problème donné, renforcer le filtrage jusqu'à la CB réduit significativement le nombre de nœuds (par exemple via la technique de reparcours d'arbre de recherche de [23]).

Ciré et al. [5] ont introduit un diagramme de décision multivaleur (MDD) fondé sur le séquencement des tâches, noté NC-MDD dans la suite. Chaque arête est étiquetée par une tâche. Ainsi, un nœud à la couche $k$ correspond à un état où un préfixe de $k$ tâches a été planifié. Comme la largeur du NC-MDD peut croître rapidement (proportionnellement au nombre de permutations), ils proposent de la borner à $W$ via la fusion de nœuds (c.-à-d. en construisant un NC-MDD relaxé). Ce diagramme est alors utilisé pour détecter des relations de précédence entre les tâches afin de filtrer les dates de début. Étant donné un NC-MDD construit, détecter toutes les précédences nécessite un temps $O(n^3W)$, ou $O(n^2|\mathcal{V}|)$ (avec $\mathcal{V}$ l'ensemble des nœuds).

*L'objectif de cet article, s'appuyant sur le* NC-MDD *[5], est triple. Il offre une propagation plus forte avec des* MDD *exacts, propose un filtrage polynomial pratique basé sur des* MDD *relaxés et fournit une évaluation empirique de ces méthodes.*

Plus précisément, il décrit d'abord un filtrage CB qui examine toutes les arêtes du NC-MDD exact en $\mathcal{O}(|\mathcal{E}|)$ (où $\mathcal{E}$ désigne l'ensemble des arêtes), offrant un filtrage plus fort que celui de [5]. Il présente ensuite un propagateur en *temps polynomial* utilisant un NC-MDD relaxé en $\mathcal{O}(n^2W)$, dont l'implémentation exploite les raffinements dynamiques introduits dans Haddock [11]. Enfin, il propose une évaluation empirique du nouveau propagateur, utilisé comme contrainte redondante. Les expériences compare notre méthode en terme de puissance de filtrage à l'état de l'art et à la méthode fondée sur les précédences [5].

## 2 Filtrage Non-Chevauchement cohérent en bornes

Cette section a pour but de rappeler ce qu'est le MDD Non-Chevauchement de [5] et de présenter la première contribution : atteindre la cohérence de bornes lors du filtrage des dates de début à l'aide d'un MDD exact.

### 2.1 Le MDD exact pour Non-Chevauchement

Un MDD peut être défini par un ensemble d'*états* $\mathcal{S}$, un ensemble d'*étiquettes* possibles $\mathcal{U}$, une fonction de génération d'*étiquettes* $\lambda : \mathcal{S} \to \mathcal{U}$ (c.-à-d. les arêtes possibles depuis un état), et une fonction de *transition d'état* $\tau : \mathcal{S} \times \mathcal{U} \to \mathcal{S}$ qui calcule le nouvel état à partir d'un état et d'une étiquette. Pour le NC-MDD, on a :

- Un état est un couple $\langle A^{\downarrow}, E^{\downarrow} \rangle$, où $A^{\downarrow} \subseteq J$ est l'ensemble des tâches déjà placées, et $E^{\downarrow}$ est la date la plus tôt à laquelle la prochaine tâche peut être placée. L'état initial (*racine*) est $\langle \emptyset, 0 \rangle$, c.-à-d. l'état où aucune tâche n'a encore été ordonnancée. L'état final (*sortie*) est défini par $\langle J, H \rangle$, c.-à-d. l'unique état où toutes les tâches ont été ordonnancées, et $H$ est une borne supérieure de la date de fin (horizon).
- Les étiquettes correspondent aux tâches à ordonner (c.-à-d. $\mathcal{U} = J$).
- $\lambda(\langle A^{\downarrow}, E^{\downarrow} \rangle) = J \setminus \big(A^{\downarrow} \cup \{i \in J \mid \max(E^{\downarrow}, \underline{s}_i) + p_i > \overline{e}_i\}\big)$. Cette fonction élimine les tâches déjà ordonnancées ainsi que celles qui ne peuvent pas être planifiées tout en respectant leur échéance.
- La fonction de transition d'état $\tau$ est définie comme suit : si $A^{\downarrow} \cup i = J$, alors $\tau(\langle A^{\downarrow}, E^{\downarrow} \rangle, i) = \langle J, H \rangle$, sinon $\tau(\langle A^{\downarrow}, E^{\downarrow} \rangle, i) = \langle A^{\downarrow} \cup i,\ \max(E^{\downarrow}, \underline{s}_i) + p_i \rangle$. Cette fonction garantit que la tâche $i$ n'est jamais planifiée avant la date faisable la plus tôt de l'état d'origine $E^{\downarrow}$, ni avant sa propre date faisable.

Le NC-MDD est construit de haut en bas, couche par couche, en garantissant qu'un état est unique dans chaque couche. On peut remarquer que l'application de $\lambda$ au nœud *sortie* renvoie l'ensemble vide, ce qui garantit, par construction, un MDD comportant au plus $|J|$ couches d'arêtes, ce qui mène à des séquences d'au plus $|J|$ tâches. Comme les solutions sont, par construction, des permutations des tâches de $J$, les nœuds appartenant à un chemin qui n'atteint pas la *sortie* ne font pas partie des solutions. C'est pourquoi un parcours de bas en haut est ensuite appliqué afin de supprimer tout nœud n'appartenant à aucun chemin menant à l'état final.

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

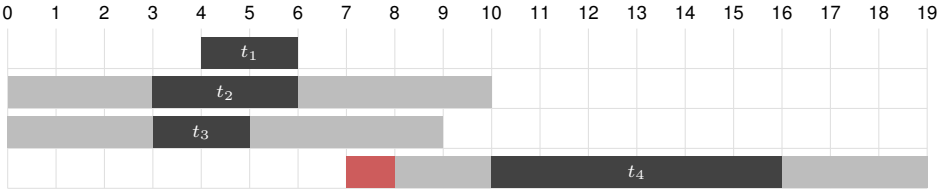
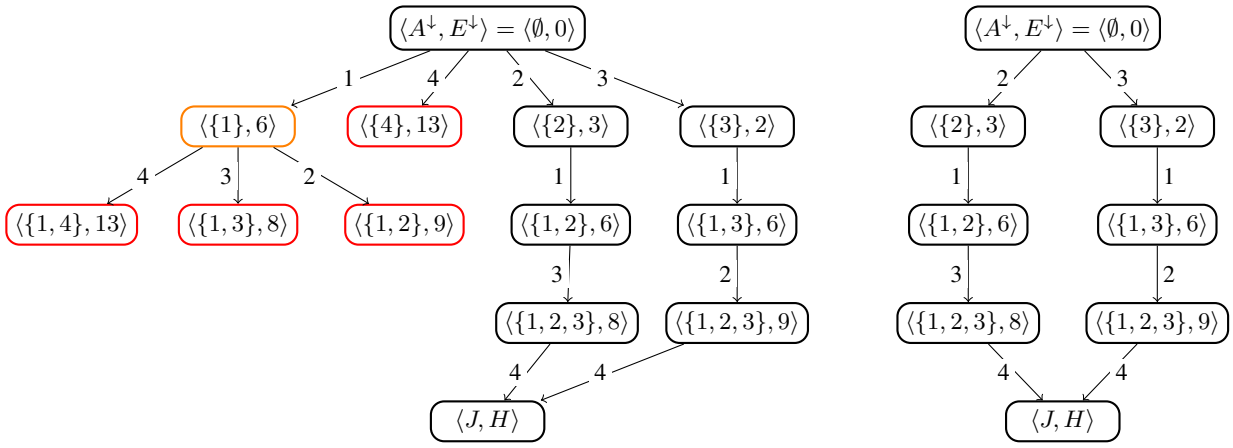



FIGURE 1 – Domaines des tâches et visualisation temporelle avec date de début resserrée pour $t_4$.




(a) NC-MDD exact avec des nœuds sans issue encore à élaguer

(b) NC-MDD exact

FIGURE 2 – Processus de construction d'un NC-MDD exact

Soit $\mathcal{E}$ l'ensemble des arêtes du NC-MDD. Chaque arête orientée $e = (u, v) \in \mathcal{E}$ est étiquetée par une tâche $\phi_e$. On définit également $e.orig = u$ et $e.dest = v$.

**Exemple 2.** *Le processus de construction du* NC-MDD *exact est illustré à la Fig. 2. La Fig. 2a montre le calcul de chaque couche du* NC-MDD *exact de haut en bas. Les nœuds affichent l'état* $\langle A^{\downarrow}, E^{\downarrow} \rangle$. *En rouge apparaissent les nœuds sans successeur qui seront supprimés lors du parcours de bas en haut, et en orange ceux supprimés par propagation. Le* NC-MDD *exact final est repris à la Fig. 2b.*

### 2.2 Filtrage cohérent en bornes à l'aide du NC-MDD

La date de début au plus tôt de la tâche $i \in J$ est calculée comme le minimum des dates de début au plus tôt de tous les états d'origine des arêtes étiquetées par $i$ dans le NC-MDD. Plus formellement :

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

Chaque chemin de la racine au nœud final du NC-MDD encode un ordonnancement faisable de toutes les tâches de $J$. Cela garantit que, si on les exécute séquentiellement dans cet ordre à leurs dates de début au plus tôt, les contraintes d'échéance sont satisfaites. La formule prend donc implicitement la date de début la plus tôt possible de la tâche $i$ parmi toutes les séquences valides. Ainsi, ces règles de filtrage assurent la cohérence de bornes de la contrainte Non-Chevauchement.

Notons que le filtrage peut être effectué pour toutes les tâches en $O(|\mathcal{E}|)$, puisqu'il suffit de créer un tableau initialisé par $\underline{s}' = [+\infty \mid i \in J]$ et de mettre à jour l'entrée correspondante $\underline{s}'_{\phi_e} \leftarrow \min\left(\underline{s}'_{\phi_e}, E^{\downarrow}_{e.orig}\right)$ en temps constant pour chaque arête $e \in \mathcal{E}$. Un filtrage similaire peut être appliqué pour resserrer les dates de fin au plus tard. Avec des intervalles de domaine pour les variables de début et de fin, le point fixe est atteint en un seul passage.

**Exemple 3.** *En considérant le* NC-MDD *exact de la Fig. 2b et la tâche* $t_4$, *deux arêtes portent cette tâche en étiquette. La date de début la plus tôt devient donc* $\underline{s}_4 \leftarrow \min(8, 9)$.

## 3 Cohérence de bornes relâchée pour Non-Chevauchement

La complexité du filtrage cohérent en bornes de la contrainte Non-Chevauchement est donc linéaire en le nombre d'arêtes du NC-MDD. Cependant, ce nombre peut croître exponentiellement. Pour y remédier, [5] propose de travailler sur un NC-MDD relâché, en bornant la largeur de

chaque couche à $W$, ce qui correspond à un sur-ensemble de toutes les séquences valides du NC-MDD exact. Cela est obtenu en fusionnant certains états au sein des couches. Cette section décrit comment construire un NC-MDD relâché [5]. Elle explique également comment raffiner le NC-MDD selon la méthode proposée par Haddock [11].

### 3.1 Le MDD relâché pour Non-Chevauchement

Étant donné un MDD représentant un ensemble de solutions $S$, sa relaxation représente un sur-ensemble de $S$. Un MDD relâché peut être obtenu en fusionnant des nœuds aux états différents au sein d'une même couche d'un MDD. Cela réduit sa taille tout en ajoutant de nouveaux chemins invalides. Pour construire un MDD relâché de haut en bas, il est nécessaire d'étendre la définition d'un état afin de permettre la fusion de nœuds, comme proposé dans [5]. La spécification d'un NC-MDD relâché, incluant l'opérateur de fusion $\oplus$, est présentée ci-dessous.

- Un état est un quadruplet $\langle A^{\downarrow}, S^{\downarrow}, E^{\downarrow}, K\rangle$, où $A^{\downarrow} \subseteq J$ représente l'ensemble des tâches présentes dans tous les chemins menant à ce nœud, $A^{\downarrow} \subseteq S^{\downarrow}$ est l'ensemble des tâches présentes dans au moins un chemin menant à ce nœud, $E^{\downarrow}$ est une borne inférieure sur la date de début la plus tôt de la prochaine tâche à planifier après celles déjà placées, et $K$ est le nombre de tâches ordonnancées (indice de la couche). L'état initial est $\langle \emptyset, \emptyset, 0, 0\rangle$, c.-à-d. un état où aucune tâche n'a encore été ordonnancée, et $0$ est supposé être une borne inférieure sur la date de début au plus tôt de toutes les tâches. L'état final est $\langle J, J, H, n\rangle$, c.-à-d. l'état où toutes les tâches ont été ordonnancées ($H$ est l'horizon, borne supérieure de la date de fin).
- $\mathcal{U} = J$.
- $\lambda'(\langle A^{\downarrow}, S^{\downarrow}, E^{\downarrow}, K\rangle) = \lambda(\langle A^{\downarrow}, E^{\downarrow}\rangle) \setminus \left(\mathbf{1}_{\{|S^{\downarrow}|=K\}} S^{\downarrow}\right)$ où $\mathbf{1}$ est la fonction indicatrice. Cette fonction élimine les tâches déjà ordonnancées ainsi que celles qui ne peuvent être planifiées tout en respectant leur échéance.
- La fonction de transition $\tau'$ est définie comme suit : si $K = n-1$, alors $\tau'(\langle A^{\downarrow}, S^{\downarrow}, E^{\downarrow}, K\rangle, i) = \langle J, J, H, n\rangle$, sinon $\tau'(\langle A^{\downarrow}, S^{\downarrow}, E^{\downarrow}, K\rangle, i) = \langle A^{\downarrow} \cup \{i\}, S^{\downarrow} \cup \{i\}, \max(E^{\downarrow}, \underline{s_i}) + p_i, K+1\rangle$. Cette fonction garantit que la tâche $i$ n'est jamais planifiée avant la date la plus tôt possible de l'état d'origine $E^{\downarrow}$, ni avant sa propre date de début la plus tôt possible.
- L'opérateur de fusion est défini par
- $\oplus(\langle A_u^{\downarrow}, S_u^{\downarrow}, E_u^{\downarrow}, K\rangle, \langle A_v^{\downarrow}, S_v^{\downarrow}, E_v^{\downarrow}, K\rangle) = \langle\langle A_u^{\downarrow} \cap A_v^{\downarrow}, S_u^{\downarrow} \cup S_v^{\downarrow}, \min(E_u^{\downarrow}, E_v^{\downarrow}), K\rangle$

La construction de haut en bas d'un MDD relâché est similaire à celle d'un MDD exact, sauf qu'après l'expansion d'une couche, sa taille doit être réduite à $W$ par fusion de nœuds ($\oplus$). Les nœuds sont regroupés en groupes [24] en fonction de leur propriété $E^{\downarrow}$. Ces groupes sont obtenus en partitionnant l'intervalle des valeurs possibles de $E^{\downarrow}$ en $W$ sous-intervalles, puis en fusionnant les nœuds appartenant au même sous-intervalle.

**Exemple 4.** *La Fig. 3 illustre la construction, de haut en bas, d'un* NC-MDD *relâché.*

*La propriété $K$ (triviale) est omise pour plus de lisibilité. Depuis la racine, quatre nœuds sont créés ($\lambda'$ génère les étiquettes valides, $\tau'$ crée les nouveaux états). Cette couche est ensuite réduite par regroupement (fusion des nœuds orange). À la couche suivante, sept nœuds sont générés, regroupés puis fusionnés. Le processus est répété jusqu'à la construction complète.*

*Ensuite, le* MDD *est rendu cohérent en supprimant les trois nœuds sans issue (encerclés en rouge) et en propageant leur suppression vers le haut (les nœuds encerclés en orange sont alors supprimés). Les nœuds encerclés en vert forment le* MDD *final.*

*Remarque : bien que la largeur soit limitée à 3 (supérieure à la largeur 2 du* MDD *exact de la Fig. 2b), le* MDD *final n'est pas exact. Cela provient de l'introduction de solutions infaisables lors des fusions, ce qui conduit à des états relâchés incapables de supprimer certaines arêtes incorrectes. La règle (1) ne resserre pas encore la date de début au plus tôt de $t_4$ à 8. Comme la largeur maximale n'est pas atteinte, ce* MDD *peut toutefois être raffiné.*

### 3.2 Mise à jour incrémentale du NC-MDD relâché

Au cours de la recherche, le long d'une branche de l'arbre, les fenêtres temporelles des tâches ne peuvent que se resserrer. Certaines transitions sont ainsi invalidées, ce qui réduit la taille du NC-MDD lors du filtrage de ces chemins. Une solution, pour utiliser à nouveau toute la largeur autorisée, serait de reconstruire le NC-MDD à chaque étape de la recherche. Cependant, il est moins coûteux de le mettre à jour de manière incrémentale à l'aide d'une procédure de *raffinement*.

Cette procédure met d'abord à jour toutes les propriétés de haut en bas et supprime les transitions devenues invalides. Certains nœuds peuvent alors devenir orphelins ou sans issue. Après leur suppression, certaines couches peuvent contenir moins de nœuds que la limite $W$. Dans ce cas, des nœuds des couches précédentes sont à nouveau étendus via la fonction $\tau'$, puis la couche est compressée si nécessaire à l'aide de l'opérateur $\oplus$.

De telles procédures de raffinement sont décrites en détail dans [5] et [11], que nous adaptons ici à notre contexte.

**Exemple 5.** *Le processus de raffinement est illustré à la Fig. 4. De haut en bas, un nœud relâché est sélectionné, une de ses arêtes entrantes est extraite, puis un nouveau nœud est créé pour la recevoir.*

*Dans l'exemple, l'arête étiquetée* 3 *est d'abord extraite du nœud $\langle \emptyset, \{2,3\}, 2\rangle$, puis le* MDD *est mis à jour. Selon*

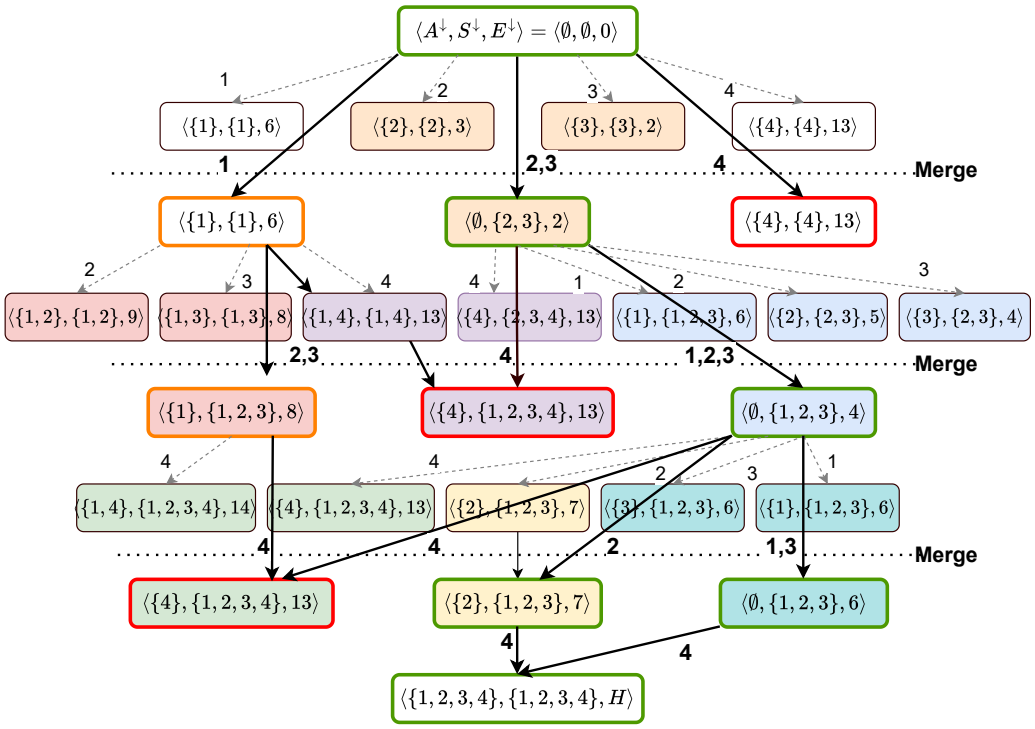



FIGURE 3 – Construction d'un NC-MDD relâché borné à une largeur de 3

*la règle de filtrage* (1)*, la valeur* 7 *est obtenue. Ensuite, en extrayant l'arête étiquetée* 2 *puis l'arête étiquetée* 3 *du nœud* $\langle \emptyset, \{1, 2, 3\}, 5 \rangle$*, le* MDD *relâché devient suffisamment précis pour resserrer la date de début au plus tôt de* $t_4$ *à* 8.

Jusqu'ici, nous avons décrit uniquement le NC-MDD utilisé pour filtrer les dates de début au plus tôt des tâches, et noté qu'un autre MDD pourrait être construit dans l'ordre inverse afin d'élaguer également les dates de fin au plus tard. En pratique, comme dans [5, 11], ces deux MDD sont combinés en étendant l'état avec les valeurs correspondantes calculées de bas en haut, c.-à-d. $A^{\uparrow}$, $S^{\uparrow}$, $L^{\uparrow}$. La mise à jour des états est ainsi effectuée à la fois de haut en bas et de bas en haut, ce qui permet une vérification de l'existence des arêtes plus forte, en tenant compte des parties de la séquence situées avant et après chaque arête.

### 3.3 Filtrage cohérent en bornes / extraction de précédences sur le NC-MDD relâché

L'application du filtrage cohérent en bornes ou l'extraction des précédences sur un NC-MDD relâché suit les mêmes formules que pour un NC-MDD exact. Cependant, en raison du caractère relâché de ce MDD, le même niveau de propagation n'est pas garanti. C'est pourquoi on parle ici de *filtrage cohérent en bornes relâché*.

Rappelons que les précédences sont extraites d'un NC-MDD exact par combinaison des propriétés descendantes $A^{\downarrow}$ et montantes $A^{\uparrow}$ d'un nœud (Théorème 6 dans [5]). Une tâche $i$ précède une tâche $j$ (notée $i \prec j$) si et seulement si

$$\forall \text{ nœuds } u,\ (j \notin A_u^{\downarrow}) \ \vee \ (i \notin A_u^{\uparrow}).$$

La vérification nécessite l'examen de chaque paire de tâches.

Dans un NC-MDD relâché, les ensembles $S^{\downarrow}$ et $S^{\uparrow}$ sont utilisés pour extraire les précédences (Corollaire 8 dans [5]). Une tâche $i$ précède une tâche $j$ si et seulement si

$$\forall \text{ nœuds } u,\ (j \notin S_u^{\downarrow}) \ \vee \ (i \notin S_u^{\uparrow}).$$

La complexité temporelle est donc $O(n^2|\mathcal{V}|)$ (où $\mathcal{V}$ est l'ensemble des nœuds du NC-MDD)[2], soit $O(n^3 W)$ si $W$ est la largeur du MDD.

**Exemple 6.** *Dans l'exemple fil rouge, en utilisant le* MDD *exact, les précédences extraites sont* $1 \prec 4$, $2 \prec 4$, $3 \prec 4$. *L'extraction des précédences ne parvient pas non plus à resserrer la fenêtre temporelle de* $t_4$. *Ainsi, même sur un* NC-MDD *exact, le filtrage par précédences n'atteint pas la cohérence de bornes de la contrainte Non-Chevauchement.*

## 4 Expériences

Nous étudions expérimentalement le problème d'ordonnancement juste-à-temps $(1 \mid r_j, d_j, \bar{d}_j \mid \sum E_j + \sum T_j)$

2. Peut également s'exprimer en $\Omega(n|\mathcal{E}|)$ afin de faciliter la comparaison avec le $O(|\mathcal{E}|)$ du filtrage cohérent en bornes.

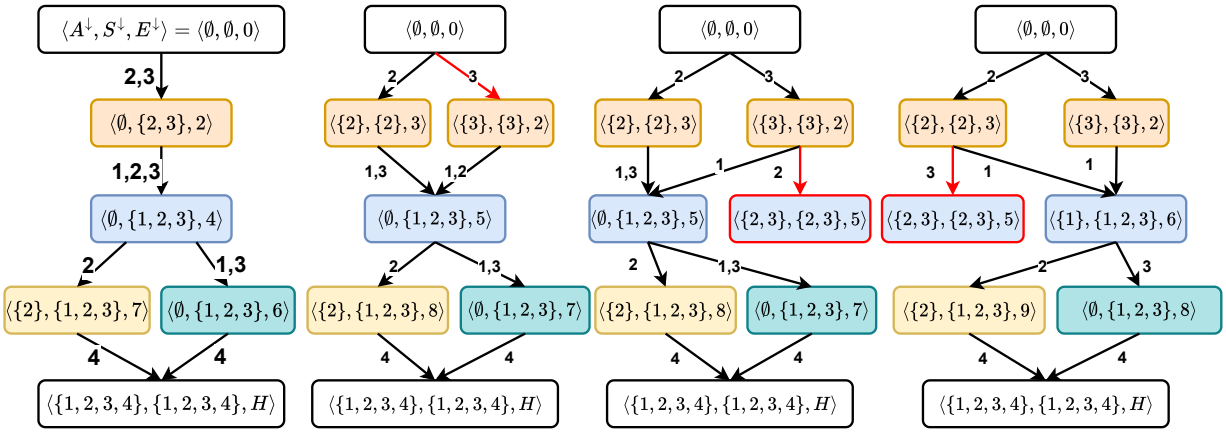



FIGURE 4 – Extractions d'arêtes lors du raffinement du MDD relâché

[13, 1]. Ce problème consiste à ordonnancer $n$ tâches sur une seule machine, étant données leurs dates de disponibilité $r_j$, leurs durées de traitement $p_j$, leurs échéances $d_j$, et une échéance stricte $\bar{d}_j$ (c.-à-d., le temps d'achèvement maximal). Étant donné $E_j$ l'avance de la tâche $j$ (c.-à-d., $E_j = \max(0, d_j - e_j)$) et le retard $T_j$ (c.-à-d., $T_j = \max(0, e_j - d_j)$), l'objectif du problème est de minimiser la somme totale des avances et des retards des tâches (c.-à-d., $\sum E_j + \sum T_j$) tout en planifiant, sans chevauchement, les tâches dans leur fenêtre temporelle définie par leur date de disponibilité et leur échéance stricte. Nous avons généré un certain nombre d'instances de tailles variées (selon l'expérience). La durée de traitement de chaque tâche a été tirée aléatoirement entre 1 et 25. Pour générer une instance, les tâches ont été placées séquentiellement, et la fenêtre temporelle de chaque tâche a été définie de manière à chevaucher les deux tâches précédentes et les deux suivantes. L'échéance souhaitée a ensuite été choisie aléatoirement dans cette fenêtre temporelle.

Notre code source est disponible en ligne[3] et est implémenté comme une contrainte dans MaxiCP [20], une version étendue du solveur MiniCP [19]. Les expériences ont été exécutées sur un Intel Xeon Platinum 8160 @ 2.100GHz avec 96 cœurs et 320 Go de RAM.

Nous comparons quatre modèles : (i) le modèle (*base*) utilisant uniquement tous les algorithmes de filtrage de [25][4] pour le Non-Chevauchement, (ii) le modèle (*Relaxed CB Filtering*) avec le NC-MDD, filtrage CB relâché et Non-Chevauchement (redondant), (iii) le modèle (*Precedence Extraction/PE*) avec le NC-MDD, extraction de précédence et Non-Chevauchement, et (iv) le modèle (*CB Filtering/CB*) avec le NC-MDD, filtrage CB exact et Non-Chevauchement (redondant). Le dernier modèle n'est pas utilisé dans toutes les expériences (en particulier avec des instances plus grandes) en raison de sa complexité.

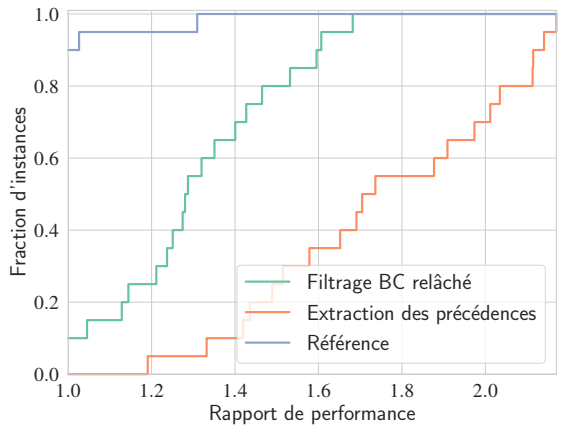
Afin d'éviter tout biais dû aux différences de langage de programmation, de stratégies de raffinement MDD ou de solveur utilisé, nous avons implémenté l'extraction de précédence dans notre solveur. Cela permet aux deux filtrages basés sur MDD de se comparer dans un environnement similaire. De plus, nous utilisons une méthode de recherche basée sur le reparcours d'arbre de recherche [23] afin d'assurer une comparaison équitable entre les modèles. Tout d'abord, le modèle *base*, le plus faible, est utilisé avec la *conflict ordering search* [10]) avec une limite de temps d'une minute. Durant cette exécution, la séquence des décisions de branchement est enregistrée, définissant ainsi un arbre de recherche de référence (éventuellement partiel si le temps est écoulé). Cet arbre de recherche enregistré est ensuite reparcouru pour les autres modèles, qui ne diffèrent que par leur puissance d'élagage. Ainsi, tous les modèles sont forcés d'explorer exactement les mêmes parties de l'arbre de recherche, et toute différence de performance peut être attribuée uniquement au filtrage supplémentaire qu'ils fournissent en élaguant cette exploration forcée de l'arbre. En particulier, cela évite les effets de confusion dus aux interactions heuristiques (Il est bien connu qu'une propagation plus forte, combinée à des heuristiques de type échec-en-premier, peut paradoxalement conduire à l'exploration d'arbres de recherche plus grands.).

### 4.1 Réduction de l'espace de recherche et comparaison des temps d'exécution

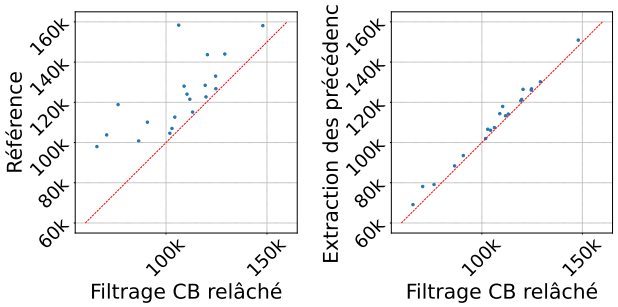
Dans cette expérience, en comparant le modèle de base, le modèle avec filtrage CB relâché, et le modèle avec extraction de précédence, vingt instances du problème ont été générées pour chaque $n \in \{18, 25, 30, 40\}$. La Fig. 5a présente un profil de performance [6] (c.-à-d.,

3. Code anonymisé sur Github https://github.com/RevenMyst/Disjunctive-Bound-Consistency-with-MDD

4. C'est-à-dire vérification de surcharge (i.e., *overload-check*), raisonnement par précédence détectable, pas-premier/pas-dernier, et edge-finding.




(a) Profil de performance en temps




(b) Graphique du nombre de nœuds de recherche

FIGURE 5 – Résultats pour des instances de taille $n = 40$ et des MDDs relâchés bornés à $W = 16$

un cumul, sur les instances, du ratio entre la performance de la méthode et la meilleure performance globale) des temps de chaque méthode. Le graphique montre que notre méthode surpasse l'extraction de précédence mais est dominée par le modèle de base. L'Annexe (https://github.com/RevenMyst/Disjunctive-Bound-Consistency-with-MDD) contient les résultats pour des instances plus petites. Nous avons également confirmé empiriquement la nature quadratique de la complexité temporelle du filtrage CB relâché, contrairement à la complexité cubique de l'extraction de précédence, en comparant le temps pris à chaque nœud de l'arbre de recherche. Les résultats complets sont en Annexe.

Les méthodes basées sur MDD réduisent systématiquement le nombre de nœuds explorés (Fig.5b). De plus, le filtrage CB relâché réduit encore davantage l'arbre de recherche que l'extraction de précédence, montrant que même dans un contexte relâché, l'effet d'une propagation plus forte est visible (Fig.5b).

Nous avons également fait varier la taille du MDD afin d'en évaluer l'impact. Les résultats en Annexe sont cohérents avec ceux des expériences précédentes dans [5, 11]. Plus le MDD est large, plus la réduction de l'arbre de recherche est importante, mais il existe un surcoût exponentiel croissant des opérations liées à la construction et au raffinement du MDD. Une taille de MDD de 16 s'est également révélée (comme dans [5, 11]) être un bon compromis entre la qualité de la relaxation et le temps passé.

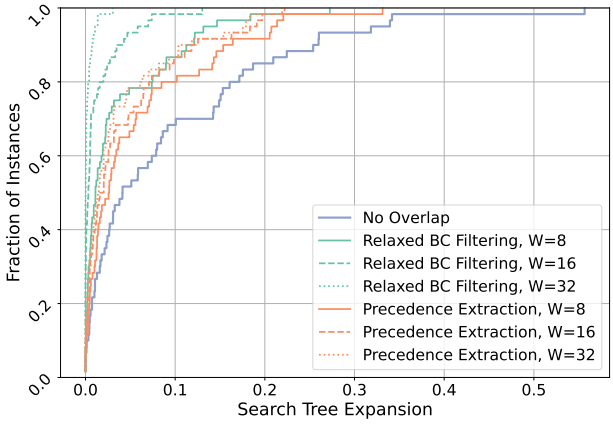



FIGURE 6 – Graphique cactus montrant, pour chaque méthode et largeur de MDD, la proportion de recherches rejouées complétées avec un écart $g$ donné par rapport à la méthode avec filtrage en cohérence de bornes. Une expansion plus faible de l'arbre de recherche indique un comportement plus proche de la cohérence de bornes.

## 4.2 Comparaison à la Cohérence de Bornes

Disposer d'un algorithme de cohérence de bornes, même inefficace en pratique pour des instances plus grandes, permet de comparer empiriquement, sur de petites instances, la capacité d'élagage d'algorithmes plus faibles par rapport à la cohérence de bornes (c.-à-d., combien de nœuds sont inutilement explorés par rapport à CB). Pour cette expérience, 20 instances de tailles 12, 14 et 16 ont été générées. La méthode de reparcours enregistre d'abord l'arbre de recherche du modèle de base avec un timeout d'une minute. Le nombre de nœuds explorés et le temps de recherche pour les quatre modèles, ainsi que la largeur pour le NC-MDD exact, ont été mesurés.

La Figure 6 présente un graphique cactus pour chaque méthode non-CB de l'expansion de l'arbre de recherche entre la méthode et le filtrage CB. L'expansion de l'arbre de recherche $g$ est calculée avec la formule suivante :

$$g = \frac{Z - Z'}{Z'}$$

où $Z$ est le nombre de nœuds explorés durant la recherche par la méthode considérée, et $Z'$ est le nombre de nœuds explorés lors de l'utilisation du propagateur avec filtrage en cohérence de bornes. L'expansion de l'arbre indique le pourcentage supplémentaire de nœuds requis par la méthode par rapport à l'arbre de recherche CB. Comme toutes les

| Méthode | $n$ | $W$ | min (s) | max (s) | moyenne (s) |
|---|---|---|---|---|---|
| CB (exact NC-MDD) | 12 | - | 4.6 | 449.0 | 82.7 |
| Base | 12 | - | 0.7 | **18.0** | 5.7 |
| CB Relaché | 12 | 8 | 0.3 | 20.4 | 4.3 |
| CB Relaché | 12 | 16 | **0.2** | 18.6 | **4.1** |
| CB Relaché | 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 |
| CB (exact NC-MDD) | 14 | - | 121.8 | 7141.5 | 2066.9 |
| Base | 14 | - | 4.9 | **60.0** | **24.1** |
| CB Relaché | 14 | 8 | 2.6 | 82.1 | 27.7 |
| CB Relaché | 14 | 16 | 2.5 | 98.3 | 29.2 |
| CB Relaché | 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 |
| CB (exact NC-MDD) | 16 | - | 1183.8 | 20630.4 | 8991.9 |
| Base | 16 | - | **2.7** | **60.0** | **27.4** |
| CB Relaché | 16 | 8 | 3.2 | **60.0** | 28.1 |
| CB Relaché | 16 | 16 | 3.7 | 64.1 | 28.5 |
| CB Relaché | 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 – Comparaison du temps de recherche en secondes pour différentes méthodes, tailles de problème et largeurs de NC-MDD.

méthodes comparées ont une consistance plus faible que CB, les expansions mesurées ici sont toutes positives. Une expansion proche de zéro signifie un élagage proche de la cohérence de borne

Les résultats, comparant les quatre modèles (de base, filtrage CB relâché, extraction de précédence, et filtrage CB), montrent d'abord que la relaxation classique utilisée dans le modèle de base est relativement éloignée de CB, nécessitant au moins 10% de nœuds en plus sur 30% des instances testées, et avec seulement 20% des instances proches ($<$1% d'expansion) de CB. L'extraction de précédence se comporte légèrement mieux, nécessitant au moins 10% de nœuds en plus pour 10 à 20% des instances (selon $W$). Le modèle CB relâché montre empiriquement les meilleurs résultats, nécessitant au moins 10% de nœuds en plus pour 0 à 7% des instances. Avec une largeur de 32, il montre même une expansion maximale de 5% sur l'ensemble du benchmark. Pour les deux modèles avec NC-MDD, une largeur plus grande conduit à une expansion plus faible de l'arbre, démontrant l'amélioration attendue de la qualité d'un MDD moins relâché.

Bien que les instances considérées soient petites, il a été observé que le NC-MDD exact atteint déjà une largeur moyenne de 20 000 nœuds lors de sa construction. Cela limite fortement le passage à l'échelle, car des instances plus grandes ou réelles dépasseraient rapidement les contraintes pratiques de mémoire, entraînant des temps de recherche prohibitifs. Les MDDs relâchés permettent d'éviter une telle explosion mémoire en bornant la largeur.

Le Tableau 1 rapporte les temps minimum, maximum et moyen de rejeu pour chaque méthode. Pour le modèle de base, l'extraction de précédence et le CB relâché, les valeurs suivent des tendances similaires à celles des instances plus grandes (voir Fig.5). Pour le modèle avec filtrage CB, les résultats montrent l'augmentation drastique (plusieurs ordres de grandeur) du temps résultant du comportement exponentiel de l'algorithme exact. Cela démontre que, pour une fraction du temps de recherche et une fraction de la largeur de NC-MDD requise, notre méthode est capable d'atteindre des performances proches de la cohérence de bornes.

Pour résumer les résultats de cette expérience, notre algorithme CB relâché présente la plus faible expansion de l'arbre de recherche parmi les algorithmes relâchés (écart inférieur à $5\%$ pour $W = 32$ sur chaque instance), tout en maintenant un contrôle de l'utilisation de la mémoire.

## 5 Conclusion

Cet article propose le premier algorithme CB pour resserrer la fenêtre temporelle des tâches dans la contrainte Non-Chevauchement basé sur le Non-Chevauchement MDD introduit dans [5]. Une version relâchée en temps polynomial est obtenue en relâchant le MDD, c.-à-d. en bornant sa largeur. Des expériences sur un problème d'ordonnancement juste-à-temps sur une machine unique ont montré que la nouvelle méthode de filtrage fournit un élagage supplémentaire par rapport au filtrage standard pour la contrainte no-overlap. Les expériences montrent également que, comparée à l'utilisation du filtrage classique seul, l'utilisation redondante du filtrage CB relâché permet d'obtenir un filtrage plus proche de la cohérence de borne.

## Références


[1] Philippe BAPTISTE, Marta FLAMINI et Francis SOURD : Lagrangian bounds for just-in-time job-shop scheduling. *Computers & Operations Research*, 35(3):906–915, 2008.

[2] Philippe BAPTISTE, Claude LE PAPE et Wim NUIJTEN : *Constraint-based scheduling : applying constraint programming to scheduling problems*, volume 39. Springer Science & Business Media, 2001.

[3] Nicolas BLAIS, Alexis REMARTINI, Claude-Guy QUIMPER, Nadia LEHOUX et 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 et Eric PINSON : Adjustment of heads and tails for the job-shop problem. *Euro-*

*pean Journal of Operational Research*, 78(2):146–161, 1994.

[5] Andre CIRE et Willem-jan VAN HOEVE : Mdd propagation for disjunctive scheduling. *Proceedings of the International Conference on Automated Planning and Scheduling*, 22:11–19, mai 2012.

[6] Elizabeth D DOLAN et Jorge J MORÉ : Benchmarking optimization software with performance profiles. *Mathematical programming*, 91(2):201–213, 2002.

[7] Hamed FAHIMI, Yanick OUELLET et Claude-Guy QUIMPER : Linear-time filtering algorithms for the disjunctive constraint and a quadratic filtering algorithm for the cumulative not-first not-last. *Constraints*, 23(3):272–293, 2018.

[8] Hamed FAHIMI et Claude-Guy QUIMPER : Linear-time filtering algorithms for the disjunctive constraint. *In Proceedings of the AAAI Conference on Artificial Intelligence*, volume 28, 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 et Pierre SCHAUS : *Conflict Ordering Search for Scheduling Problems*, page 140–148. Springer International Publishing, 2015.

[11] Rebecca GENTZEL, Laurent MICHEL et W.-J. van HOEVE : *HADDOCK : A Language and Architecture for Decision Diagram Compilation*, page 531–547. Springer International Publishing, 2020.

[12] Xavier GILLARD, Pierre SCHAUS et 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 et 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 et 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 et Emmanuel HEBRARD : Solving variants of the job shop scheduling problem through conflict-directed search. *INFORMS Journal on Computing*, 27(2):268–284, 2015.

[16] Emmanuel HEBRARD : Disjunctive scheduling in tempo. *In 31st International Conference on Principles and Practice of Constraint Programming (CP 2025)*, pages 13–1. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2025.

[17] Philippe LABORIE et Jerome ROGERIE : Reasoning with conditional time-intervals. *In FLAIRS*, pages 555–560, 2008.

[18] Philippe LABORIE, Jérôme ROGERIE, Paul SHAW et Petr VILÍM : Ibm ilog cp optimizer for scheduling : 20+ years of scheduling with constraints at ibm/ilog. *Constraints*, 23(2):210–250, mars 2018.

[19] L. MICHEL, P. SCHAUS et P. VAN HENTENRYCK : Minicp : a lightweight solver for constraint programming. *Mathematical Programming Computation*, 13(1):133–184, 2021.

[20] Pierre SCHAUS, Guillaume DERVAL, Augustin DELECLUSE, Laurent MICHEL et Pascal Van HENTENRYCK : Maxicp : A constraint programming solver for scheduling and vehicle routing, 2024.

[21] Mohamed SIALA, Christian ARTIGUES et Emmanuel HEBRARD : Two clause learning approaches for disjunctive scheduling. *In International Conference on Principles and Practice of Constraint Programming*, pages 393–402. Springer, 2015.

[22] Gilles SIMONIN, Christian ARTIGUES, Emmanuel HEBRARD et Pierre LOPEZ : Scheduling scientific experiments for comet exploration. *Constraints*, 20(1):77–99, 2015.

[23] Sascha VAN CAUWELAERT, Michele LOMBARDI et Pierre SCHAUS : *Understanding the Potential of Propagators*, page 427–436. Springer International Publishing, 2015.

[24] Hélène VERHAEGHE, Roger KAMEUGNE, Christophe LECOUTRE et Pierre SCHAUS : Improved filtering of scheduling problems using redundant table constraints. *In 20th workshops on Constraint Modelling and Reformulation (ModRef2021)*, 2021.

[25] Petr VILÍM, Roman BARTÁK et Ondřej ČEPEK : Extension of o(n log n) filtering algorithms for the unary resource constraint to optional activities. *Constraints*, 10(4):403–425, octobre 2005.