OptalCP un nouveau moteur d'ordonnancement

Diego Olivier Fernandez Pons[1] **Petr Vilim**[1]

[1] ScheduleOpt

{diego.olivier.fernandez.pons, petr.vilim}@scheduleopt.com

Résumé

OptalCP est un nouveau moteur d'ordonnancement conçu comme un successeur à IBM ILOG CP Optimizer, et corrigeant trois problèmes qui limitent la performance de CPO : la mauvaise gestion des multiples cœurs des processeurs modernes, la collaboration insuffisante des bornes inférieures avec le reste du moteur, enfin la difficulté à combiner le moteur avec des heuristiques métier. Nous décrivons dans cet article les choix architecturaux faits pour OptalCP et montrons les améliorations de performances obtenues.

Abstract

OptalCP is a state-of-the-art scheduling engine, designed as a successor of IBM ILOG CP Optimizer and correcting three problems that limit CPO's performance: poor management of multi-core hardware, poor usage of the information provided by the lower bounds, and difficulty to combine the engine with domain-knowledge based heuristics.

1 Introduction

OptalCP est un moteur d'ordonnancement conçu comme un successeur à IBM CP Optimizer (CPO) dont il hérite le style et principaux concepts, tout en améliorant certaines faiblesses de CPO, notamment le parallélisme, les bornes inférieures et l'interaction avec des heuristiques métier.

La principale nouveauté d'OptalCP 2024.1 est une architecture construite autour des bornes inférieures et du parallélisme qui permet

- l'utilisation efficace de la puissance supplémentaire fournie par l'ajout de cœurs de calcul (section 3)
- l'utilisation des bornes inférieures pour améliorer le fonctionnement des autres composants du moteur (sections 3.3, 3.4)
- l'hybridation coopérative du moteur avec des heuristiques ou même d'autres moteurs (section 4)

OptalCP peut résoudre des problèmes complexes comme de l'ordonnancement d'usines de production de semi-conducteurs, de produits intermédiaires (métaux, fibre de verre) ou de consommation (bières, yaourts), ordonnancement de projets pour la construction civile ou navale, conception d'usines, ou création d'emplois du temps pour infrastructures (aéro-)portuaires et transport maritime.

2 Composants d'un moteur d'ordonnancement

2.1 Concepts d'ordonnancement

OptalCP supporte quasiment les mêmes concepts et contraintes que ILOG IBM CP Optimizer [5]

- **Intervalles** de temps optionels avec un début (`start`), une fin (`end`) et une durée (`length`)
- **Sequences** d'intervalles (`sequence`)
- Expressions **cumulatives** construites à partir d'intervalles (`cumulative`)
- **Alternatives** entre intervalles (`alternative`)
- **Unions** d'intervalles (`span`)
- Les contraintes qui s'y appliquent comme les précédences, les temps de transition, limites pour les cumulatives, etc.

Ces concepts sont désormais standards dans le domaine de l'ordonnancement, et de ce fait sont disponibles sous des formes assez proches dans des moteurs aussi divers que OR-tools, miniCP ou Choco. Nous invitons donc le lecteur à se référer aux références [3][7] pour plus de détails.

2.2 Implémentation des contraintes

OptalCP 2024.1 n'introduit que des améliorations mineures aux implémentations publiées des contraintes d'ordonnancement [9]

Les expressions cumulatives utilisent des algorithmes différents en fonction de comment elles sont construites :

- **Cas disjonctif** (`noOverlap`) : les intervalles doivent être disjoints, ce que l'on peut voir comme une expression cumulative dont la somme de contributions +1 au début et -1 à la fin de chaque intervalle doit être $C \in [0, 1]$
- **Cas cumulatif** (`cumul` avec `pulse`) : l'expression cumulative est la somme de contributions +h au début et -h à la fin d'un intervalle (opérateur `pulse`), encadrée par $[a, b]$ avec $a \leq b \in \mathbb{Z}$ [1]
- **Cas réservoir ou inventaire** (`cumul` avec `stepAtStart` et `stepAtEnd`) : l'expression cumulative est la somme de contributions arbitraires du début et fin d'un intervalle, encadrée par $[a, b]$ avec $a \leq b \in \mathbb{Z}$

Les autres contraintes sont implémentées de façon basique ; contrairement à d'autres moteurs d'ordonnancement, OptalCP n'agrège pas les précédences en un graphe acyclique global, ne contient pas d'algorithme de propagation des implications par 2-SAT, ou d'algorithme pour agréger puis propager les contraintes de différence (`alldiff`), etc.

2.3 Implémentation de la recherche

Dans la lignée de ILOG IBM CP Optimizer, OptalCP fournit une recherche automatique simplement paramétrable, mais pas de moyens de décrire précisément une stratégie de recherche spécifique à un problème.

Les stratégies de recherche prédéfinies sont

- Recherche locale arborescente (LNS) [4]
- Recherche arborescente dirigée par les échecs (Failure Directed Search) [10] déclinée en version descendante (FDS) et ascendante (FDSLB)

On peut considérer que LNS est une adaptation de la recherche locale aux moteurs de programmation par contraintes, que FDS est l'extension du principe **first-fail** et des stratégies du type **min-domain**, et que FDSLB est l'extension à des problèmes d'ordonnancement plus généraux des bornes inférieures destructives (**destructive lower bounds**) issues du RCPSP [2].

Toutes les stratégies de recherche sont adaptatives, autrement dit elles partent d'un ensemble de paramètres prédéfinis qu'elles ajustent en fonction de leur succès sur le problème en cours de résolution.

1. La limitation de CP Optimizer dont les cumulatives doivent être positives a donc été levée

3 Parallélisme

Plusieurs études signalent la difficulté de IBM ILOG CP Optimizer à utiliser efficacement les processeurs multi-cœurs, par exemple [1] :

> « *ORT[ools] seems to benefit slightly from additional cores, while CPO is better off with the single-core configuration* »

A contrario, OptalCP a été conçu pour exploiter les processeurs multi-cœur modernes.

3.1 Parallélisme par intensification

Deux choix on été considérés pour l'implémentation du parallélisme.

- Parallélisme par **partition** : L'ensemble des nœuds non-explorés est reparti entre les différents cœurs. La duplication du travail (cœurs explorant le même voisinage) est impossible.
- Parallélisme par **intensification** : Chaque cœur choisit dynamiquement la région de l'espace à explorer, ces régions peuvent se recouvrir partiellement ou totalement, sans aucun mécanisme statique pour empêcher la duplication des efforts.

Après expérimentations préliminaires, le parallélisme par intensification a été choisi pour OptalCP. Cette forme de parallélisme n'est efficace que si de grandes régions de l'espace de recherche peuvent être écartées sans les traverser, autrement dit si l'on dispose de **preuves locales d'optimalité** (en réalité preuves qu'une region ne contient pas de solution meilleure que la meilleure solution déjà disponible). Un système de **no-goods** réduit la duplication des efforts de recherche.

3.2 Preuves locales d'optimalité

Dans OptalCP les preuves locales d'optimalité sont fournies par les solutions et bornes inférieures, ces dernières étant conditionnelles à une région de l'espace de recherche. Quand la borne inférieure d'une région dépasse la meilleure solution connue, on a la preuve qu'il est inutile de parcourir cette région. Cette approche est semblable dans le principe à celle des moteurs MIP dont le LP fournit une borne inférieure locale à une région (le sous-arbre enraciné dans le nœud où le LP est calculé). Mais dans OptalCP, les voisinages utilisés pour le calcul des bornes inférieures sont décorrélés des voisinages utilisés pour le calcul des solutions : chaque stratégie de recherche a son arbre de recherche propre ; ces arbres ne sont synchronisés qu'au moyen des domaines et des no-goods.

3.3 Stratégies de recherche pour le parallélisme

Ce sont ces besoins qui justifient le choix des stratégies de recherche proposées par OptalCP

— La stratégie de recherche LNS génère des solutions par recherche locale arborescente [2]

— Les stratégies de recherche FDS et FDSLB génèrent des bornes inférieures, des no-goods et parfois des solutions [3]

Les essais sur une batterie de problèmes d'ordonnancement indiquent que dédier la moitié des cœurs à LNS et l'autre aux variantes de FDS est un choix efficace et robuste.

3.4 Les bornes inférieures intensifient les heuristiques

Intuitivement on est tenté de croire que par rapport à LNS+FDS, affecter tous les cœurs à LNS fournirait plus rapidement des solutions de bonne qualité mais que le moteur s'arrêterait de progresser au bout d'un moment. En réalité LNS+FDS domine dans presque tous les tests, même pour trouver rapidement des solutions.

Une explication plausible est la **monotonie statistique** des heuristiques. Nous disons qu'une heuristique h est **monotone** si la différence à l'optimum est moindre quand le nombre de choix que l'heuristique doit considérer est plus petit. Autrement dit, plus le problème / les domaines sont petits, plus l'heuristique tombe près de l'optimum.

Plus rigoureusement, si $\text{opt}(\mathcal{A})$ est l'optimum du problème dans le sous ensemble $\mathcal{A}$, une heuristique est monotone si $\mathcal{A} \subseteq \mathcal{B} \Rightarrow | \text{opt}(\mathcal{A}) - h(\mathcal{A}) | \leq | \text{opt}(\mathcal{B}) - h(\mathcal{B}) |$. Une heuristique est statistiquement monotone si cette propriété est vraie en moyenne.

La plupart des heuristiques usuelles sont statistiquement monotones : plus le problème est petit, plus elles tombent près de l'optimum - c'est en particulier le cas de la recherche LNS incomplète utilisée dans OptalCP. La combinaison de solutions et bornes inférieures locales élimine des régions de l'espace de recherche, donc des choix d'affectation des variables, et rend les heuristiques plus efficaces.

Quand on utilise LNS+FDS, la réduction de l'espace de recherche effectuée par FDS permet à LNS de se concentrer sur des régions plus prometteuses : FDS **intensifie** LNS. Dans OptalCP on gagne davantage en guidant LNS vers des régions prometteuses avec (les réductions de domaines créées par les bornes inférieures calculées par) FDS, qu'en dupliquant LNS.

3.5 Implémentation du parallélisme

OptalCP implémente un système de message-passing bidirectionnel pour les cœurs : chaque cœur échange bornes inférieures, solutions et no-goods avec les autres cœurs. Un des cœurs fait office de relais pour les messages (en plus d'exécuter l'heuristique qui lui est attribuée).

Un système de no-goods est utilisé pour partager entre les cœurs de l'information sur les parties de l'espace de recherche qui ont déjà été explorées, réduisant de ce fait la duplication des efforts.

3.6 Non-déterminisme fondamental d'OptalCP

Les processeurs multi-cœur sont fondamentalement non-déterministes : un changement minime de la charge des cœurs peut altérer l'ordre dans lequel les branches sont explorées, les solutions sont trouvées, les informations entre cœurs sont partagées, aboutissant à un résultat différent.

Les moteurs IBM ILOG Cplex et CP Optimizer ont un **mode déterministe** qui crée des points d'attente où les différents cœurs doivent se rencontrer ce qui permet d'obtenir la même solution à chaque exécution du moteur, au prix d'avoir des cœurs inoccupés. Ils ont également un mode **opportuniste** qui maximise l'occupation des cœurs au détriment du déterminisme.

OptalCP est entièrement opportuniste et n'a aucun mécanisme pour garantir que la même solution sera trouvée dans des exécutions successives du moteur.

Afin d'aider l'utilisateur à étudier les performances de son problème, OptalCP dispose un **mode benchmark** qui permet d'exécuter un ensemble de modèles sur un ensemble de données, et fournit un sommaire détaillé des résultats (valeur et temps de chaque solution trouvée, moyenne et écart types des temps d'exécution, des solutions obtenues).

3.7 Déterminisme et validation de modèles

Le mode déterministe de CP Optimizer est souvent utilisé pour construire et valider incrémentalement un modèle :

— un premier modèle minimal est testé sur des données minimales, l'utilisateur vérifie sa correction sur la solution optimale

— l'utilisateur ajoute des contraintes progressivement tout en validant que la solution optimale évolue comme attendu

Ce besoin devra être comblé dans les versions futures d'OptalCP par une approche différente, en effet les

2. Le fait que la recherche arborescente du voisinage est incomplète, la petite taille du voisinage et sa définition par relaxation d'une solution existante rendent complexe la génération de no-goods par la stratégie LNS

3. En essayant de prouver qu'une affectation $x = C$ est impossible parfois FDS ne réussit pas et à la place génère une solution avec $x = C$

choix faits pour le parallélisme de OptalCP rendent difficile la création d'un mode déterministe.

4 Ajout de connaissances métier

Contrairement à CP Optimizer qui permet de paramétrer partiellement la recherche du moteur par des phases de recherche (quelles variables doivent être fixées en premier) ou des évaluateurs (quelles variables ou valeurs choisir pour brancher), OptalCP ne fournit aucun moyen de modifier les stratégies de recherche autre qu'une paramétrisation simple du niveau de filtrage des contraintes.

A la place OptalCP peut être combiné avec des heuristiques externes, avec d'autres moteurs, et avec lui-même.

4.1 Combinaison avec des heuristiques ou moteurs externes

Des heuristiques externes peuvent être combinées avec OptalCP pour résoudre un problème : OptalCP bénéficie ainsi des connaissances métier incorporées dans ces heuristiques, et les heuristiques bénéficient des bornes inférieures et solutions trouvées par OptalCP afin d'affiner leurs résultats, par exemple en exploitant leur monotonie statistique.

Ces heuristiques peuvent être écrites entièrement en dehors d'OptalCP, dans le langage de choix de l'utilisateur. Elles doivent juste implémenter l'interface de message-passing bidirectionnel du moteur, après quoi elles seront traitées par le moteur comme un cœur supplémentaire. Pour l'instant cependant OptalCP ne partage pas les no-goods avec les heuristiques externes.

4.2 Solutions bi-modèle

OptalCP peut être hybridé avec lui-même afin de résoudre un problème avec deux modèles différents

— un modèle complet pour le problème
— un modèle contenant des contraintes simplificatrices

L'idée de l'approche bi-modèle est que le modèle simplifié a une plus grande probabilité de trouver très rapidement de bonnes solutions, donnant un point de départ proche de l'optimum au modèle complet.

Dans CP Optimizer un résultat semblable s'obtient par la résolution successive du modèle simple, puis l'injection des solutions dans le modèle complet via les **warm-start**.

5 Performances d'OptalCP

5.1 Amélioration du parallélisme

La figure 1 montre sur le problème de job-shop `dmu2` comment le temps de calcul pour trouver et prouver l'optimum décroît avec l'augmentation du nombre de cœurs, tandis que le nombre total de branches explorées par seconde (globalement par le moteur) croit presque linéairement avec le nombre de cœurs.

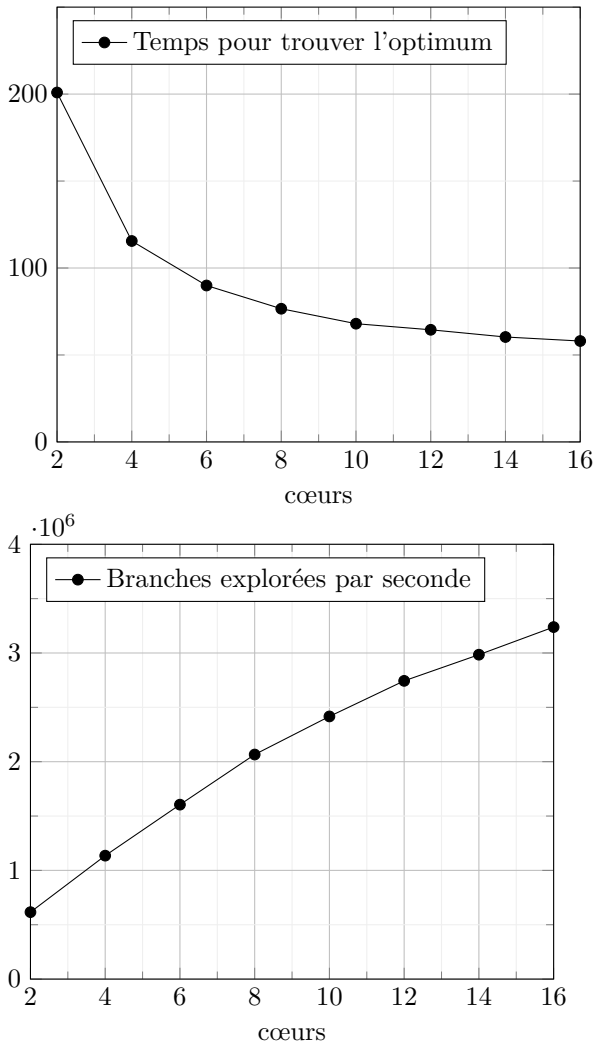

FIGURE 1 – Performances du parallélisme d'OptalCP illustrée sur le problème de job-shop dmu02 avec une machine à 16 cœurs

5.2 Amélioration des bornes inférieures

OptalCP améliore 79 des meilleures bornes inférieures connues pour divers problèmes de job-shop, et de ce fait prouve l'optimalité de 4 nouvelles instances du benchmark `dmu`.

— **abz** (1 borne) : 08
— **dmu** (51 bornes) : 01, 02, 04, 06-11, 17, 19, 20, 41-79
— **swv** (5 bornes) : 06-10
— **taillard js** (17 bornes) : 18, 22, 23, 25-27, 29, 30, 33, 40-42, 44, 46-49
— **yn** (5 bornes) : 2-4

La table 1 montre les résultats d'OptalCP sur les instances **dmu** en 1h de calcul par instance sur un i7-1185G7 comptant 4 cœurs, cadencé à 3Ghz avec 16Gb RAM

TABLE 1 – Résultats d'OptalCP sur les instances **dmu**

dmu	LB	UB	dmu	LB	UB
01	2563	2563	41	3138	3305
02	2706	2706	42	3312	3450
03	2731	2731	43	3386	3463
04	2628	2669	44	3387	3547
05	2749	2749	45	3188	3338
06	3165	3294	46	3726	4169
07	2982	3076	47	3671	4064
08	3188	3188	48	3588	3870
09	3092	3092	49	3507	3753
10	2984	2984	50	3577	3835
11	3397	3457	51	4053	4296
12	3481	3546	52	4183	4496
13	3681	3723	53	4235	4646
14	3394	3394	54	4271	4492
15	3343	3343	55	4181	4388
16	3734	3769	56	4711	5160
17	3718	3894	57	4431	4926
18	3844	3855	58	4439	4950
19	3698	3805	59	4337	4845
20	3618	3760	60	4443	4966
21	4380	4380	61	5020	5358
22	4725	4725	62	5164	5420
23	4668	4668	63	5230	5547
24	4648	4648	64	5148	5580
25	4164	4164	65	5122	5457
26	4647	4697	66	5510	5933
27	4848	4848	67	5650	6043
28	4692	4692	68	5491	6132
29	4691	4691	69	5500	6043
30	4732	4779	70	5616	6073
31	5640	5640	71	6128	6612
32	5927	5927	72	6430	6769
33	5728	5728	73	6107	6495
34	5385	5385	74	6168	6489
35	5635	5635	75	6122	6527
36	5621	5621	76	6454	7028
37	5851	5851	77	6518	7127
38	5713	5713	78	6639	7133
39	5747	5747	79	6708	7386
40	5577	5577	80	6459	7042

Tous ces résultats peuvent être reproduits avec la version d'évaluation d'OptalCP disponible en téléchargement sur le site du moteur https://optalcp.com

5.3 Comparaison avec CPO

Le site web d'OptalCP compare les performances d'OptalCP et CP Optimizer sur 8 problèmes d'ordonnancement avec des contraintes diverses (job-shop, job-shop avec nombre maximal d'opérateurs, flexible job-shop, job-shop TT, open-shop, rcpsp, mmrcpsp et rcpsp-cpr).

Le lecteur intéressé y trouvera des statistiques très détaillées sur leur performances relatives. Par exemple pour le job-shop

```
Number of instances: 242
objective(OptalCP) < objective(CP Optimizer): 92
objective(OptalCP) = objective(CP Optimizer): 145
objective(OptalCP) > objective(CP Optimizer): 5
When objectives are the same (145 times):
sol. time(OptalCP) < sol. time(CP Optimizer): 140
sol. time(OptalCP) > sol. time(CP Optimizer): 5
duration(OptalCP) < duration(CP Optimizer): 134
duration(OptalCP) > duration(CP Optimizer): 11
```

5.4 Comparaison avec l'état de l'art

Les instances utilisées pour développer et ajuster les performances d'OptalCP sont progressivement mises en ligne sur Github (https://github.com/ScheduleOpt/benchmarks/).

Un certain nombre de ces instances sont des instances classiques, pour lesquelles de nombreuses méthodes ont été testées au cours du temps. Par exemple pour les instances de job-shop et flow-shop, les meilleurs résultats connus ont été obtenus par les approches suivantes : branch & bound, guided local search, recherche taboue, programmation dynamique, recuit simulé, SAT, algorithmes génétiques, CP Optimizer et OptalCP.

Le site web dénombre par publication / moteur le nombre de meilleures bornes détenues. Ces résultats sont à considérer avec prudence car ils ont été obtenus sur une période de plus de 30 ans et que les différentes méthodes ne sont pas équivalentes dans leur niveau de généralité, disponibilité ou facilité de mise en oeuvre. Le lecteur intéressé trouvera plus de détails dans la section méthodologie du site.

6 Un jeu d'instances de test public

Le site web regroupant les instances utilisées pour tester OptalCP fournit pour chaque type de problème
— Les instances dans un format standardisé, simple à lire, classifiées par publication

— Une description des formats alternatifs que l'on peut rencontrer
— Une description mathématique du problème et ses variantes
— Une classification des instances basées sur le temps nécessaire pour les résoudre avec un moteur d'ordonnancement : simple (*easy*) $\leq$ 1 min, moyen (*medium*) $\leq$ 1 heure, difficile (*hard*) dont l'optimum est connu mais pas prouvable par un moteur d'ordonnancement, ou ouverte (*open*)
— Les meilleures bornes inférieures et supérieures connues avec les références correspondantes
— Des instances intéressantes pour tester des moteurs d'optimisation par exemple des instances petites mais ouvertes, ou très grandes, etc.
— Une liste de publications d'intérêt pour le problème

Le site web est fortement inspiré du travail de Naderi, Ruiz and Roshanaei [6] lequel compare CP Optimizer, Cplex et Gurobi sur une batterie de 17 problèmes et 6623 instances avec un temps de calcul de 1 heure. Leurs résultats sont disponibles sous forme de feuille de calcul à l'adresse `http://soa.iti.es/problem-instances`

6.1 Sites similaires

D'autres sites qui collectent des problèmes d'ordonnancement et meilleures solutions connues sont
— Le site web de Taillard [8]
— Le site web Jobshop `http://jobshop.jjvh.nl/index.php`
— Optimizizer `https://optimizizer.com/jobshop.php`
— Le site de Thomas Weise `https://github.com/thomasWeise/jsspInstancesAndResults`

Nous avons collecté toutes les instances et solutions disponibles sur ces sites.

7 Conclusion

OptalCP est nouveau un moteur d'ordonnancement reprenant certains des éléments qui ont fait le succès de IBM ILOG CP Optimizer, tout en améliorant ses limitations comme le parallélisme, les bornes inférieures et l'hybridation avec d'autres moteurs ou heuristiques. Nous avons décrit les choix architecturaux qui ont été faits pour OptalCP, montrant à la fois les bénéfices en termes de performance et fonctionnalité, mais aussi les limitations qui en résultent. On pourra trouver sur les site web d'OptalCP (`www.optalcp.com`) des comparatifs de performances, la documentation du moteur, et un lien vers l'ensemble de problèmes de test public servant à son développement.

Références

[1] Giacomo Da Col et Erich C. Teppan : Industrial-size job shop scheduling with constraint programming. *Operations Research Perspectives*, 9:100249, 2022.

[2] Robert Klein et Armin Scholl : Computing lower bounds by destructive improvement : An application to resource-constrained project scheduling. *European Journal of Operational Research*, 112(2):322–346, 1999.

[3] Philippe Laborie : IBM ILOG CP Optimizer for detailed scheduling illustrated on three problems. *In Integration of AI and OR Techniques in Constraint Programming : 6th International Conference, CPAIOR 2009 Pittsburgh, PA, USA, May 27-31, 2009 Proceedings 6*, pages 148–162. Springer, 2009.

[4] Philippe Laborie et Daniel Godard : Self-adapting large neighborhood search : Application to single-mode scheduling problems. *Proceedings MISTA-07, Paris*, 8, 2007.

[5] 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*, 23:210–250, 2018.

[6] Bahman Naderi, Rubén Ruiz et Vahid Roshanaei : Mixed-integer programming vs. constraint programming for shop scheduling problems : New results and outlook. *INFORMS Journal on Computing*, 2023.

[7] ScheduleOpt : OptalCP tutorial, 2024.

[8] Eric Taillard : Benchmarks for basic scheduling problems, 2015.

[9] Petr Vilím : Timetable edge finding filtering algorithm for discrete cumulative resources. *In International Conference on AI and OR Techniques in Constriant Programming for Combinatorial Optimization Problems*, pages 230–245. Springer, 2011.

[10] Petr Vilím, Philippe Laborie et Paul Shaw : Failure-directed search for constraint-based scheduling. *In Integration of AI and OR Techniques in Constraint Programming : 12th International Conference, CPAIOR 2015, Barcelona, Spain, May 18-22, 2015, Proceedings 12*, pages 437–453. Springer, 2015.