Une méthode exacte pour le problème d'ordonnancement d'atelier avec temps de préparation

Christian Artigues, Dominique Feillet

Une méthode exacte pour le problème d'ordonnancement d'atelier avec temps de préparation

Christian Artigues[1,2] **et Dominique Feillet**[1]

[1] LIA – Université d'Avignon
339 chemin meinajariés, Agroparc, BP 1228, 84911 Avignon Cedex 9, FRANCE

[2] CRT – Université de Montréal
C.P. 6128, succursale Centre-ville, Montréal, QC H3C 3J7 CANADA
{prénom.nom}@univ-avignon.fr

Résumé

Nous présentons une nouvelle méthode exacte pour résoudre le problème d'ordonnancement d'atelier avec temps de préparation dépendant de la séquence. Pour résoudre ce problème NP-difficile au sens fort, nous proposons une méthode de recherche arborescente où, à chaque nœud, des techniques de propagation de contraintes temporelles et de ressources sont appliquées et la relaxation du problème à un ensemble de problèmes de voyageurs de commerce avec fenêtres de temps est résolue par programmation dynamique. Testée sur les problèmes proposés par Brucker et Thiele [10], la méthode améliore significativement les meilleures bornes inférieures et supérieures connues sur la plupart des instances auparavant non résolues

1 Introduction

Le problème d'ordonnancement d'atelier, qui consiste à ordonnancer des travaux sur un ensemble de machines, a été très étudié dans la littérature et beaucoup de méthodes performantes ont été proposées pour sa résolution [8]. La variante que nous considérons ici, notée SDST-JSP[1], considère le cas, fréquent en pratique, où les machines peuvent avoir à subir un certain temps de préparation (ou de reconfiguration) entre deux opérations consécutives. Nous cherchons ici à résoudre de manière exacte le problème de minimisation de la durée totale de l'ordonnancement.

La méthode que nous proposons est basée de manière classique sur la résolution d'une suite de problèmes de satisfaction de contraintes (CSP[2]) par recherche arborescente.

A chaque nœud de chaque arbre de recherche, des algorithmes de propagation de contraintes adaptés à la présence de temps de préparation sont utilisés pour filtrer les domaines des variables de décision et vérifier la cohérence du CSP correspondant. Lorsque la propagation des contraintes ne permet pas de détecter d'incohérence, nous cherchons à résoudre la relaxation du SDST-JSP en plusieurs problèmes du voyageur de commerce avec fenêtres de temps (TSPTW[3]).

Le problème considéré est présenté en 2 et la relaxation en 3. Un survol de la littérature est effectué en 4. La méthode proposée est décrite en 5. Les résultats de la méthode exacte sur les instances proposées par Brucker et Thiele [10] sont présentés en 6. Une conclusion et quelques perspectives de poursuite de ces travaux sont données en 7.

2 Formulation du problème

Le SDST-JSP comporte un ensemble de $n \geq 1$ travaux $\mathcal{J} = \{J_i\}_{1 \leq i \leq n}$ et un ensemble de $m \geq 1$ machines $\mathcal{M} = \{M_k\}_{1 \leq k \leq m}$. Chaque travail $J_i \in \mathcal{J}$ est défini comme un ensemble de m opérations $J_i = \{O_{ij}\}_{1 \leq j \leq m}$. Chaque opération O_{ij} a une du-

[1] job-shop problem with sequence dependent setup times

[2] constraint satisfaction problem

[3] traveling salesman problem with time windows

rée entière non négative $p_{ij} \in \mathbb{N}$ et nécessite pour son exécution une et une seule machine $m_{ij} \in \mathcal{M}$. Les opérations d'un même travail sont affectées à des machines distinctes et sont soumises à des contraintes de précédence. Ainsi, l'opération O_{ij} précède l'opération $O_{i,j+1}$ pour tout $i = 1, \ldots, n$ et pour tout $j = 1, \ldots, m-1$. L'ensemble des opérations de tous les travaux est noté $\mathcal{O} = \cup_{1 \leq i \leq n} J_i$ et $\mathcal{O}_k$ désigne l'ensemble des opérations affectées à la machine M_k. Les opérations doivent être exécutées sans interruption.

Un temps de préparation dépendant de la séquence, noté s_{ijk}, est défini pour tout couple de travaux distincts (J_i, J_j) et pour chaque machine M_k. Un temps de préparation initial s_{0ik} est défini pour chaque travail J_i et pour chaque machine M_k. Nous supposons que l'inégalité triangulaire est vérifiée : pour chaque machine M_k et pour chaque triplet de travaux distincts (J_i, J_j, J_x), l'inégalité $s_{ixk} \leq s_{ijk} + s_{jxk}$ est vérifiée ; pour chaque machine M_k et pour chaque couple de travaux distincts (J_i, J_j), l'inégalité $s_{0jk} \leq s_{0ik} + s_{ijk}$ est vérifiée.

Un ordonnancement est donné par une affectation de valeurs aux variables $\mathcal{T} = (t_{ij})_{O_{ij} \in \mathcal{O}}$ où $t_{ij} \in \mathbb{N}$ désigne la date de début de l'opération O_{ij}. Dans ce papier l'objectif de minimisation de la durée totale est considéré. Ce problème particulier d'optimisation, noté $J|s_{ij}|C_{\max}$ dans la notation standard des problèmes d'ordonnancement, est noté (P) dans ce qui suit.

$$\text{(P)} \quad \text{minimiser } C_{\max} \tag{1}$$

sous les contraintes :

$$C_{\max} \geq C_{im} \qquad \forall J_i \tag{2}$$

$$C_{ij} = t_{ij} + p_{ij} \qquad \forall O_{ij} \tag{3}$$

$$C_{ij} \leq t_{i,j+1} \qquad \forall O_{ij}, j < m \tag{4}$$

$$C_{ij} + s_{ixk} \leq t_{xy} \vee C_{xy} + s_{xik} \leq t_{ij} \qquad \forall \{O_{ij}, O_{xy}\} \in \mathcal{O}_k \tag{5}$$

$$t_{ij} \geq s_{0im_{ij}} \qquad \forall O_{ij} \tag{6}$$

L'objectif (1) est la minimisation de la durée totale $C_{\max}$, la plus grande date de fin des travaux (2). Les contraintes (3) lient les dates de fin et les dates de début des opérations. Les contraintes (4) sont les contraintes de précédence. Les contraintes (5) représentent les limitations de ressources par des disjonctions. On parle de ressources disjonctives pour désigner les machines. Ainsi, si O_{ij} et O_{xy} sont deux opérations distinctes affectées à la machine k ou bien O_{ij} doit commencer après la fin de O_{xy} plus le temps de préparation s_{xik}, ou bien O_{xy} doit commencer après la fin de O_{ij} plus le temps de préparation s_{ixk}. Les contraintes (6) imposent qu'aucune opération ne peut commencer avant le temps de préparation initial.

Une représentation alternative des temps de préparation consiste à définir un ensemble $\mathcal{F}$ de $f \geq 1$ familles telles que toute opération $O_{ij} \in \mathcal{O}$ appartient à une certaine famille $f_{ij} \in \mathcal{F}$. Le temps de préparation s_{ixk} sur la machine M_k d'une opération $O_{ij} \in \mathcal{O}_k$ du travail J_i vers une opération $O_{xy} \in \mathcal{O}_k$ du travail J_x peut être simplement écrit $s_{f_{ij}f_{xy}}$. Cette représentation peut être intéressante lorsque le nombre de familles est significativement plus petit que le nombre d'opérations. Des techniques efficaces de prétraitement peuvent alors être effectuées (voir 5.2.1).

Le SDST-JSP est un problème NP-difficile au sens fort, en tant qu'extension du problème standard de job-shop noté $J||C_{\max}$, obtenu en considérant que tous les temps de préparation sont nuls.

Dans ce papier, la solution optimale de (P) est cherchée en résolvant successivement des variantes décisionnelles, c'est-à-dire des problèmes de satisfaction de contraintes (FP) définis comme suit. Soit $T \geq 0$. Le problème $FP(T)$ consiste à trouver un ordonnancement $\mathcal{T}$ tel que $C_{max} \leq T$ et vérifiant les contraintes (2-6). Soit UB la durée d'un ordonnancement réalisable pour les contraintes (2-6). Soit LB une borne inférieure de la solution optimale de (P). Soit $C^*_{\max}$ la solution optimale de (P). On a alors :

$$C^*_{\max} = \min_{LB \leq T \leq UB} \{T | FP(T) \text{ est réalisable}\} \tag{7}$$

Nous utilisons un graphe disjonctif [26] pour représenter les solutions partielles et complètes du problème. Le graphe comporte un sommet par opération, plus deux sommets fictifs représentant le début et la fin de l'ordonnancement. Il comporte un ensemble d'arcs U représentant les contraintes de précédence et valués par la durée de l'opération origine, et un ensemble d'arêtes E représentant les contraintes de ressources disjonctives. Par définition, une sélection est un état du graphe disjonctif pour lequel une orientation a été choisie pour certaines arêtes. Une sélection est dite complète si chaque arête possède une direction. On suppose qu'une fois l'arête orientée, l'arc obtenu est valué par la durée de l'opération origine de l'arc, plus le temps de préparation nécessaire entre les deux opérations sur la machine. On peut alors montrer qu'il existe une bijection entre l'ensemble dominant des ordonnancements semi-actifs (dans lesquels aucune opération ne peut être décalée d'une unité de temps vers la gauche, en laissant les autres opérations fixées, sans violer une contrainte) et l'ensemble des sélections complètes telles que le graphe orienté obtenu soit sans circuit. La date de début

d'une opération dans l'ordonnancement semi-actif est égale à la longueur du plus long chemin dans ce graphe entre le sommet origine et le sommet représentant l'opération.

3 Relaxation en TSPTW

Si $m = 1$, (P) peut être noté $1|s_{ij}|C_{max}$ et est équivalent au problème du voyageur de commerce. Lorsqu'il y a plusieurs machines, cette relaxation peut être renforcée en calculant des fenêtres de temps pour chacune des opérations, définies par r_{ij} la date de début au plus tôt de l'opération O_{ij} et d_{ij} sa date de fin au plus tard. Une façon simple de calculer des dates de début au plut tôt valides est de poser

$$r_{i1} = s_{0im_{i1}} \tag{8}$$

pour la première opération de tout travail J_i puis

$$r_{ij} = \max(s_{0im_{ij}}, r_{i(j-1)} + p_{i(j-1)}) \tag{9}$$

pour les opérations suivantes O_{ij} avec $j > 1$. De même, des dates de fin au plus tard valides peuvent être calculées à partir d'une borne supérieure UB en posant

$$d_{im} = UB \tag{10}$$

pour la dernière opération de tout travail J_i puis

$$d_{ij} = d_{i(j+1)} - p_{i(j+1)} \tag{11}$$

pour les opérations précédentes O_{ij} avec $j < m$. Toute solution de (P) de durée totale inférieure ou égale à UB doit vérifier obligatoirement $r_{ij} \leq t_{ij} \leq d_{ij} - p_{ij}$ pour chaque opération $O_{ij} \in \mathcal{O}$. En 5.2, nous présentons des algorithmes de propagation de contraintes pour resserrer ces fenêtres qui représentent en fait les domaines des variables du CSP considéré. Soit la relaxation de (P) obtenue en considérant uniquement les opérations de la machine M_k. On obtient un problème d'ordonnancement à une machine avec fenêtres de temps noté $1|r_i, d_i, s_{ij}|C_{max}$, dont la solution est une borne inférieure de (P). Si les temps de préparation sont tous nuls, le problème est déjà NP-difficile mais peut être résolu efficacement par l'algorithme de Carlier [13]. Sinon, il s'agit d'une variante du problème de voyageur de commerce avec fenêtre de temps (TSPTW) où l'objectif est la minimisation du temps total de trajet augmenté des temps d'attente et de service [2, 5, 19]. Soit $C^*_{\max}(\text{TSPTW}_{\mathcal{O}_k})$ la solution optimale de la relaxation de TSPTW en considérant uniquement la machine M_k. On obtient pour (P) une borne inférieure LB_{TSPTW} comme suit.

$$LB_{\text{TSPTW}} = \max_{M_k \in \mathcal{M}} (C^*_{\max}(\text{TSPTW}_{\mathcal{O}_k})) \tag{12}$$

Si nous considérons maintenant la variante décisionnelle $FP(T)$, la relaxation devient la recherche d'une solution réalisable au TSPTW décrit ci-dessus en posant $UB = T$. Ce problème (noté F-TSPTW par la suite) est également NP-difficile.

4 Revue de la littérature

Le problème de SDST-JSP n'a pas reçu beaucoup d'attention dans la littérature bien qu'il représente une extension naturelle du problème d'atelier classique avec beaucoup d'applications industrielles [1]. Brucker et Thiele [10] ont par ailleurs généré 15 instances du problème de SDST-JSP qui seront désignées par BT et qui nous serviront de base d'évaluation et de comparaison de nos résultats. Ces instances comportent 5 "petits" problèmes, 5 problèmes "moyens" et 5 "grands" problèmes. Quelques méthodes gloutonnes ont été proposées, basées sur la simulation [20] ou sur des extensions d'algorithmes de génération d'ordonnancements actifs [4, 10, 25]. Nous décrivons brièvement en 5.1 la méthode que nous avons utilisée au sein de notre méthode exacte.

Parmi les méthodes de recherche locale et/ou les metaheuristiques on trouve certaines méthodes spécifiques [15, 16, 30], d'autres basées sur le graphe disjonctif comme un algorithme génétique [12], des méthodes tabou [11, 3]. Dans cette dernière catégorie, des heuristiques basées sur le réordonnancement de la machine goulot (Shifting Bottleneck Heuristic) ont été étendues au temps de préparation [5, 24, 27].

A notre connaissance, les seules méthodes exactes précédemment proposées pour le SDST-JSP sont celles de Brucker et Thiele [10], Focacci *et al* [18] et Artigues *et al* [2].

Brucker et Thiele [10] proposent une procédure de séparation et évaluation. Le problème considéré est le problème d'atelier général avec temps de préparation qui inclue le SDST-JSP comme cas particulier. La méthode est basée sur le graphe disjonctif et chaque nœud de l'arbre de recherche correspond à une sélection partielle. Des bornes inférieures sont calculées à chaque nœud en appliquant plusieurs extensions aux temps de préparation de l'algorithme de Jackson préemptif [14]. A chaque nœud, des algorithmes de propagation de contraintes de ressources sont appliqués pour resserrer les fenêtres de temps et enrichir la sélection partielle. En 5.2.2, nous décrivons notre version des algorithmes de propagation des contraintes de ressources pour le SDST-JSP, inspirée de [10, 23, 29].

Focacci *et al* [18] utilisent la programmation par contraintes pour résoudre exactement une variante

du SDST-JSP avec des ressources alternatives. Ils considèrent en outre un problème d'optimisation bicritère, cherchant à minimiser le temps de préparation total en plus de la durée totale. Les algorithmes de propagation des contraintes de ressources utilisés ignorent les temps de préparation mais une contrainte globale, dite de chemin, dont la propagation est basée sur la relaxation des problèmes de TSPTW en problèmes d'affectation polynomiaux et sur le filtrage par les coûts réduits est proposée [19].

Brucker et Thiele [10] et Focacci *et al* [18] trouvent toutes les solutions optimale des 5 plus petites instances BT. Brucker et Thiele [10] fournissent des bornes inférieures et supérieures pour les moyennes et grandes instances. Focacci *et al* [18] ont amélioré une des bornes supérieures trouvées. L'heuristique de Balas *et al* [5] a obtenu récemment toutes les meilleures bornes supérieures connues sur les 10 instances non résolues.

Une version préliminaire de la méthode exacte proposée ici est décrite dans [2]. L'idée principale est de résoudre de manière exacte à chaque nœud le problème de F-TSPTW plutôt que la relaxation préemptive de Jackson comme dans [10] ou le problème d'affectation comme dans [18]. Le problème de F-TSPTW est formulé comme un problème d'ordonnancement à une machine et résolu par le solveur commercial ILOG Scheduler. Un dictionnaire est utilisé pour mémoriser les solutions réalisables déjà rencontrées et ainsi éviter, en parcourant le dictionnaire, de résoudre inutilement des sous-problèmes. Cette méthode, en fermant pour la première fois deux instances de taille moyenne, a obtenu des résultats encourageants. Nous présentons ici une version plus élaborée où le F-TSPTW est résolu par programmation dynamique sans l'utilisation d'un solveur commercial et où les algorithmes de propagation de contraintes tiennent compte explicitement des temps de préparation.

5 Méthode de recherche arborescente

Nous proposons une méthode de recherche arborescente pour résoudre exactement $FP(T)$, basée sur la représentation du problème par le graphe disjonctif. Comme dans la méthode de Brucker et Thiele [10], un nœud de l'arbre correspond à une sélection partielle $\mathcal{E}$ où $\mathcal{E}$ est un ensemble d'arcs (orientés) obtenus par une orientation complète ou partielle des arêtes E. Un nœud est aussi défini par des fenêtres de temps compatibles avec la sélection courtante. Un nœud ν est ainsi défini par un triplet $(\mathcal{R}, \mathcal{D}, \mathcal{E})$ avec $\mathcal{R} = (r_{ij})_{O_{ij} \in \mathcal{O}}$ et $\mathcal{D} = (d_{ij})_{O_{ij} \in \mathcal{O}}$. L'algorithme 1 décrit la recherche arborescente proposée. A partir d'une durée totale T, il retourne un échec ou un succès et dans le dernier cas un ordonnancement réalisable $\mathcal{T}$ de durée totale inférieure ou égale à T. Les fenêtres de temps sont initialisées à partir de T et des équations (8-11) (étape 1). Le nœud racine est créé avec une sélection vide (étape 2). Un premier algorithme de propagation des contraintes (SHAVING) est appliqué pour détecter au plus tôt l'incohérence du nœud racine ou tout au moins resserrer les fenêtres de temps et insérer des arcs dans la sélection (étape 3). Si SHAVING ne détecte pas d'incohérence, la recherche arborescente est démarrée en initialisant une pile de nœuds BBQ avec le nœud racine (étape 6). Les étapes 8 à 19 sont répétées tant qu'il reste des nœuds non explorés. Pour chaque nœud ν, un second (et plus rapide) algorithme de propagation des contraintes (PROPAGATE) est appliqué (étape 9). Si aucune incohérence n'a été détectée, une méthode de programmation dynamique SOLVETSPTW est utilisée pour résoudre la relaxation de F-TSPTW (étape 10).

Si SOLVETSPTW ne détecte pas d'incohérence, une solution ($\mathcal{T}'$) aux m TSPTW est retournée (ou bien un temps limite est atteint). Une heuristique est alors appliquée (étape 11) pour tenter de calculer à partir de $\mathcal{T}'$ une solution réalisable au problème global. Si l'heuristique trouve une solution $\mathcal{T}$ réalisable de durée totale inférieure ou égale à T, la recherche s'arrête. Sinon la pile de nœud est mise à jour par branchement (étape 15).

Algorithm 1 SEARCH($T, \mathcal{T}$)

```
1: Initialiser R^0 et D^0 avec (8-11) et UB = T
2: ν^0 ← (R^0, D^0, ∅)
3: if SHAVING(ν^0) =échec then
4:    return succès
5: else
6:    BBQ ← {ν^0}
7:    while BBQ n'est pas vide do
8:       ν ←pop(BBQ)
9:       if PROPAGATE(ν) ≠échec then
10:         if SOLVETSPTW(ν, T')≠échec then
11:            T ←HEURISTIC(ν, T')
12:            if C_max(T) ≤ T then
13:               return succès
14:            else
15:               BBQ ←BRANCH(ν, BBQ)
16:            end if
17:         end if
18:      end if
19:   end while
20:   return échec
21: end if
```

Nous décrivons ci après les différentes composantes de la recherche arborescente. La méthode approchée utilisée pour obtenir des bornes supérieures est décrite en 5.1. Les algorithmes de propagation des contraintes sont décrits en 5.2. Le paragraphe 5.3 donne une formulation du TSPTW comme un problème de plus court chemin élémentaire avec contraintes de ressources et présente l'algorithme de programmation dynamique utilisé pour le résoudre. L'heuristique de branchement est détaillée en 5.4, ainsi que des règles de dominance associées qui permettent de ne pas développer certains nœuds.

5.1 Algorithme glouton

A chaque nœud un algorithme glouton à base de règles de priorité est utilisé pour tenter de trouver une solution réalisable. L'idée est d'utiliser les informations contenues dans le nœud pour guider l'algorithme. La fonction HEURISTIC prend en entrée le nœud courant $\nu = (\mathcal{R}, \mathcal{D}, \mathcal{E})$ et une solution de TSPTW $\mathcal{T}'$. $\mathcal{T}'$ vérifie obligatoirement les contraintes de ressources mais peut violer les contraintes de précédence. HEURISTIC vérifie en premier lieu si un des ordonnancements obtenus en posant $\mathcal{T} = \mathcal{R}$ (on "cale à gauche"), $\mathcal{T} = \mathcal{D} - p$ (on "cale à droite") ou $\mathcal{T} = \mathcal{T}'$ (on teste la solution de TSPTW) est réalisable, auquel cas un succès est retourné. Sinon, la fonction fait appel à l'algorithme glouton SERIALSGS[4] basé sur une règle de priorité π (application de $\mathcal{O}$ dans $\mathbb{N}$) telle que si $\pi_{ij} < \pi_{xy}$ alors O_{ij} est plus prioritaire que O_{xy}. Cet algorithme a été proposé dans [4]. C'est une simple extension de l'algorithme sériel proposé dans [21] pour l'ordonnancement de projet. Son principe est de sélectionner les opérations dans un ordre indiqué par la règle de priorité (mais obligatoirement compatible avec les contraintes de précédence) et d'ordonnancer chaque opération au plus tôt sur sa machine sans violer de contraintes, et éventuellement avant une opération déjà ordonnancée. Contrairement aux méthodes proposées par Brucker et Thiele [10] et Ovacik et Uzsoy [25], l'algorithme est tel qu'une règle de priorité menant à un ordonnancement réalisable existe si un ordonnancement réalisable existe. La complexité temporelle de l'algorithme est de $O(nm^2)$. SERIALSGS est appliqué itérativement 3 fois en choisissant à chaque étape $\pi = \mathcal{R}$, $\pi = \mathcal{D} - p$, $\pi = \mathcal{T}'$. L'algorithme est également utilisé pour calculer une borne supérieure initiale au problème de SDST-JSP en utilisant la règle de priorité MINLFT (plus petite date de fin au plus tard en ignorant les contraintes de ressources).

[4] Serial Schedule Generation Scheme

5.2 Algorithmes de propagation de contraintes

5.2.1 Précalcul des temps de préparation minimaux

Les algorithmes de propagation de contraintes de ressources que nous proposons sont des extensions des algorithmes classiques pour les problèmes disjonctifs sans temps de préparation, tels les sélections immédiates et le edge-finding [6, 9, 14]. Brucker et Thiele [10] et Vilím et Barták [29] ont déjà proposé de telles extensions aux temps de préparation. Ces algorithmes cherchent à détecter de nouvelles contraintes de précédence entre opérations utilisant la même machine et utilisent pour cela les durées minimales d'exécution sans temps morts de sous-ensembles d'opérations $\Omega \subseteq \mathcal{O}_k$, pour chaque machine $M_k \in \mathcal{M}$. Lorsqu'il n'y a pas de temps de préparation, la durée minimale de Ω, notée p_Ω, est évidemment égale à $p_\Omega = \sum_{O_{ij} \in \Omega} p_{ij}$. En présence de temps de préparation la durée minimale de l'ensemble Ω est égale à $p_\Omega + s_\Omega$ où s_Ω est le temps de préparation minimal entre les opérations de Ω si elles sont exécutées consécutivement. Calculer s_Ω est NP-difficile car il s'agit d'un problème de voyageur de commerce. Lorsque les temps de préparation sont regroupés en familles, les opérations de la même famille peuvent être regroupées et le TSP peut être résolu de manière équivalente en considérant une seule opération par famille $f \in \mathcal{F}_\Omega = \cup_{O_{ij} \in \Omega} \{f_{ij}\}$. Si $|\mathcal{F}_\Omega|$ est suffisament plus petit que $|\Omega|$, les temps de résolution du TSP peuvent devenir raisonnables. Dans la suite on notera indifféremment s_Ω et $s_{\mathcal{F}_\Omega}$. Plus précisément les algorithmes de propagation nécessitent le calcul pour chaque sous-ensemble de familles $\mathcal{F}' \subseteq \mathcal{F}_{\mathcal{O}_k}$ et pour chaque famille $f \in \mathcal{F}'$, des valeurs notées $s_{f \to \mathcal{F}'}$ et $s_{\mathcal{F}' \to f}$. Ces valeurs correspondent aux temps de préparation minimaux pour ordonnancer un ensemble d'opérations de familles $\mathcal{F}'$ si f est la famille de, respectivement, la première et la dernière opération de l'ensemble. Notons qu'on a $s_{\mathcal{F}'} = \min_{f \in \mathcal{F}'} s_{f \to \mathcal{F}'} = \min_{f \in \mathcal{F}'} s_{\mathcal{F}' \to f}$.

Brucker et Thiele [10] utilisent des bornes inférieures de $s_{f \to \mathcal{F}'}$ et $s_{\mathcal{F}' \to f}$ et la valeur optimale obtenue par énumération avec une complexité en temps de $O(|\mathcal{F}'|!)$. Dans ce papier nous reprenons la méthode de Vilím et Barták [29] qui proposent de précalculer $s_{f \to \mathcal{F}'}$ et $s_{\mathcal{F}' \to f}$ une fois pour toutes avant de lancer la recherche. Chaque ensemble $\mathcal{F}'$ est codé par représentation binaire de la permutation correspondante de telle sorte que $s_{f \to \mathcal{F}'}$ et $s_{\mathcal{F}' \to f}$ peuvent être obtenus en $O(1)$ une fois que le précalcul est effectué. Les valeurs $s_{f \to \mathcal{F}'}$ pour chaque ensemble $\mathcal{F}' \subseteq \mathcal{F}_{\mathcal{O}_k}$ et chaque famille $f \in \mathcal{F}'$ sont calculées en $O(|\mathcal{F}_{\mathcal{O}_k}|^2 2^{|\mathcal{F}_{\mathcal{O}_k}|})$ par la

récurrence suivante :

$$s_{f\to\{f\}} = 0 \qquad \forall f \subseteq \mathcal{F}_{\mathcal{O}_k} \tag{13}$$

$$s_{f\to\mathcal{F}'\cup\{f\}} = \min_{g\in\mathcal{F}'}\{s_{fg} + s_{g\to\mathcal{F}'}\}$$
$$\forall \mathcal{F}' \subseteq \mathcal{F}_{\mathcal{O}_k}, \forall f \in \mathcal{F}_{\mathcal{O}_k} \setminus \mathcal{F}' \tag{14}$$

Les $s_{\mathcal{F}'\to f}$ sont calculées de manière symétrique.

Pour un ensemble d'opération Ω nous introduisons également la notation suivante :

$$r_\Omega = \min_{O_{ij}\in\Omega} r_{ij} \tag{15}$$

$$d_\Omega = \max_{O_{ij}\in\Omega} d_{ij} \tag{16}$$

5.2.2 Propagation de contraintes

La fonction PROPAGATE$(\mathcal{R},\mathcal{D},\mathcal{E})$ effectue des ajustement sur les dates de début au plus tôt $\mathcal{R}$ et les dates de fin au plus tard $\mathcal{D}$ avec la sélection courante $\mathcal{E}$, enrichit cette dernière et retourne un échec si une incohérence est détectée. Elle appelle successivement les algorithmes de propagation UPDATEEARLIESTSTART, UPDATELATESTCOMPLETION, IMMEDIATESELECTION et EDGEFINDING, chacune effectuant des déductions spécifiques. La séquence d'appels s'arrête lorsqu'un échec survient ou lorsque plus aucune déduction n'est effectuée. Les algorithmes UPDATEEARLIESTSTART et UPDATELATESTCOMPLETION mettent à jour respectivement $\mathcal{R}$ et $\mathcal{D}$ pour toutes les opérations $O_{ij} \in \mathcal{O}$, en tenant compte des contraintes de précédences initiales et de celles données par la sélection courante $\mathcal{E}$. Les dates de début au plus tôt sont mises à jour avec les règles suivantes :

$$r_{ij} \geq s_{0im_{ij}} \qquad \forall O_{ij} \tag{17}$$

$$r_{ij} \geq r_{i(j-1)} + p_{i(j-1)} \qquad \forall O_{ij}, j{>}1 \tag{18}$$

$$r_{ij} \geq r_\Omega + p_\Omega + s_{\mathcal{F}_{\Omega\cup\{f_{ij}\}}\to f_{ij}}$$
$$\forall O_{ij}, \forall \Omega \subseteq \{O_{xy} | (O_{xy}, O_{ij}) \in \mathcal{E}\} \tag{19}$$

Les dates de fin au plus tard sont mises à jour avec les règles suivantes :

$$d_{ij} \leq T \qquad \forall O_{ij} \tag{20}$$

$$d_{ij} \leq d_{i(j+1)} - p_{i(j+1)} \qquad \forall O_{ij}, j{<}m \tag{21}$$

$$d_{ij} \leq d_\Omega - p_\Omega - s_{f_{ij}\to\mathcal{F}_{\Omega\cup\{f_{ij}\}}}$$
$$\forall O_{ij} \in \mathcal{O}, \forall \Omega \subseteq \{O_{xy} | (O_{xy}, O_{ij}) \in \mathcal{E}\} \tag{22}$$

L'algorithme 2 de complexité en temps $O(n^2 m \log n)$ décrit UPDATEEARLIESTSTART. Brucker et Thiele [10] prouvent que cet algorithme donne le plus grand ajustement compte tenu de la règle (19), si l'inégalité triangulaire est vérifiée. Nous obtenons une complexité moindre que celle obtenue dans [10] grâce aux précalculs des temps de préparation. Notons que l'algorithme détecte aussi les circuits dans $G(\mathcal{E}) = (X, U \cup \mathcal{E})$ (étapes 12-13) de par son application itérative au sein de la fonction PROPAGATE. L'algorithme UPDATELATESTCOMPLETION est symétrique.

Algorithm 2 UPDATEEARLIESTSTART$(\mathcal{R},\mathcal{D},\mathcal{E})$

```
1:  for O_ij ∈ O do
2:    r_ij ← max(r_ij, s_0im_ij)
3:    if j > 1 then
4:      r_ij ← max(r_ij, r_i(j−1) + p_i(j−1))
5:    end if
6:    Ω ← {O_xy | (O_xy, O_ij) ∈ E}
7:    Trier Ω dans l'ordre croissant des r_ij
8:    for O_xy ∈ Ω do
9:      r_ij ← max(r_ij, p_Ω + s_{F_Ω∪{f_ij}→f_ij})
10:     Ω ← Ω \ {O_xy}
11:   end for
12:   if r_ij + p_ij > d_ij then
13:     return échec
14:   end if
15: end for
16: return succès
```

L'algorithme IMMEDIATESELECTION est un algorithme de propagation des contraintes de ressources disjonctives dont l'objectif est de détecter de nouvelles contraintes de précédences entre deux opérations affectées à la même machine. Il est basé sur les règles : $\forall M_k \in \mathcal{M}, \forall O_{ij}, O_{x,y} \in \mathcal{O}_k, O_{ij} \neq O_{x,y}$

$$r_{ij}{+}p_{ij}{+}p_{xy}{+}s_{ixk} > d_{xy} \implies (O_{xy}, O_{ij}) \in \mathcal{E} \tag{23}$$

$$(O_{xy}, O_{ij}) \in \mathcal{E} \implies r_{ij} \geq r_{xy} + p_{xy} + s_{xik}$$
$$\text{et } d_{xy} \leq d_{ij} - p_{ij} - s_{xik} \tag{24}$$

Il est implémenté de manière triviale en $\mathcal{O}(n^2 m)$ et retourne un échec dès qu'une mise à jour pour une opération O_{ij} donne $r_{ij} + p_{ij} > d_{ij}$.

L'algorithme EDGEFINDING effectue des déductions supplémentaires en considérant une operation O_{ij} affectée à la machine M_k et un ensemble $\Omega \subseteq \mathcal{O}_k \setminus \{O_{ij}\}$. Il est basé plus précisément sur les règles primales et duales suivantes. La règle primale s'énonce : $\forall M_k \in \mathcal{M}, \forall O_{ij} \in \mathcal{O}_k, \forall \Omega \subseteq \mathcal{O}_k \setminus \{O_{ij}\}$,

$$r_{\Omega\cup\{O_{ij}\}} + p_{\Omega\cup\{O_{ij}\}} + s_{\mathcal{F}_{\Omega\cup\{O_{ij}\}}} > d_\Omega \implies$$
$$\forall O_{xy} \in \Omega, (O_{xy}, O_{ij}) \in \mathcal{E} \tag{25}$$

La règle duale s'énonce : $\forall M_k \in \mathcal{M}$, $\forall O_{ij} \in \mathcal{O}_k$, $\forall \Omega \subseteq \mathcal{O}_k \setminus \{O_{ij}\}$

$$d_{\Omega\cup\{O_{ij}\}} - p_{ij} - s_{\mathcal{F}_{\Omega\cup\{O_{ij}\}}} < r_\Omega \implies$$
$$\forall O_{xy} \in \Omega, (O_{ij}, O_{xy}) \in \mathcal{E} \tag{26}$$

Grossièrement, les règles primales et duales détectent qu'une opération O_{ij} n'est pas insérable dans l'ensemble Ω. La règle primale détecte en outre que O_{ij} doit être ordonnancée après toutes les opérations de Ω sur M_k alors que la règle duale détecte que O_{ij} doit être ordonnancée avant toutes les opérations de Ω. Les règles (19) et (22) peuvent être combinées avec les règles duales et primales pour effectuer les ajustements des fenêtres de temps apportés par les nouveaux arcs de $\mathcal{E}$. Elles donnent également la régle de détection d'incohérence suivante : $\forall M_k \in \mathcal{M}$, $\forall \Omega \subseteq \mathcal{O}_k$

$$r_\Omega + p_\Omega + s_{\mathcal{F}_\Omega} > d_\Omega \tag{27}$$

Vilím et Barták [29] proposent un algorithme en $O(|\mathcal{F}|n^2)$ pour effectuer tous les ajustements liés au edge-finding et un autre algorithme en $O(n \log n)$ pour effectuer toutes les détections d'incohérence. Dans ce papier nous proposons une extension du edge-finding de Nuijten [23] qui effectue les détections d'incohérence et les ajustement simultanément. Notre extension a une complexité de $O(|\mathcal{F}|n^2)$ et est détaillée dans l'algorithme 3 pour la partie primale, la partie duale étant symétrique.

Nous rappelons les principes de l'algorithme de Nuijten et expliquons les changements que nous avons opérés pour tenir compte des temps de préparation, dans l'esprit de la méthode de Vilím et Barták [29].

Théorème 1 *L'algorithme 3 effectue toutes les vérifications de cohérence (27).*

Soit $l(\Omega)$ le membre de gauche de l'inégalité de la règle (27). Soit σ la permutation donnant l'ordre de parcours de opérations dans l'ordre des r_{xy} croissants. En considérant un indice de boucle q variant de n à 1, la boucle 6-17 génère tous les ensembles Ω_q tels que $\Omega_q = \{O_{ab} \in \mathcal{O} | \sigma_{ab} \geq q \wedge d_{ab} \leq d_{ij}\}$ et calcule $C = \max_{q=1}^{n} l(\Omega_q)$. Pour l'opération O_{ij} considérée, Soit les ensembles $\Omega(O_{ab}, O_{ij}) = \{O_{vw} \in \mathcal{O}_k | d_{vw} \leq d_{ij}, r_{vw} \geq r_{ab}\}$, $\forall O_{ab} \in \mathcal{O}_k$. Ces ensembles (appelés également intervalles de tâches) sont tous inclus dans $\cup_{q=1}^{n} \Omega_q$ et sont, en raison du respect de l'inégalité triangulaire, dominants pour la règle (27) avec tout ensemble Ω tel que $d_\Omega = d_{ij}$. ∎

Théorème 2 *L'algorithme 3 effectue tous les ajustements à partir des règles (25) et (19).*

La règle (25) concernant un ensemble Ω tel que $d_\Omega = d_{ij}$ et une opération O_{xy} ne peut être appliquée sans déclencher d'échec (alors détecté par la règle (27)) que si $d_{xy} > d_{ij}$. Les intervalles de tâches $\Omega(O_{ab}, O_{ij})$, $\forall O_{ab} \in \mathcal{O}_k$ et donc

Algorithm 3 primalEdgeFinding$(k, \mathcal{R}, \mathcal{D})$

```
1: for f ∈ F_{O_k} do
2:   for O_ij ∈ O_k (r_ij croissants) do
3:     P ← 0
4:     C ← 0
5:     Ω ← ∅
6:     for O_xy ∈ O_k (r_xy décroissants) do
7:       if d_xy ≤ d_ij then
8:         P ← P + p_xy
9:         Ω ← Ω ∪ {O_xy}
10:        C ← max(C, r_xy + P + s_{F_Ω})
11:        D ← max(D, r_xy + P + s_{F_Ω ∪ {f} → f})
12:        if C > d_ij then
13:          return échec
14:        end if
15:      end if
16:      c_xy ← D
17:    end for
18:    H ← 0
19:    for O_xy ∈ O_k (r_xy croissants) do
20:      if d_xy ≤ d_ij then
21:        H ← max(H, r_xy + P + s_{F_Ω ∪ {f}})
22:        P ← P − p_xy
23:        Ω ← Ω \ {O_xy}
24:      else if f_xy = f then
25:        if r_xy + P + p_xy + s_{F_Ω ∪ {f}} > d_ij then
26:          r_xy ← max(r_xy, c_xy)
27:        end if
28:        if H + p_xy > d_ij then
29:          r_xy ← max(r_xy, D)
30:        end if
31:      end if
32:    end for
33:  end for
34: end for
35: return succès
```

les ensembles Ω_q sont également dominants. Soit $L(\Omega, O_{xy})$ le membre de gauche de l'inégalité de la règle (25) avec l'ensemble Ω et l'opération O_{xy}. Soit $A(\Omega)$ la valeur d'ajustement donnée par l'ensemble Ω dans la règle (19). Soit une itération q telle que $d_{xy} > d_{ij}$. L'ensemble considéré est Ω_q qui vérifie $r_{\Omega_q} \geq r_{xy}$ et donc $r_{\Omega_q \cup \{O_{xy}\}} = r_{xy}$. L'étape 25 effectue le test de la règle (25) pour Ω_q et O_{xy}, ce qui domine tout autre ensemble Ω_ρ avec $\rho > q$. De plus, on a $c_{xy} = \max_{\rho=q}^{n} A(\Omega_q)$ ce qui constitue l'ajustement maximal de r_{xy} avec la règle (19) et l'ensemble Ω_q si $f_{xy} = f$. Il reste à examiner les ensembles Ω_ρ avec $\rho < q$ pour lesquels $r_{\Omega_\rho \cup \{O_{xy}\}} = r_{\Omega_\rho}$. H, utilisé à l'étape 28 pour tester la règle (25) est bien égal à $\max_{\rho=1}^{q-1} L(\Omega_\rho, O_{xy}) - p_{xy}$ si $f_{xy} = f$. Enfin si O_{xy} doit être après un des ensembles Ω_ρ avec $\rho < q$ alors O_{xy} doit être après

l'ensemble Ω_1 car $d_{xy} > d_{\Omega_1}$. L'ajustement maximal est donc $D = A(\Omega_1)$. ∎

Au nœud racine de l'arbre de recherche, nous appliquons un algorithme de propagation de contraintes, appelé SHAVING[5]. Pour chaque opération O_{ij}, la contrainte $t_{ij} = r_{ij}$ est posée et la fonction PROPAGATE est appelée. Si une incohérence est détectée, r_{ij} peut être remplacé par $r_{ij} + 1$ et le processus est itéré tant que des ajustement sont effectués et jusqu'à la détection d'une incohérence globale (la fenêtre devient alors vide), en considérant successivement toutes les opérations. La procédure symétrique est appliquée avec les dates de fin au plus tard. Nous renvoyons à [22, 28] pour des implémentations plus efficaces et des variantes de cette technique.

5.3 Résolution des relaxations de TSPTW

Chaque F-TSPTW peut être défini comme un problème de plus court chemin élémentaire avec contraintes de ressources (ESPPRC[6]) [17]. Le ESPPRC considère un réseau constitué d'un nœud origine, d'un nœud destination, d'arcs valués par un coût ρ_{uv} et de Q resources consommables. Passer par un arc (u, v) consomme une quantité l^q_{uv} de chaque ressource q, avec $1 \leq q \leq Q$. Les valeurs l^q_{uv} doivent vérifier l'inégalité triangulaire pour chaque ressource. La consommation de chaque ressource q à un nœud u du réseau est limitée par un intervalle $[a^q_u, b^q_u]$. Si un chemin élémentaire utilise l'arc (u, v), la consommation W^q_v de la ressource q entre l'origine et v doit vérifier $W^q_v \geq \max(a^q_v, W^q_u + l^q_{uv})$ et $W^q_v \leq b^q_v$. L'objectif est de trouver un chemin élémentaire de coût minimal de l'origine à la destination satisfaisant les contraintes de ressources ainsi définies.

Pour définir un problème de F-TSPTW comme un ESPPRC considérons la machine k et l'ensemble des opérations $\mathcal{O}_k$. Un nœud est introduit pour chaque opération $\mathcal{O}_k$. Un arc (u, v) est introduit pour chaque paire d'opérations $(u = O_{ij}, v = O_{xy})$ de coût $\rho_{uv} = -M$ où M est une constante arbitraire. Une seule ressource ($Q = 1$) est définie, avec $l^1_{uv} = s_{ixk} + p_{xy}$. L'intervalle de consommation d'un nœud $u = O_{ij}$ est $[a^1_{uv}, b^1_{uv}] = [r_{ij} + p_{ij}, d_{ij}]$. Deux nœuds additionnels sont définis comme origine et destination. Un arc de consommation s_{0ik} est défini entre l'origine et chaque nœud $u = O_{ij}$, et un arc de consommation 0 est défini entre chaque nœud et la destination. L'inégalité triangulaire des consommations est bien respectée puisqu'on a $s_{ijk} \leq s_{izk} + s_{zjk} \implies s_{ijk} + p_j \leq s_{izk} + p_z + s_{zjk} + p_j$, pour tous $i \in [0, n], j, z \in [1, n]$ distincts. Les coûts sont définis de telle sorte que la solution optimale du ESPPRC visite tous les nœuds si elle existe. Résoudre le ESPPRC revient ainsi à résoudre le F-TSPTW.

Nous résolvons le problème par l'algorithme de programmation dynamique proposé par Feillet *et al* [17]. Cet algorithme suit le principe de l'algorithme classique de Bellman. Une étiquette est ainsi associée à chaque chemin partiel possible et l'algorithme étend ces étiquettes en vérifiant les contraintes de ressources jusqu'à ce qu'une solution réalisable soit obtenue où qu'il n'existe plus d'extension réalisable. Des ressources fictives indiquant l'accessibilité des nœuds sont introduites pour préserver l'élémentarité du chemin. Des règles de dominance sont utilisées pour comparer des étiquettes et en supprimer certaines. L'algorithme est adapté pour éviter l'extension d'une étiquette qui ne pourrait atteindre un nœud non visité.

Pour accélérer la résolution, nous utilisons une borne inférieure pour chaque étiquette générée. Soit t le temps associé à une étiquette correspondant à un chemin partiel de l'origine à un nœud $v = O_{ij}$, i.e. $t = W^q_v$ pour cette étiquette. Soit $\overline{\mathcal{O}}_k \subset \mathcal{O}_k$ l'ensemble des opérations de la machine k non encore visitées par ce chemin partiel. Le test d'incohérence (27) peut être utilisé. Il n'est ainsi pas utile d'étendre l'étiquette concernée lorsque $\exists \Omega \subseteq \overline{\mathcal{O}}_k$, $t + p_{ij} + p_\Omega + s_{f_{ij} \to \Omega \cup \{f_{ij}\}} > d_\Omega$. Comme le nombre d'étiquettes générées peut être très grand, nous conservons une complexité linéaire pour l'évaluation en considérant uniquement $\Omega = \overline{\mathcal{O}}_k$.

5.4 Règles de branchement et de dominance

Nous proposons un schéma de branchement chronologique basé sur la sélection partielle courante. Soit $E_{ij} \subseteq E$ l'ensemble des arêtes connectées à O_{ij} pour le nœud courant de l'arbre de recherche $\nu = (\mathcal{R}, \mathcal{D}, \mathcal{E})$. Si la sélection est complète, on a $E_{ij} = \emptyset$, pour toute opération O_{ij}. Soit $O_{i^*j^*}$ une opération telle que $E_{i^*j^*} \neq \emptyset$ et de date de début au plus tot $r_{i^*j^*}$ minimale. Soit $\mathcal{C}$ l'ensemble des opérations potentiellement en conflit avec $O_{i^*j^*}$, c'est-à-dire affectées à la même machine et vérifiant $r_{ij} < r_{i^*j^*} + p_{i^*j^*} + s_{i^*im_{ij}}$. La règle de branchement que nous proposons génère $|\mathcal{C}|$ nœuds $\{\nu_{ij}\}_{O_{ij} \in \mathcal{C}}$ à partir du nœud courant ν tels que $\nu_{ij} = (\mathcal{R}, \mathcal{D}, \mathcal{E} \cup \{(O_{ij}, O_{xy})\}_{O_{xy} \in \mathcal{C} \setminus O_{ij}})$.

Il faut souligner qu'une telle règle de branchement peut mener à des ordonnancements non actifs. Certaines opérations de $\mathcal{C}$ peuvent voir en ef-

[5] Il peut être vu comme un algorithme de maintien de la cohérence de singletons qui revient à tester si le problème est arc-cohérent après l'instantiation d'une variable à une valeur [7]

[6] elementary shortest path problem with resource constraints

fet leur date de début au plus tôt augmenter dans la descendance du nœud courant, jusqu'à éventuellement pouvoir être insérée avant l'opération sélectionnée. Par définition, ce ne peut être le cas de l'opération $O_{i^*j^*}$. Toutefois, pour toute opération $O_{ij} \neq O_{i^*j^*}$ de l'ensemble $\mathcal{C}$ un nœud descendant de ν_{ij} tel que r_{ij} devient supérieur ou égal à $r_{i^*j^*} + p_{i^*j^*} + s_{i^*im_{ij}}$ peut être supprimé. Ceci est simplement réalisé en posant $d_{ij} = r_{i^*j^*} + p_{i^*j^*} + s_{i^*im_{ij}} + p_{ij} - 1$.

6 Résultats expérimentaux

Nous présentons les résultats de notre approche sur les instances BT. Ces instances ont la particularité d'être résolues facilement en ignorant les temps de préparation. Chaque instance est caractérisée par un triplet (#machines,#travaux,#familles). Les 5 premières instances sont de type $(5, 10, 5)$. Les 5 suivantes sont de type $(5, 15, 5)$. Les 5 dernières sont de type $(5, 20, 5)$. Le nombre d'itérations de l'algorithme de programmation dynamique pour la résolution du TSPTW est limité à 3000. Une limite de 20 millions de nœuds est fixée pour la résolution de chaque CSP. L'heuristique appliquée à chaque nœud et décrite en 5.1 peut trouver des solutions de valeur supérieures à T. La meilleure solution trouvée est conservée pour chaque résolution de CSP. Dès qu'elle est mise à jour, une phase d'intensification est appliquée en lançant 50000 itérations de l'algorithme glouton en faisant varier aléatoirement la règle de priorité autour de la valeur ayant mené à l'amélioration. La durée totale optimale est cherchée linéairement à partir de la borne inférieure triviale. Nous avons effectué nos tests en C++ sur une architecture AMD64 sous Linux. La table 1 compare nos résultats avec les meilleurs résultats précédemment obtenus par Brucker et Thiele [10] pour les bornes inférieures et Balas *et al* [5] pour les bornes supérieures. La colonne #nodes donne pour chaque instance le nombre cumulé de nœuds sur l'ensemble des CSP résolus. Le temps de calcul est indiqué en secondes.

Notre méthode parvient à résoudre dans des temps raisonables les 5 problèmes de petite taille déjà résolus par Brucker et Thiele [10]. Nous parvenons à fermer 4 sur 5 des instances de taille moyenne, trouvant ainsi des solutions meilleures que l'heuristique performante de Balas *et al* [5] sur 3 problèmes. Nous améliorons considérablement les bornes inférieures de Brucker et Thiele [10] sur les autres instances, au prix de temps de calculs prohibitifs.

Tab. 1 – Résultats sur les instances BT

P	LB/UB	#nodes	CPU	LB[10]/UB[5]
1	798/798	1	56.7	798/798
2	784/784	9498	242.3	784/784
3	749/749	181090	699.3	749/749
4	730/730	3594	251.6	730/740
5	691/691	2918	58.2	691/693
6	1009/1009	256520	1797.6	986/1018
7	970/970	71106	781.8	940/1003
8	963/963	33491952	349923	913/975
9	1051/1061	20105686	169582	1001/1060
10	1018/1018	227	35.1	1008/1018
11	1395/1617	57981046	916833	1322/1470
12	1242/1424	69387629	914086	1139/1305
13	1342/1457	58651163	895059	1250/1439
14	1432/1499	19999854	306899	1402/1485
15	1406/1671	52337255	792196	1307/1527

7 Conclusion

Notre méthode, basée sur une utilisation conjointe de la propagation des contraintes de ressources adaptée aux temps de préparation et de la résolution exacte de relaxations à des problèmes de TSPTW a considérablement amélioré les meilleurs résultats connus sur les instances réputées très difficiles de Brucker et Thiele [10]. Pour passer à une échelle supplémentaire des problèmes résolus, la clé pourrait consister en une meilleure résolution des problèmes de TSPTW. Le nombre limite d'itérations de programmation dynamique est en effet souvent atteint pour les problèmes de grande taille et de taille moyenne.

Références

[1] A. Allahverdi, J. N. D. Gupta, and T. Aldowaisan. A review of scheduling research involving setup considerations. *Omega*, 27 :219–239, 1999.

[2] C. Artigues, S. Belmokhtar, and D. Feillet. A new exact solution algorithm for the job shop problem with sequence-dependent setup times. In *CPAIOR 2004*, pages 37–49.

[3] C. Artigues, F. Buscaylet, and D. Feillet. Lower and upper bound for the job shop scheduling problem with sequence-dependent setup times. In *MISTA 2005*, New York.

[4] C. Artigues, P. Lopez, and P.D. Ayache. Schedule generation schemes and priority rules for the job-shop problem with sequence-dependent setup times : Dominance properties

and computational analysis. *Annals of Operations Research*, 138(1) :21–52, 2005.

[5] E. Balas, N. Simonetti, and A. Vazacopoulos. Job shop scheduling with setup-times, deadlines and precedence constraints. In *MISTA 2005*, New York, USA.

[6] P. Baptiste, C. Le Pape, and W. Nuijten. *Constraint-based scheduling*. Springer, 2001.

[7] C. Bessière and R. Debruyne. Algorithmes optimaux et sous-optimaux de singlenton consistance d'arc. In *JFPC 2005*, Lens, France.

[8] J. Blazewicz, W. Domschke, and E. Pesch. The job shop scheduling problem : Conventional and new solution techniques. *European Journal of Operational Research*, 93(1) :1–33, 1996.

[9] P. Brucker, P. Jurisch, and A. Krämer. The job-shop problem and immediate selection. *Annals of Operations Research*, 50 :73–114, 1994.

[10] P. Brucker and O. Thiele. A branch and bound method for the general-shop problem with sequence-dependent setup times. *Operations Research Spektrum*, 18 :145–161, 1996.

[11] F. Buscaylet and C. Artigues. A fast tabu search method for the job-shop problem with sequence-dependent setup times. In *MIC'2003*.

[12] M. A. B. Candido, S.K. Khator, and R. M. Barcias. A genetic algorithm based procedure for more realistic job shop scheduling problems. *International Journal of Production Research*, 36(12) :3437–3457, 1998.

[13] J. Carlier. The one machine sequencing problem. *European Journal of Operational Research*, 11 :42–47, 1982.

[14] J. Carlier and E. Pinson. An algorithm for solving the job-shop problem. *Management Science*, 35 :164–176, 1989.

[15] I.-C. Choi and O. Korkmaz. Job shop scheduling with separable sequence-dependent setups. *Annals of Operations Research*, 70 :155–170, 1997.

[16] I.-N. Choi and D.-S. Choi. A local search algorithm for jobshop scheduling problems with alternative operations and sequence-dependent setups. *Computers and Industrial Engineering*, 42 :43–58, 2002.

[17] D. Feillet, P. Dejax, M. Gendreau, and C. Gueguen. An exact algorithm for the elementary shortest path problem with resource constraints : Application to some vehicle routing problems. *Networks*, 44(3) :216–229, 2004.

[18] F. Focacci, P. Laborie, and W. Nuijten. Solving scheduling problems with setup times and alternative resources. In *AIPS 2000*, pages 92–101.

[19] F. Focacci, A. Lodi, and M. Milano. A hybrid exact algorithm for the TSPTW. *INFORMS Journal on Computing*, 14 :403–417, 2002.

[20] S.C. Kim and P.M. Bobrowski. Impact of sequence-dependent setup time on job shop scheduling performance. *International Journal of Production Research*, 32(7) :1503–1520, 1994.

[21] R. Kolisch. serial and parallel resource-constrained project scheduling methods revisited : theory and computation. *European Journal of Operational Research*, 90 :320–333, 1996.

[22] P. Martin and D.B. Shmoys. A new approach to computing optimal schedules for the job-shop scheduling problem. In *IPCO 1996*, pages 389–403.

[23] W. P. M. Nuijten. *Time and Resource Constrained Scheduling : A Constraint Satisfaction Approach*. PhD thesis, Eindhoven University of Technology, 1994.

[24] I.M. Ovacik and R. Uzsoy. A shifting bottleneck algorithm for scheduling semiconductor testing operations. *Journal of Electronics Manufacturing*, 2 :119–134, 1992.

[25] I.M. Ovacik and R. Uzsoy. Exploiting shop floor status information to schedule complex job shop. *Journal of manufacturing systems*, 13 :73–84, 1994.

[26] B. Roy and B. Sussman. Les problèmes d'ordonnancement avec contraintes disjonctives. Technical Report Note DS no 9bis, SEMA, Paris, 1964.

[27] X. Sun and J.S. Noble. A modified shifting bottleneck approach to job shop scheduling with sequence dependent setups. *Journal of Manufacturing Systems*, 18(6) :416–430, 1999.

[28] P. Torres and P. Lopez. Overview and possible extensions of shaving techniques for job-shop problems. In *CPAIOR 2000*, pages 181–186, Paderborn, Germany.

[29] P. Vilím and R. Barták. Filtering algorithms for batch processing with sequence dependent setup times. In *AIPS 2002*, pages 312–320.

[30] C. Zhou and P.G. Egbelu. Scheduling in manufacturing shop with sequence-dependent setups. *Robotics and Computer Integrated Manufacturing*, 5 :73–81, 1989.