Chapitre 7

Heuristiques d'arbitrage dans un atelier

Ce chapitre décrit plusieurs variantes de la méthode de résolution des problèmes d'atelier présentée au chapitre 6. La majorité des approches complètes en ordonnancement sous contraintes sont basées sur des algorithmes de filtrage et des heuristiques de sélection dédiés à ces problèmes. En effet, une croyance répandue est que ces problèmes sont trop difficiles pour être résolus autrement. Nous proposons une nouvelle heuristique inspirée de celle du degré pondéré combinée à un modèle du graphe disjonctif dont la propagation reste naïve. Nous montrons que les résultats de cette approche surpassent souvent ceux des meilleures approches en ordonnancement sous contraintes pour l'open-shop et le job-shop.

Sommaire

Nous présentons nos travaux [5] sur les heuristiques d'arbitrage pour la résolution des problèmes d'atelier par l'algorithme présenté au chapitre 6. Nous examinerons la variation des performances des heuristiques d'arbitrage (bloc 9 de l'algorithme en figure 6.2) en fonction de différents modèles en ordonnancement sous contraintes (bloc 3). Des expérimentations préliminaires, non détaillées ici, ont confirmé les résultats de l'analyse de sensibilité présentée en section 6.3.2 page 60. De ce fait, nous ne remettrons pas en cause les autres composantes de l'algorithme.

En constatant l'importance des redémarrages, nous avons souhaité exploiter les informations issues des différentes exécutions autrement que par l'enregistrement des *nogoods*. L'heuristique du degré pondéré (voir section 2.3) a attiré notre attention, car il s'agit d'une alternative à la randomisation de l'arbitrage (bloc 7). En effet, il suffit de conserver les poids des contraintes d'une exécution à l'autre pour espérer résoudre plus facilement le problème en identifiant les parties difficiles (*backdoors*) qui correspondent aux variables dont le degré pondéré est élevé. Par conséquent, la stratégie de branchement doit maintenant instancier une variable à chaque point de choix et non ajouter une contrainte. Pour ce faire, nous délaisserons la contrainte globale modélisant le graphe de précédence au profit d'un ensemble de contraintes de disjonction réifiées modélisant le graphe disjonctif. Ces changements nous amèneront à distinguer deux modèles en fonction de la présence de contraintes disjonctives.

La section 7.1 décrit les modèles *Light* et *Heavy* puis les nouvelles stratégies de branchement. Nous uti-

lisons deux implémentations du modèle *Light* basées sur les solveurs `choco` et `mistral` (voir section 2.5). La section 7.2 présente les différences mineures entre les implémentations `choco` et `mistral`. Finalement, La section 7.3 démontre la pertinence de cette approche sur les problèmes d'open-shop et de job-shop. Nous croyons que la combinaison de l'heuristique inspirée du degré pondéré et du modèle *Light* est très efficace pour identifier les paires de tâches critiques.

7.1 Modèles en ordonnancement sous contraintes

Nous présentons dans cette section deux modèles logiquement équivalents, mais dont les niveaux d'inférence sont différents. Ces modèles sont basés sur la modélisation du graphe disjonctif par des contraintes de disjonction réifiées présentée en section 7.1.1 qui remplace la contrainte globale décrite en section 6.1.2. Ils sont destinés à être combinés avec les stratégies de branchement présentées en section 7.1.2. Le modèle *Heavy* pose des contraintes globales disjonctives définies en section 6.1 contrairement au second modèle, dénommé *Light*. Finalement, la section 7.1.3 discute de ces deux modèles et de la modélisation des contraintes temporelles dans le chapitre 6 et dans celui-ci.

7.1.1 Contraintes temporelles

Ainsi, nous passons d'une représentation du graphe de précédences par une contrainte globale à une représentation du graphe disjonctif par des contraintes de disjonction entre deux tâches définies ci-dessous. Nous introduisons une variable booléenne $b_{ij \preceq kl}$ qui représente l'ordre d'enchaînement pour chaque paire de tâches T_{ij} et T_{kl} partageant une ressource (lot ou machine). La variable booléenne $b_{ij \preceq kl}$ réifie la contrainte de disjonction présentée en section 3.2 en prenant la valeur 1 lorsque T_{ij} précède T_{kl} et la valeur 0 lorsque T_{kl} précède T_{ij} :

$$T_{ij} \simeq T_{kl} \quad \Leftrightarrow \quad \begin{cases} T_{ij} \preceq T_{kl} & \text{si } b_{ij \preceq kl} = 1 \\ T_{kl} \preceq T_{ij} & \text{si } b_{ij \preceq kl} = 0 \end{cases} \tag{7.1}$$

Tant que la disjonction n'est pas arbitrée, la contrainte applique les règles de filtrage : précédence interdite - précédence obligatoire (voir section 3.2). Après son arbitrage par l'instanciation de la variable booléenne $b_{ij \preceq kl}$, le filtrage applique une règle assurant la consistance de bornes de la contrainte de précédence. Pour n lots et m machines, la représentation du graphe disjonctif nécessite $\frac{nm(m+n-2)}{2}$ disjonctions réifiées pour l'open-shop et $\frac{nm(m+n-2)}{4}$ pour le flow-shop ou le job-shop. Dans le cas du flow-shop et du job-shop, l'ordre des opérations dans un lot est imposé par l'ajout des contraintes de précédences (arithmétiques) entre les paires d'opérations consécutives.

7.1.2 Stratégie de branchement

Un arbitrage complet du graphe disjonctif (impliquant l'existence d'une solution) est maintenant obtenu en instanciant toutes les variables booléennes réifiant les disjonctions entre les paires de tâches partageant une ressource (lot ou machine).

Sauf mention contraire, on sélectionne aléatoirement (distribution uniforme) une des meilleures variables en cas d'égalité.

profile instancie maintenant une variable booléenne à chaque point de choix au lieu d'ajouter une contrainte de précédence (voir section 6.1.4).

top10%-profile sélectionne aléatoirement (distribution uniforme) une disjonction appartenant au 10 % des meilleures paires (ressource/instant) des profils.

slack sélectionne une variable $b_{ij \preceq kl}$ dont le « domaine » est de taille minimale. La taille du domaine est mesurée comme la somme des tailles des fenêtres de temps dans la disjonction $T_{ij} \simeq T_{kl}$: $(lst_{ij} - est_{ij} + 1) + (lst_{kl} - est_{kl} + 1)$.

wdeg sélectionne la variable dont le poids pondéré de ses contraintes est maximal. Le poids d'une contrainte est initialisé à la valeur de son degré. Il est incrémenté à chaque fois qu'elle déclenche une contradiction pendant la propagation. Dans notre modèle, toutes les variables de décision $b_{ij \preceq kl}$

sont booléennes et *wdeg* est par conséquent équivalente à *dom/wdeg*. Le degré des variables $b_{ij \preceq kl}$ est unitaire.

slack/wdeg sélectionne une variable $b_{ij \preceq kl}$ dont le quotient du « domaine » sur le degré pondéré est minimal.

top3-slack, top3-wdeg, top3-dom/wdeg sélectionnent aléatoirement (distribution uniforme) une des trois meilleures variables quand il n'y a pas d'égalité. En cas d'égalité, elles sélectionnent aléatoirement une des meilleures variables

L'heuristique *wdeg* ne devient discriminante que lorsqu'une première contradiction a été levée. Ainsi, l'heuristique *slack/wdeg* comporte un élément de discrimination basé sur les fenêtres de temps jusqu'à l'entrée en lice des degrés pondérés.

Nous considérons trois heuristiques de sélection de valeur (ou arbitrage) :

minVal sélectionne la plus petite valeur du domaine.

centroid sélectionne la valeur associée à la précédence qui préserve l'ordre des centroids (voir section 6.1.4). En cas d'égalité, on sélectionne aléatoirement (distribution uniforme) une précédence.

jobOrdering est une heuristique statique pour le job-shop et le flow-shop où une paire d'opérations est ordonnée sur une machine selon leur position relative dans leur lot respectif. Par exemple, si la tâche T_{ij} est la seconde de son lot et T_{ik} est la quatrième, l'heuristique sélectionne la valeur 1 pour $b_{ij \preceq ik}$, c'est-à-dire T_{ij} précède T_{ik}.

Comme précédemment, *centroid* est toujours combiné à *profile*. Sauf mention contraire, *minVal* est combiné aux autres heuristiques

Le solveur `choco` propose maintenant plusieurs heuristiques d'arbitrages basées sur la fonction proposée par Laborie [122] : $preserved(x \leq y)$ (voir chapitre 9). Ces dernières n'ont pas été évaluées pendant cette thèse.

7.1.3 Notes sur les modèles

L'heuristique du degré pondéré est très sensible à la modélisation puisque deux ensembles de contraintes logiquement équivalents provoquent des répartitions différentes des poids. En effet, les poids dépendent des contraintes du modèle, mais aussi de l'ordre de propagation des contraintes par le solveur. Lorsqu'une inconsistance peut être détectée par plusieurs contraintes, l'ordre de propagation des contraintes détermine quel poids va être incrémenté. La présence des contraintes disjonctives perturbe donc l'apprentissage des poids si elles sont propagées avant les disjonctions. Par contre, les réductions supplémentaires des fenêtres de temps augmentent la précision du profil des ressources ce qui peut bénéficier à *profile*.

La première qualité de la contrainte globale associée au graphe de précédence du chapitre 6 est d'offrir une propagation optimisée et incrémentale sur le sous-réseau de contraintes. Ensuite, le branchement et le filtrage peuvent exploiter l'information structurelle à propos du graphe de précédences (fermeture transitive, ordre topologique). Enfin, on ne réalise aucune hypothèse sur l'ensemble des disjonctions $\mathcal{D}$ du problème ce qui permet de traiter éventuellement des problèmes dynamiques ou cumulatifs.
En revanche, les contraintes sont ajoutées de manière réversible et toute information associée est perdue lors d'un retour arrière. L'apprentissage des poids des disjonctions est donc compliqué d'autant plus qu'il faudrait expliquer les échecs provoqués par la contrainte globale. Par ailleurs, la gestion des temps d'attente dans un graphe disjonctif généralisé n'est pas immédiate puisqu'elle modifie la relation de transitivité.

Les contraintes de disjonctions réifiées représentant le graphe disjonctif sont taillées sur mesure pour l'apprentissage des poids. L'introduction des variables booléennes $b_{ij \preceq kl}$ permet d'imposer facilement des restrictions supplémentaires sur les gammes opératoires par l'intermédiaire de contraintes booléennes. Elles facilitent aussi l'enregistrement des *nogoods* qui sont toujours définis pour une borne supérieure courante *ub* mais par un ensemble d'instanciations de variables booléennes $b_{ij \preceq kl}$ au lieu d'un ensemble de contraintes de précédence P. Un *nogood* est alors défini par une clause booléenne dont la propagation peut être déléguée au solveur et bénéficier ainsi d'éventuelles fonctionnalités dédiées au problème SAT. De plus, la gestion du graphe disjonctif généralisé demande uniquement une modification mineure de l'implémentation des contraintes de disjonction. Finalement, nous verrons qu'une propagation naïve permet d'explorer rapidement un nombre conséquent de nœuds améliorant ainsi l'apprentissage des poids.
L'inconvénient majeur est que ce modèle implique de fixer l'ensemble des disjonctions $\mathcal{D}$ avant le début de

la recherche. Ensuite, la propagation des précédences est déléguée au solveur et il devient alors difficile de l'optimiser puisqu'elle dépend de l'ordre dans lequel les contraintes sont propagées. La perte de l'information structurelle complique la détection de la transitivité et des cycles. Pour pallier cet inconvénient, on peut envisager d'implémenter une nouvelle contrainte globale structurelle (ne propageant pas les fenêtres de temps) ou de poser des clauses booléennes associées aux inégalités triangulaires entre les précédences (voir section 9.4). Dans ce chapitre, nous n'appliquons aucune de ces alternatives.

7.2 Algorithme de résolution

La solution initiale est calculée en `choco` par `CROSH`, mais par une variante incomplète de dichotomic-bottom-up (voir section 2.4) en `mistral` pour cause d'incompatibilités de langages (Java vs C). Dans cette variante, nous imposons une limite de temps à chaque étape.
Dans toutes les évaluations, les *nogoods* sont enregistrés lors des redémarrages. Les *nogoods* sont maintenant définis par des clauses booléennes. La contrainte globale propageant les clauses n'incrémente son poids lors de la détection d'une inconsistance. En effet, cette information est non discriminante puisque toutes les variables $b_{ij \preceq kl}$ appartiennent à cette contrainte globale. De surcroît, cette augmentation peut influencer les décisions ultérieures, par exemple lorsque les variables de décision ne sont pas toutes booléennes.

7.3 Évaluations expérimentales

Toutes les expérimentations ont été menées sur une grille de calcul composée d'Intel Xeon cadencés à 2.66GHz (12GB de mémoire vive) tournant sous Fedora 9. La configuration matérielle est plus puissante que dans le chapitre 6. Les évaluations concernent les problèmes d'open-shop (section 7.3.1) et de job-shop (section 7.3.2). La durée d'une exécution de l'algorithme est limitée à 3600 secondes. Les évaluations apportent la preuve qu'il est possible de résoudre ces problèmes d'atelier sans utiliser de techniques complexes d'inférence et de branchement. Nous avons réalisé différentes expérimentations dans le but de : (a) comparer les différents modèles (*Light*, *Heavy*) et heuristiques (*profile*, *slack/wdeg*) pour l'open-shop et le job-shop ; (b) évaluer l'importance des deux composantes de l'heuristique *slack/wdeg*.
Le solveur `choco` permettra une comparaison poussée des différentes configurations sur l'open-shop. Nous utiliserons aussi une implémentation du modèle *Light* avec le solveur `mistral` [27] dont nous verrons que la rapidité est un atout majeur pour la résolution de ce modèle. Ensuite, nous comparerons le modèle *Light* en `mistral` avec un modèle voisin du *Heavy* implémenté avec `Ilog Scheduler` (aimablement fourni par Beck [123].

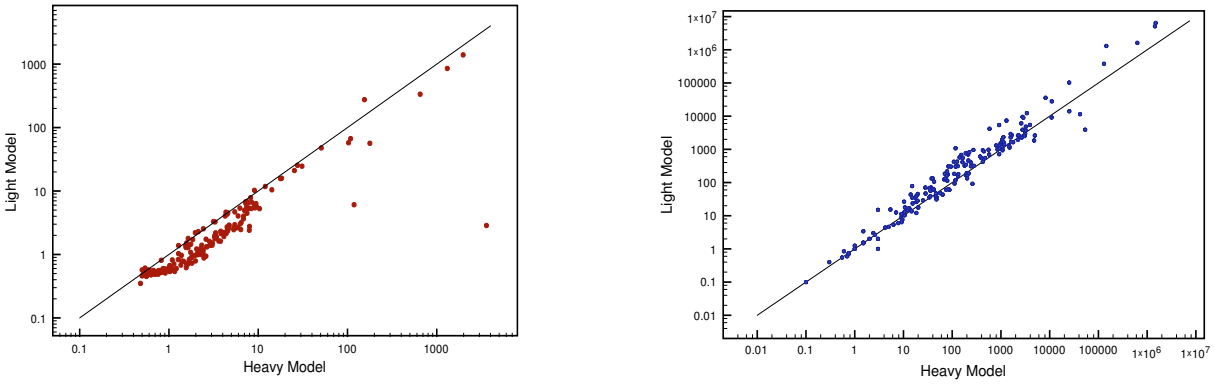
7.3.1 Problèmes d'open-shop

Nous utilisons les jeux d'instances de la littérature Taillard, Brucker et Guéret-Prins composés de 192 instances dont la taille varie entre 3×3 et 20×20 (voir section 6.3).

7.3.1.1 Comparaison des modèles *Light* et *Heavy*

Nous comparons 4 modèles implémentés en `choco`, les modèles classiques *Light* et *Heavy* (« *Light-slack/wdeg* » et « *Heavy-profile* »), ainsi que les variantes où les heuristiques sont inversées (*Light-profile* et *Heavy-slack/wdeg*). Le modèle *Light* en `mistral` se différencie de celui de `choco` lors du calcul de la solution initiale par une procédure d'optimisation dichotomique à la place de `CROSH`. Pour comparer les deux solveurs, le modèle *Light* utilise la politique de redémarrage de Walsh où les facteurs d'échelle et géométrique sont respectivement égaux à 256 et 1.3. Tous les autres modèles utilisent les réglages finaux du chapitre 6. Les modèles sont résolus 20 fois avec une limite de temps de 3600 secondes. Le générateur de nombre aléatoire utilise une graine différente à chaque exécution.

La figure 7.1 compare la résolution des 192 instances d'open-shop par les modèles *Light* et *Heavy*. Chaque point représente une instance et ses coordonnées x et y sont respectivement les temps de résolution (figure 7.1(a)) et les nombres de nœuds (figure 7.1(a)) du modèle *Heavy* et *Light*. Les échelles des axes sont logarithmiques.



(a) Temps de résolution. (b) Nombre de nœuds.

FIGURE 7.1 – Comparaison des modèles *Light* et *Heavy* sur le problème d'open-shop.

Le tableau 7.1 récapitule les résultats sur les 11 instances les plus difficiles, c'est-à-dire pour lesquelles le temps moyen de résolution est supérieur à 20 secondes pour au moins une approche. Cette sélection est constituée de 1 instance de Guéret-Prins, 4 de Taillard et 6 de Brucker dont la taille varie entre 7×7 et 20×20. Nous rapportons les temps moyens de résolution (colonne $\bar{t}$) en secondes et nombre moyen de nœuds visités (colonne $\bar{n}$) en milliers (K) ou millions (M). Les moyennes intègrent les résultats des exécutions pour lesquelles le modèle n'a pas prouvé l'optimalité dans la limite de temps (3600 secondes). Ces moyennes sont donc en fait des bornes inférieures. Les champs en gras sont les minima sur l'ensemble des résultats.

	Light-slack/wdeg		*Heavy-profile*		*Light-profile*		*Heavy-slack/wdeg*		`mistral`	
Name	$\bar{t}$	$\bar{n}$	$\bar{t}$	$\bar{n}$	$\bar{t}$	$\bar{n}$	$\bar{t}$	$\bar{n}$	$\bar{t}$	$\bar{n}$
`GP10-01`	6.1	4K	118.4	53K	2523.2	6131K	9.6	**3K**	**0.3**	**3K**
`j7-per0-0`	854.5	5.1M	1310.9	**1.4M**	979.1	4.3M	3326.7	2.6M	**327.0**	8.7M
`j7-per10-2`	57.6	0.4M	102.7	**0.1M**	89.5	0.4M	109.6	**0.1M**	**33.5**	1.2M
`j8-per0-1`	1397.1	6.4M	1973.8	**1.5M**	1729.3.	6.0M	3600.0	2.4M	**427.1**	10M
`j8-per10-0`	24.6	0.10M	30.9	**0.02M**	19.7	0.05M	68.0	0.06M	**17.6**	0.47M
`j8-per10-1`	275.2	1.3M	154.8	**0.1M**	92.8	0.3M	796.7	0.7M	**89.3**	2.3M
`j8-per10-2`	335.3	1.6M	651.4	**0.6M**	754.0	2.7M	697.1	**0.6M**	**93.1**	2.3M
`tai-20-1`	25.4	**2K**	27.5	3K	2524.7	1458K	34.4	3K	**3.9**	21K
`tai-20-2`	56.5	11K	178	42K	3600.0	2261K	81.9	**10K**	**14.1**	61K
`tai-20-7`	47.8	9K	60.9	11K	3600.0	2083K	63.7	**8K**	**9.5**	47K
`tai-20-8`	66.8	14K	108.3	25K	3600.0	2127K	84.8	**11K**	**8.2**	39K
Total Avg	286.1	1.3M	428.9	**0.4M**	1774.3	2.5M	806.6	0.6M	**93.0**	2.3M

TABLE 7.1 – Temps de résolution des 11 instances les plus difficiles du problème d'open-shop.

Les modèles *Light* et *Heavy* en `choco` résolvent toutes les instances. Par contre, le modèle *Light* est généralement plus rapide, notamment sur 10 des 11 instances les plus difficiles, malgré l'exploration d'un nombre bien plus important de nœuds. Le modèle *Light* en `mistral` est le plus rapide. Le nombre de nœuds de `mistral` est majoré, car il comprend la phase de recherche dichotomique. Il est intéressant de remarquer que sur les six instances difficiles de Brucker, le modèle *Heavy* est légèrement plus rapide que le modèle *Light* (`choco`) pour découvrir une solution optimale (310s vs 360s), mais est trois fois plus lent pour prouver son optimalité (385s vs 125s). Cela confirme notre intuition sur la capacité de l'heuristique du degré pondéré à identifier les paires de tâches critiques.

Dans l'espoir de mieux comprendre le lien entre les heuristiques et les modèles, nous examinons les variantes où les heuristiques sont inversées ce qui a entraîné dans les deux cas une dégradation des performances. Pour le modèle *Light*, *slack/wdeg* est significativement plus efficace que *profile*. En effet, la variante *Light-profile* ne prouve l'optimalité que dans 60 % des exécutions et leur temps moyen augmente

quasiment d'un ordre de grandeur par rapport à celui de *Light-slack/wdeg*. Pour le modèle *Heavy*, la situation s'inverse puisque *slack/wdeg* est en moyenne plus lent que *profile* même si le taux d'exécution optimale de *slack/wdeg* atteint 89 %. En effet, wdeg est moins discriminante dans le modèle *Heavy* où les nombreux échecs provoqués par les contraintes disjonctives (leurs inférences sont plus fortes) ne sont pas discriminants.

Finalement, des expérimentations supplémentaires sur l'utilisation de la stratégie de Luby avec le modèle *Light* ont également montré une dégradation des performances. Au contraire, nous avions montré que les réglages finaux du modèle *Heavy* obtenaient des performances équivalentes au chapitre 6.

7.3.1.2 Influence des composantes de slack/wdeg

Nous évaluons maintenant l'influence des deux composantes (taille des fenêtres de temps et degré pondéré de notre heuristique *slack/wdeg* sur la résolution du modèle *Light*. Pour cela, nous comparons la résolution du modèle *Light* de `mistral` par les heuristiques *top3-slack*, *top3-wdeg* et *slack/wdeg*. L'élément de randomisation supplémentaire des heuristiques *top3-slack* et *top3-wdeg* améliore la diversification au début de la recherche. Chaque étape de la procédure dichotomique est limitée à 30 secondes pour permettre à toutes les heuristiques de démarrer avec une bonne solution initiale.

	top3-slack		*top3-wdeg*		*slack/wdeg*	
Name	%	$\bar{t}$	%	$\bar{t}$	%	$\bar{t}$
`Brucker`	89.2	1529.2	100.0	381.4	100.0	249.7
`Taillard`	0.0	3600.0	93.7	1423.9	100.0	8.8
Total Avg	53.0	2358.5	97.0	798.4	100.0	153.3

TABLE 7.2 – Comparaison des heuristiques de sélection de disjonction sur le problème d'open-shop.

Le tableau 7.2 récapitule les pourcentages d'exécution prouvant la solution optimale (colonnes %) et les temps moyens de résolution (colonnes $\bar{t}$) pour 20 exécutions sur chaque instance. L'heuristique *top3-slack* est relativement peu efficace sur ces instances puisqu'elles ne prouvent l'optimum que pour 53 % des exécutions et que les bornes supérieures sont faibles pour les instances de Taillard.
Les résultats montrent clairement l'efficacité de *top3-wdeg* sur ces instances puisque l'optimum est prouvé pour 97 % des exécutions et la borne supérieure est à distance au plus unitaire de l'optimum en cas d'échec. Par contre, *top3-wdeg* ne discrimine pas les variables au début de la recherche puisque les degrés pondérés sont unitaires jusqu'à ce qu'une première contradiction soit levée. Cette première contradiction apparaît quelquefois profondément dans l'arbre de recherche selon la taille du problème, la borne supérieure initiale, la charge de travail des lots et machines. En fait,la sélection des disjonctions est aléatoire avant la première contradiction.
Finalement, la combinaison de l'information sur les domaines et les degrés pondérés dans *slack/wdeg* augmente le taux de résolution à 100 % tout en diminuant les temps de résolution. L'information sur les domaines : (a) discrimine les disjonctions au début de la recherche ; (b) améliore la qualité des premiers poids appris en privilégiant les disjonctions dont les arbitrages ont une probabilité élevée d'être inconsistants. Des expériences préliminaires, non détaillées ici, ont montré que la randomisation des x meilleurs choix dégrade les performances de *slack/wdeg* même quand x est petit. Cette dégradation s'accentue quand x augmente. Contrairement aux autres heuristiques, la randomisation de la sélection d'une disjonction est moins importante pour les heuristiques du degré pondéré, car l'apprentissage de la pondération joue le rôle d'un processus de diversification.

7.3.2 Problèmes de job-shop

Nous comparons maintenant le modèle *Light* en `mistral` à l'algorithme nommé *randomized restart* utilisé par Beck [123] et implémenté avec `Ilog Scheduler`. Il est important de remarquer que nous ne comparons pas à l'algorithme *solution guided multi point constructive search* (SGMPCS), mais à l'algorithme *randomized restart* qui est utilisé à titre de comparaison dans ses expérimentations. Nos expérimentations utilisent la configuration matérielle décrite plus haut et la version 6.2 d'`Ilog Scheduler`.

Toutes les valeurs des paramètres sont prises de [123]. La politique de redémarrage est donc celle de Luby sans facteur d'échelle ($s = 1$). Un élément de randomisation supplémentaire est ajouté en utilisant l'heuristique de sélection de disjonction *top10%-profile*. Finalement, plusieurs techniques d'inférence forte sont intégrées au modèle telles que les techniques de *time-table* [167], d'*edge finding* [168] et de la contrainte *balance* [169].
Les paramètres de `mistral` sont identiques à ceux de l'open-shop à l'exception d'une limite de 300 secondes pour chaque étape de la recherche dichotomique. Nous utilisons aussi l'heuristique d'arbitrage *jobOrdering* dédiée aux problèmes de job-shop et flow-shop qui est apparue comme meilleure lors d'expérimentations préliminaires.

Le tableau 7.3 récapitule les résultats sur 4 jeux de 10 instances pour le problème de job-shop proposés par Taillard [103] Les jeux correspondent à des tailles de problème ($n \times m$) : 20×15, 20×20, 30×15 et 30×20. Le nombre d'exécutions pour chaque instance et chaque algorithme est réduit à 10, car les instances sont rarement résolues optimalement dans la limite de temps de 3600 secondes (cette expérience a nécessité plus d'un mois de temps CPU). Les résultats sont rapportés sous la forme de valeurs moyennes sur chaque jeu d'instances. Concernant la valeur de l'objectif (*Makespan*), nous considérons pour chaque instance sa valeur moyenne (Avg), sa meilleure valeur trouvée (Best) et son écart-type (Std Dev). En suivant le protocole de [123], nous comparons aussi les deux approches en terme d'erreurs relatives moyennes (*mean relative error* – MRE) par rapport à la meilleure borne supérieure connue pour ces instances. MRE est la moyenne arithmétique de l'erreur relative de toutes les exécutions sur toutes les instances :

$$MRE(a, K, R) = \frac{1}{|R||K|} \sum_{r \in R} \sum_{k \in K} \frac{c(a, k, r) - c^*(k)}{c^*(k)}$$

où K est l'ensemble des instances, R est l'ensemble des exécutions, $c(a, k, r)$ est la valeur de l'objectif trouvée par l'algorithme a sur l'instance k pendant l'exécution r, et $c^*(k)$ est la meilleure borne supérieure connue pour l'instance k.

	`Ilog Scheduler`					`mistral`				
	Makespan			MRE		Makespan			MRE	
Name	Avg	Best	Std Dev	Avg	Best	Avg	Best	Std Dev	Avg	Best
`tai11-20`	1411.1	1409.9	11.12	0.0336	0.0294	**1409.9**	**1398.1**	23.17	**0.0328**	**0.0241**
`tai21-30`	**1666.0**	1659.0	13.51	**0.0297**	0.0254	1669.9	**1655.4**	25.05	0.0322	**0.0232**
`tai31-40`	1936.1	1927.1	18.67	0.0817	0.0767	**1922.5**	**1898.8**	37.56	**0.0742**	**0.0608**
`tai41-50`	2163.1	2153.1	17.85	0.1055	0.1004	**2143.9**	**2117.6**	44.85	**0.0958**	**0.0824**

TABLE 7.3 – Résultats sur le problème de job-shop.

Comme attendu, les résultats obtenus avec `Ilog Scheduler` sont comparables ou meilleurs que ceux rapportés par Beck [123] puisque nous utilisons une configuration matérielle plus puissante. Les deux approches ont des performances comparables sur les deux premiers jeux d'instance même si les meilleures solutions obtenues par `mistral` sont systématiquement meilleures que celles obtenues avec `Ilog Scheduler`. `mistral` passe mieux à l'échelle sur les deux derniers jeux d'instances composés d'instances de taille supérieure puisque les moyennes des objectifs moyens sont meilleures que les moyennes des meilleurs objectifs obtenues avec `Ilog Scheduler`. Par contre, le modèle *Light* est moins robuste puisque l'écart-type de `mistral` est plus important que celui d'`Ilog Scheduler`. En contrepartie, on peut espérer que le calcul parallèle sera plus bénéfique à `mistral` qu'à `Ilog Scheduler`.

7.4 Conclusion

Dans ce chapitre, nous avons montré que contrairement à une croyance répandue, les problèmes d'atelier peuvent être efficacement résolus par la combinaison d'un modèle offrant une propagation naïve et une heuristique simple inspirée du degré pondéré par l'algorithme décrit au chapitre 6. Nous avons

montré que cette approche surpassait les résultats de l'état de l'art sur de nombreuses instances d'open-shop et de job-shop. Par contre, ces performances restent inférieures à celles de l'algorithme *solution guided multi point constructive search* (SGMPCS) sur les problèmes de job-shop [123].

À la suite de ce travail, Grimes et Hebrard [170] ont montré la pertinence de cette approche sur deux variantes du problème de job-shop avec temps d'attente minimaux ou maximaux. Un mécanisme de guidage similaire à celui SGPMCS a aussi été incorporé. Cependant, une analyse détaillée de l'apprentissage des poids a montré que l'apprentissage des poids pouvait avoir un impact négatif sur une des deux variantes.

Ces modèles peuvent d'ores et déjà être reproduits dans `choco` (voir chapitre 9). Dans de futurs travaux, nous souhaitons comparer les modèles *Light* et *Heavy* sur ces deux variantes mais aussi sur d'autres avec des contraintes de fenêtre de temps qui sont a priori plus favorables aux inférences fortes du modèle *Heavy*.