Chapitre 4

Ordonnancement d'atelier

Nous rappelons la définition et la classification des problèmes d'atelier ainsi que les principaux résultats de complexité. Nous établissons ensuite un état de l'art en insistant sur le problème d'atelier à cheminement libre (open-shop), le problème de base retenu lors de nos expérimentations.

Sommaire

L'ordonnancement d'atelier consiste à exploiter au mieux des moyens limités, les machines, pour réaliser un ensemble varié de produits, les lots. La complexité ne réside pas dans le processus de fabrication prédéterminé mais plutôt dans la combinatoire qui naît de la prise en compte des limitations de ressources. Ce chapitre dresse un panorama de la classification et de la résolution des problèmes d'atelier. Nous nous focaliserons en particulier sur le problème d'atelier à cheminements libres et nous ne discuterons que des méthodes évoquées par la suite pour les autres problèmes d'atelier. L'objectif principal est de donner les clés pour la compréhension des chapitres 6 et 7 présentant notre méthode de résolution.
Ce chapitre est organisé de la manière suivante. La section 4.1 définit les problèmes d'atelier et présente leur classification. Ensuite, les sections 4.2 et 4.3 récapitulent respectivement les principaux résultats de complexité et méthodes de résolution. Finalement, la section 4.4 situe nos axes de recherche par rapport à la littérature.

4.1 Définition des problèmes d'atelier

Nous donnons dans cette section une définition des problèmes d'atelier et introduisons une classification avant d'illustrer ces problèmes par quelques applications réelles.

4.1.1 Définition des problèmes de base

Dans un problème d'atelier, une pièce doit être usinée ou assemblée sur différentes machines. Chaque machine est une ressource disjonctive, c'est-à-dire qu'elle ne peut exécuter qu'une tâche à la fois, et les tâches sont liées exclusivement par des contraintes d'enchaînement. Plus précisément, les tâches sont regroupées en n entités appelées travaux ou lots. Chaque lot est constitué de m tâches à exécuter sur m machines distinctes. Il existe trois types de problèmes d'atelier, selon la nature des contraintes liant les tâches d'un même lot. Lorsque l'ordre de passage de chaque lot est fixé et commun à tous les lots, on parle d'atelier à cheminement unique (*flow-shop*). Si cet ordre est fixé mais propre à chaque lot, il s'agit d'un atelier à cheminements multiples (*job-shop*). Enfin, si le séquencement des tâches des travaux n'est pas imposé, on parle d'atelier à cheminements libres (*open-shop*). Un critère d'optimalité souvent étudié est la minimisation du délai total de l'ordonnancement (*makespan*). Ce critère est particulièrement intéressant puisque les ordonnancements au plus tôt constituent des sous-ensembles dominants [1] pour de nombreux critères réguliers. De plus, l'analyse des ordonnancements au plus tôt permet de déterminer des chemins critiques, c'est-à-dire des chemins sur lesquels tout retard a des conséquences sur toute la chaîne, ou des goulots d'étranglement, c'est-à-dire les étapes qui vont limiter la production de tout l'atelier.
Nous supposons que les durées d'exécution des tâches sont données par une matrice entière $P : m \times n$, dans laquelle $p_{ij} \geq 0$ est la durée d'exécution de la tâche $T_{ij} \in T$ du lot J_j réalisée sur la machine M_i.
Une borne inférieure classique pour les problèmes d'atelier, notée C_{max}^{LB}, est égale au maximum de la charge des machines et des durées des lots :

$$C_{max}^{LB} = \max \left(\max_{1 \leq i \leq m} \left(\sum_{j=1}^{n} p_{ij} \right), \max_{1 \leq j \leq n} \left(\sum_{i=1}^{m} p_{ij} \right) \right).$$

La figure 4.1 illustre un ordonnancement optimal d'un problème d'open-shop où chaque ligne correspond à l'ordonnancement d'une machine alors que les tâches d'un même lot sont identifiées par leur couleur.

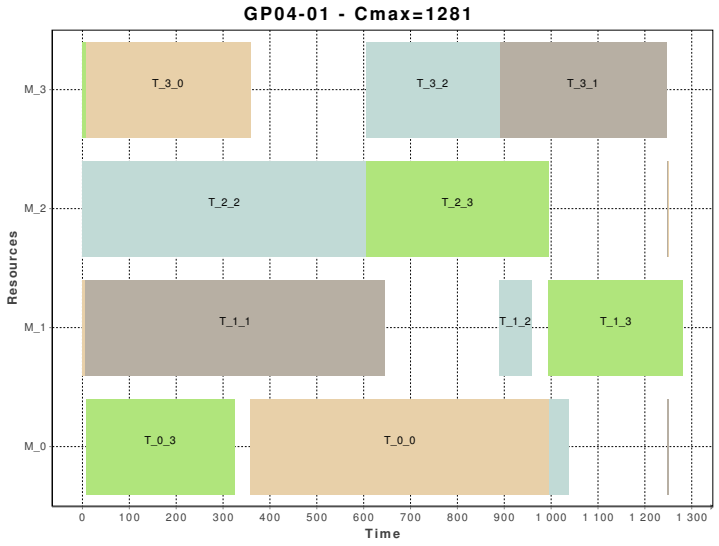

FIGURE 4.1 – Un ordonnancement optimal de délai total 1170 pour l'instance d'open-shop `GP04-01`.

4.1.2 Variantes et classification de Lawler

Les variantes des problèmes d'atelier imposent généralement des restrictions supplémentaires sur l'ordonnancement (contraintes temporelles, contraintes de ressource) ou autorisent la préemption des tâches. Le problème d'open-shop ne contient pas, comme le job-shop ou le flow-shop, de contraintes de précédence

1. Un ensemble de solutions d'un problème d'optimisation est dit dominant s'il contient au moins une solution optimale.

entre les tâches d'un même lot. Cependant, il peut exister des précédences entre les lots.
Dans le cas préemptif, l'exécution d'une opération T_{ij} peut être interrompue par l'exécution d'une autre tâche. Elle sera alors terminée plus tard, éventuellement après d'autres interruptions. Notons que dans l'open-shop, la tâche préemptée peut appartenir au même lot, ce qui n'est pas permis dans les autres problèmes d'atelier à cause des contraintes de précédence.
Enfin, dans certains problèmes appelés problèmes d'atelier sans attente (*no-wait problems*), les lots doivent être exécutés d'une seule traite, sans attente entre les machines. En d'autres termes, il n'y a pas de préemption des lots. Cette contrainte peut être due à l'absence de capacité de stockage intermédiaire ou au processus de fabrication lui-même.

Nous utiliserons la classification des problèmes d'ordonnancement suivant la notation $\alpha|\beta|\gamma$ proposée par Graham *et al.* [87] et étendue par Lawler *et al.* [88]. Notons o le symbole vide. Par souci de clarté, nous négligerons la double indexation des tâches T_{ij} dans un atelier.

- α permet de spécifier l'environnement machine : $\alpha = \alpha_1; \alpha_2$.
  - $\alpha_1 = O$ pour l'open-shop, F pour le flow-shop et J pour le job-shop.
  - α_2 correspond au nombre de machines, o si ce nombre n'est pas fixé.
- β décrit les caractéristiques des tâches : $\beta = \beta_1; \beta_2; \beta_3; \beta_4; \beta_5; \beta_6$.
  - $\beta_1 = pmtn$ dans le cas d'un problème préemptif, o sinon.
  - $\beta_2 = prec$ si des contraintes de précédence existent entre les lots, *tree* si le graphe de précédences est une arborescence, o s'il n'existe pas de contraintes de précédence.
  - $\beta_3 = r_j$ si des dates de disponibilité sont associées aux tâches, o sinon.
  - $\beta_4 = d_j$ si des échéances sont associées aux tâches, o sinon.
  - $\beta_5 = (p_j = p)$ si les tâches ont des durées identiques, $(p_j = 1)$ si les tâches ont des durées unitaires, o sinon.
  - $\beta_6 = no\text{-}wait$ pour les problèmes d'atelier sans attente, o sinon.
- γ est le critère d'optimalité du problème. Notons C_j les dates d'achèvement des tâches.
  - $\gamma = f_{max} (= \max f_j)$ où $f_j(t)$ est la fonction de coût de la tâche j en fonction de sa date d'achèvement t. toutes les fonctions de coûts étudiées sont régulières, c'est-à-dire f_j est une fonction non décroissante de t pour $1 \leq i \leq n$.
  - $\gamma = C_{max}$ $(= \max C_j)$ si l'on cherche à minimiser le délai total ou date d'achèvement maximale (*makespan*).
  - $\gamma = \sum C_j$ si l'on veut minimiser le délai moyen (*flow time*).
  - $\gamma = \sum w_j C_j$ si l'on veut minimiser le délai moyen pondéré (*weighted flow time*).
  - $\gamma = L_{max}$ $(= \max(C_j - d_j))$ si l'on veut minimiser le retard algébrique maximal (*maximum lateness*).
  - $\gamma = \sum U_j$ si l'on veut minimiser le nombre de tâches en retard (*number of late jobs*). La fonction U_j indique si la tâche j est en retard (la valeur de U_j est 1 si la tâche j est en retard $(C_j > d_j)$ et égale à 0 autrement).

Une fonction de coût f_j est dite additive si $f_j(t + \Delta) = f_j(t) + f_j(\Delta)$ pour tout Δ et incrémentale si $f_j(t + \Delta) = f_j(t) + \Delta$. Le délai moyen pondéré d'une tâche est une fonction additive, alors que son retard algébrique est une fonction incrémentale.

4.1.3 Applications

Dans la réalité, un lot correspond généralement à un produit qui doit subir des opérations sur différentes machines. Un exemple classique est le problème d'ordonnancement d'un garage dans lequel des voitures (lots) doivent passer sur différents postes de travail (machines) comme la vidange, le graissage, le lavage ... On retrouve aussi des problèmes de ce type dans l'accomplissement de formalités administratives (passage à différents guichets), dans l'organisation d'examens médicaux de patients hospitalisés, dans la maintenance d'avion lors d'une escale, en télécommunications ... Ces applications peuvent comporter des contraintes de séquencement ou pas : un examen radiologique doit être programmé avant une consultation avec un spécialiste ; un étudiant doit récupérer une convention de stage et un relevé de notes dans un ordre quelconque. Beaucoup de problèmes d'emploi du temps peuvent être vus comme des problèmes d'open-shop avec des contraintes de partage de ressource supplémentaires.

4.2 Résultats de complexité

De nombreux travaux ont été effectués sur la complexité des problèmes d'atelier. Très peu de problèmes d'atelier peuvent être résolus en temps polynomial. Dans la quasi-totalité des cas polynomiaux, les algorithmes sont conçus pour construire des ordonnancements de durée C_{max}^{LB}. Les cas les plus connus sont les problèmes à deux machines ou préemptifs.

Nous allons résumer les principaux résultats connus à ce jour. Nous nous focaliserons sur le délai total C_{max}. Nous donnerons cependant quelques résultats concernant d'autres critères.

4.2.1 Problèmes à deux machines

Parmi les problèmes d'open-shop non préemptifs, le problème $O2||L_{max}$ dans lequel on cherche à minimiser le retard algébrique maximal des tâches est un problème NP-difficile *au sens fort* [89], de même que le problème $O2||\sum C_j$ [90]. Par contre, le problème $O2||C_{max}$ admet un algorithme linéaire O(n) proposé par Gonzalez et Sahni [91] qui construit un ordonnancement optimal sans temps mort de durée C_{max}^{LB}. Il construit séparément deux ordonnancements pour une partition des lots correspondant à la relation $p_{1i} \geq p_{2i}$ avant de concaténer ces deux solutions partielles pour former un ordonnancement optimal. Remarquons que cet algorithme est aussi valide dans le cas préemptif. Lawler *et al.* [89] proposent un algorithme en O(n) pour résoudre $O2|pmtn; r_j|C_{max}$.
Un ordonnancement optimal pour le problème $F2||C_{max}$ peut être recherché parmi les ordonnancements de permutation (la séquence d'exécution des lots est identique sur toutes les machines). On applique la condition suffisante d'optimalité donnée par la *règle de Johnson* [92] : le lot i précède le lot j dans la séquence optimale si $\min(p_{i1}, p_{j2}) \leq \min(p_{i2}, p_{j1})$. L'algorithme de Johnson construit un tel ordonnancement en temps polynomial. L'algorithme de Jackson [93] basé sur la règle de Johnson résout le problème $J2||C_{max}$ en temps polynomial. En revanche, $J2|pmtn|C_{max}$ est NP-difficile, même si $F2|pmtn|C_{max}$ peut être résolu en temps polynomial [94]

4.2.2 Problèmes à trois machines : la frontière

Peu de problèmes non préemptifs avec un nombre de machines supérieur à 2 sont polynomiaux. Gonzalez et Sahni [91] démontrent que le problème $O3||C_{max}$ est NP-difficile *au sens faible*. Cependant, certains cas particuliers sont polynomiaux, Adiri et Aizikowitz [95] exploitent l'existence d'une relation de dominance pour construire un ordonnancement optimal à deux machines avant de placer les tâches sur la machine dominée sans créer de conflits.
Un ordonnancement optimal du problème $F3||C_{max}$ peut encore être recherché parmi les ordonnancements de permutation. La règle de Johnson peut être étendue lorsque la seconde machine n'est pas un goulet d'étranglement, c'est-à-dire qu'elle est complètement dominée par une des deux autres machines. Le problème est alors résolu après transformation en un problème à deux machines. Par contre, le problème préemptif $F3|pmtn|C_{max}$ est NP-difficile [94].

4.2.3 Cas général

Graham *et al.* [87] ont démontré la NP-difficulté *au sens fort* de $O||C_{max}$. Le problème devient polynomial lorsque la plus longue tâche est assez courte par rapport à la charge maximale des machines [96]. de Werra [97], de Werra et Solot [98] construisent des ordonnancements optimaux de durée C_{max}^{LB} quand la structure du graphe biparti pondéré associé à l'instance permet le calcul d'une coloration en temps polynomial. Les problèmes $F||C_{max}$ et $J||C_{max}$ sont aussi NP-difficiles *au sens fort*.
Les problèmes préemptifs sont, en général, plus faciles à résoudre que les non préemptifs car on peut faire appel aux mathématiques du continu. Cho et Sahni [99] utilisent la programmation linéaire pour résoudre le problème $O|pmtn|L_{max}$. Lawler *et al.* [89] proposent un algorithme en O(n) pour résoudre $O2|pmtn; r_j|C_{max}$. Gonzalez et Sahni [91] proposent un algorithme basé sur la construction de couplages de cardinal maximal dans un graphe biparti pour $O|pmtn|C_{max}$. En revanche, nous avons vu précédemment que les problèmes préemptifs de job-shop et flow-shop sont tous deux NP-difficiles pour $m \geq 3$.
Le problème $O|r_j; d_j|C_{max}$ est NP-difficile dans le cas général, même si les tâches ont des durées unitaires sauf dans certains cas particuliers. Graham *et al.* [87] ont démontré que $O|tree|C_{max}$ est NP-difficile au

sens fort pour $m \geq 2$, sauf dans certains cas particuliers où les tâches ont des durées unitaires. Pour tous les critères usuels, les problèmes d'open-shop sans attente sont NP-difficiles *au sens fort*, même les problèmes ne comportant que deux machines [100]. Ces problèmes restent NP-difficiles au sens fort lorsque toutes les tâches ont des durées nulles ou identiques [101].
Il est évident que les problèmes d'atelier avec ressources supplémentaires (c'est-à-dire autres que les machines) ne sont pas plus faciles que les mêmes problèmes sans ressource. Ainsi, tous les résultats de NP-difficulté vus précédemment restent valables pour les problèmes avec ressources. Pire, l'introduction de ressources complique le cas préemptif. de Werra *et al.* [102] ont démontré que le problème $O|pmtn|C_{max}$ avec une ressource renouvelable ou consommable est NP-difficile *au sens fort*.

4.2.4 Conclusion

En résumé, les problèmes d'atelier non préemptif sont NP-difficiles dans le cas général. Le problème d'open-shop préemptif est polynomial contrairement aux problèmes de flow-shop et job-shop, mais il devient NP-difficile lorsque l'on ajoute une ressource. Le lecteur intéressé par la complexité des problèmes d'atelier peut se référer au site complexity results for scheduling problems qui actualise régulièrement ces résultats de complexité.

4.3 État de l'art

Nous allons maintenant présenter les méthodes de résolution pour les problèmes d'atelier non préemptif où l'on cherche à minimiser le délai total, et plus particulièrement pour le problème d'open-shop, le problème de base retenu lors des évaluations des chapitres 6 et 7. L'open-shop a été étudié tardivement par les chercheurs au vu de l'intérêt porté au flow-shop et au job-shop. Les premiers travaux ont principalement concerné les résultats de complexité, l'étude de cas particuliers et la démonstration des garanties de performance pour les heuristiques simples. Depuis, plusieurs approches exactes ou approchées ont été proposées sans toutefois égaler le nombre de travaux portant sur le problème de job-shop. Pour le job-shop, nous nous restreindrons aux dernières approches de la littérature discutées dans le chapitre 7. Par contre, nous n'aborderons pas les approches pour le flow-shop, car ce problème n'a pas été retenu pour nos évaluations.

4.3.1 Jeux d'instances

Trois jeux d'instances pour $O//C_{max}$ sont disponibles dans la littérature. Le premier est constitué de 60 problèmes proposés par Taillard [103] comprenant entre 16 tâches (4 lots et 4 machines) et 400 tâches (20 lots et 20 machines). Il est considéré comme facile puisque la preuve d'optimalité est triviale, c'est-à-dire que la date d'achèvement maximale C_{max} est égale à la borne inférieure C^{LB}_{max}. Brucker *et al.* [104] ont proposé 52 instances difficiles allant de 3 lots et 3 machines à 8 lots et 8 machines. Finalement, le dernier jeu d'instances est constitué de 80 instances proposées par Guéret et Prins [105]. Leurs tailles varient de 3 lots et 3 machines jusqu'à 10 lots et 10 machines et le temps d'achèvement maximal est toujours strictement supérieur à la borne inférieure. Notons que la borne inférieure C^{LB}_{max} est toujours égale à 1000 pour les instances de Brucker et Guéret-Prins.
Taillard [103] a aussi proposé un jeu de 476 instances pour $J||C_{max}$ comprenant 15 ou 20 machines et entre 15 et 100 lots, ainsi qu'un jeu d' instances pour $F||C_{max}$ comprenant 5, 10 ou 20 machines et entre 20 et 500 jobs.

4.3.2 Méthodes approchées

Nous présentons ici les principales heuristiques et métaheuristiques pour les problèmes d'atelier.

4.3.2.1 Heuristiques

Williamson *et al.* [106] ont montré qu'il n'existe pas d'heuristique polynomiale pour $O||C_{max}$ ayant une garantie de performance $\frac{ub}{opt}$ inférieure à $\frac{5}{4}$. On ne dispose pas encore d'heuristique garantie à $\frac{5}{4}$ mais

on peut atteindre une garantie de $\frac{3}{2}$ pour les problèmes à 3 machines.
Une heuristique de liste ou permutation construit un ordonnancement sans temps mort en ajoutant itérativement des tâches à un ordonnancement partiel [107]. Un ordonnancement est dit sans temps mort si aucune machine n'est inactive lorsqu'il est possible d'exécuter un lot. En partant d'un ordonnancement vide, l'algorithme applique, à chaque itération, les étapes suivantes : (a) calculer la date minimale t_0 à laquelle des opérations sont disponibles (il existe au moins un lot et une machine disponibles au temps t_0), (b) ordonnancer une des tâches disponibles à l'aide d'une règle de priorité (*priority dispatching rule*). Dans le cas du flow-shop et du job-shop, une opération est disponible si l'opération la précédant immédiatement dans son lot est achevée. Nous rappelons maintenant quelques-unes des règles de priorité classiques :

random les opérations sont traitées dans un ordre aléatoire,
first fit (FF) les opérations sont traitées dans l'ordre lexicographique,
shortest processing time (SPT) les opérations sont triées par durées croissantes,
longest processing time (LPT) les opérations sont triées par durées décroissantes,
most work remaining (MWR) la priorité est donnée au travail dont la durée résiduelle (durée totale des opérations non traitées) est maximale.

Guéret et Prins [108] et Bräsel *et al.* [109] proposent deux heuristiques originales pour le problème général $O||C_{max}$. La première est basée sur la construction de couplages dans un graphe biparti. La seconde construit un ordonnancement en combinant des techniques d'insertion avec une recherche arborescente tronquée. Dewess *et al.* [110] proposent une méthode de branch-and-bound tronqué travaillant sur l'affectation et la propagation des dates de début au plus tôt et les dates de fin au plus tard.

L'heuristique de la machine goulet, initialement conçue pour $J||C_{max}$, est liée au concept de goulet d'étranglement glissant et à la minimisation du retard algébrique maximum sur une machine par une procédure arborescente pour déterminer un arbitrage complet du graphe disjonctif (voir section 3.4). Pour chaque machine, le problème $1|r_j|L_{max}$ est défini par les fenêtres de temps des tâches après la propagation des précédences du graphe disjonctif pour un ordonnancement au plus tôt. À chaque itération, un branch-and-bound détermine la séquence des opérations de la machine la plus en retard, le goulet d'étranglement, qui est ensuite ajoutée au graphe disjonctif jusqu'à ce qu'il n'y ait plus de conflits (aucune tâche n'est en retard). Des variantes considèrent d'autres critères d'optimalité et types d'atelier, notamment l'open-shop.
Campbell *et al.* [111] proposent une extension multi-étapes de la règle de Johnson générant au plus $m-1$ solutions pour le problème $F||C_{max}$.

4.3.2.2 Métaheuristiques

De nombreuses métaheuristiques pour résoudre l'open-shop ont été développées durant les dernières décennies. Taillard [103] a utilisé une recherche taboue pour résoudre des problèmes carrés de taille 4 à 20 générés aléatoirement. Il propose dans cet article un ensemble de problèmes pour lesquels la recherche taboue fournit des solutions très éloignées de la borne inférieure classique C_{max}^{LB}. Lors des évaluations, nous discuterons de quelques-unes des métaheuristiques les plus récentes et efficaces : un algorithme génétique [112], un algorithme de recherche locale par construction et réparation [113], un algorithme de colonies de fourmis [114] et un algorithme d'essaim de particules [115].
Prins [112] présente quelques algorithmes génétiques dédiés à $O||C_{max}$ basés sur deux idées originales : une population dans laquelle les individus ont des temps d'achèvement maximal distincts ; une procédure spéciale qui réarrange chaque nouveau chromosome.
Chatzikokolakis *et al.* [113] proposent un opérateur générique de réparation basé sur des techniques de recherche locale couplées à une fonction de coût évaluant les affectations partielles.
Une composante majeure de l'optimisation par colonies de fourmis est le mécanisme probabiliste de construction d'une solution. En raison de sa nature constructive, il peut être vu comme une recherche arborescente. De ce fait, Blum [114] combine le mécanisme de construction d'une solution avec un branch-and-bound tronqué en largeur *beam search*. *Beam search* est une méthode incomplète où les affectations partielles sont étendues un nombre limité de fois (cette limite est appelée *beam width*). Cette approche améliora les résultats expérimentaux obtenus par rapport aux meilleurs algorithmes d'optimisation par colonies de fourmis.
L'optimisation par essaim de particules est un algorithme d'optimisation basé sur une population. L'es-

saim est composé de particules qui représentent des solutions distinctes. Sha et Hsu [115] adaptent à l'open-shop la représentation de la position, du mouvement et de la vitesse des particules. Ils atteignent beaucoup de solutions optimales pour les instances de la littérature.

Watson et Beck [116] ont proposé une recherche taboue multi-phases pour le job-shop appelée *i-STS* qui est basée sur l'algorithme de référence *i-TSAB* proposé par Nowicki et Smutnicki [117]. Le but de *i-STS* est de simplifier l'étude de l'influence des diverses composantes de *i-TSAB* sur les performances globales. Cependant, les performances de *i-STS* sont légèrement inférieures à celles de *i-TSAB* sur le jeu d'instance de Taillard au regard de la qualité des solutions pour un même nombre d'itérations.

4.3.3 Méthodes Exactes

Brucker *et al.* [118] ont proposé une recherche arborescente reposant sur la résolution d'un problème à une machine avec des temps d'attente imposés entre les dates de démarrage des tâches. Le principe consiste à transformer le problème d'open-shop en un problème à une machine en mettant bout à bout les m machines. Des temps d'attente minimaux et maximaux sont ensuite imposés entre les dates de début des tâches afin d'être sûr qu'elles seront bien exécutées sur leur machine. Le problème à une machine ainsi obtenu est ensuite résolu par une recherche arborescente dans laquelle les auteurs adaptent les concepts d'arbitrages triviaux et les bornes inférieures utilisés dans la recherche arborescente présentée ci-après.
Brucker *et al.* [104] proposent une recherche arborescente qui arbitre à chaque nœud des disjonctions sur le chemin critique d'une solution heuristique. Malgré l'efficacité de cette méthode, des instances de Taillard ne sont pas résolues à partir de la taille 7×7.
Guéret *et al.* [119] proposent un algorithme avec retour arrière intelligent (*intelligent backtracking*) appliqué à la recherche arborescente de Brucker. Quand une contradiction est levée pendant la recherche, au lieu de revenir systématiquement à la décision précédente (*chronological backtracking*), l'algorithme analyse les causes de la contradiction pour éviter de remettre en question les décisions qui ne sont pas reliées à l'échec afin de revenir sur une décision plus pertinente. Cette approche réduit significativement le nombre de backtracks mais peut s'avérer coûteuse en temps de calcul. Ils appliquèrent plus tard des techniques supplémentaires de réduction de l'arbre de recherche basées sur les intervalles interdits, c'est-à-dire des intervalles temporels dans lesquelles aucune tâche ne peut débuter ou s'achever dans une solution optimale [120].
Dorndorf *et al.* [121] améliorent la recherche arborescente de Brucker par l'usage de techniques avancées de consistance. Au lieu d'analyser ou d'améliorer la stratégie de recherche, ils se concentrent sur l'introduction de techniques de propagation de contraintes pour réduire l'espace de recherche. Ils étudient aussi les performances des procédures d'optimisation top-down et bottom-up présentées au chapitre 2 en fonction de la charge de travail moyenne (*average workload*) de l'instance. Leurs algorithmes furent les premiers à résoudre optimalement un nombre conséquent d'instances en un temps raisonnable. Par contre, certaines instances de Taillard ne sont pas résolues à partir de la taille 15×15 ainsi que des instances de Brucker à partir de la taille 7×7.
Laborie [122] a proposé une recherche arborescente embarquée dans une procédure bottom-up pour la résolution de problèmes d'ordonnancement cumulatif. L'algorithme utilise des techniques avancées d'ordonnancement sous contraintes combinées à des techniques prospectives (*self-adapted shaving*). La stratégie de recherche est basée sur la détection de *minimal critical sets* (MCS), c'est-à-dire des ensembles minimaux de tâches allouées à une même ressource dont l'exécution simultanée entraînerait un dépassement de la capacité. Une heuristique sélectionne le MCS qui maximise la réduction de l'espace de recherche induite par son arbitrage. Cette approche a fermé 34 instances ouvertes de Guéret-Prins et 3 des 6 instances ouvertes de Brucker.
Plus récemment, Tamura *et al.* [31] applique à l'open-shop une méthode d'encodage d'un modèle en satisfaction/optimisation de contraintes composé de clauses sur des contraintes linéaires entières vers un problème de satisfiabilité booléenne (*boolean satisfiability testing problem* – SAT). Les comparaisons $x \leq a$ sont représentées par des variables booléennes différentes pour chaque variable $x \in \mathcal{X}$ et chaque valeur $a \in \mathcal{D}(x)$. Un modèle simple basé sur les contraintes de disjonction entre les tâches partageant un même lot ou une même machine et les contraintes d'achèvement est alors transformé vers SAT. Ils prouvent l'optimalité de toutes les instances dont les trois dernières instances ouvertes de Brucker.

Pour le job-shop au chapitre 7, nous nous appuierons surtout sur les résultats obtenus par Beck [123]

lorsqu'il a proposé la méthode *solution-guided multi-point constructive search* (SGMPCS). SGMPCS est une technique constructive qui réalise une série de recherches arborescentes tronquées en partant d'une affectation vide ou d'une solution découverte précédemment pendant la recherche. Un nombre restreint de solutions d'« élite » est conservé. Après une configuration et une étude poussée de leur algorithme, il montre expérimentalement que SGMPCS domine les principales méthodes constructives, mais reste légèrement moins efficace que la recherche taboue *i-STS*. Récemment, Watson et Beck [116] ont proposé une approche hybride simple entre *i-STS* et SGMPCS qui s'avère compétitive face aux meilleures méthodes de la littérature (dont *i-TSAB*).

4.4 Orientation de nos travaux

Une particularité de l'open-shop, par rapport au flow-shop ou au job-shop, est l'absence de contraintes de précédence. Cette absence augmente la combinatoire du problème et empêche l'adaptation de résultats et algorithmes conçus pour d'autres problèmes d'atelier. Par exemple, on doit considérer un plus grand nombre de disjonctions pour obtenir un arbitrage complet du graphe disjonctif. Par ailleurs, le problème obtenu par transposition des lots et des machines est équivalent au problème original.

La lecture de l'état de l'art montre qu'il n'existe pas d'approche de type top-down basée sur l'ordonnancement sous contraintes qui rivalise avec les meilleures méthodes approchées ou exactes. Or, il est crucial d'obtenir rapidement des solutions comparables à celles des métaheuristiques lors de la résolution de grandes instances, par exemple dans un contexte industriel où l'on assiste à la banalisation des ateliers flexibles. D'autre part, la recherche arborescente de Brucker a été la cible d'un intérêt constant malgré certains défauts notamment sa complexité.

Nous nous concentrerons donc sur le problème d'open-shop de base $O||C_{max}$, même si notre méthode reste valide pour de nombreux autres problèmes de minimisation du temps d'achèvement maximal sans contraintes de partage de ressource supplémentaires ($J||C_{max}$, $F||C_{max}$, $\{O, J, F\}|r_j; d_j; prec|C_{max}$). Notre principal axe de recherche concerne la conception et l'évaluation d'un algorithme pour calculer un arbitrage complet du graphe disjonctif en suivant la procédure top-down qui intègre des techniques de diversification et d'apprentissage. Dans un second temps, nous étudierons les heuristiques de sélection pour l'arbitrage du graphe disjonctif. Nous proposerons de nouvelles heuristiques basées sur les degrés des variables et discuterons de l'influence du modèle sur le comportement des heuristiques d'arbitrage. Finalement, nous décrirons dans le chapitre 9 des techniques de reformulation et de déduction pour la construction automatique du graphe disjonctif à partir d'un modèle quelconque défini par l'utilisateur.