Méthodes hybrides pour l'ordonnancement disjonctif avec flexibilité de ressources et considération de robustesse

Carla Juvin

THÈSE

En vue de l'obtention du
DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Délivré par l'Université Toulouse 3 - Paul Sabatier

Présentée et soutenue par

Carla JUVIN

Le 4 octobre 2023

Méthodes hybrides pour l'ordonnancement disjonctif avec flexibilité de ressources et considération de robustesse

Ecole doctorale : **EDMITT - Ecole Doctorale Mathématiques, Informatique et Télécommunications de Toulouse**

Spécialité : **Informatique et Télécommunications**

Unité de recherche :
LAAS - Laboratoire d'Analyse et d'Architecture des Systèmes

Thèse dirigée par
Pierre LOPEZ et Laurent HOUSSIN

Jury

M. Boris DETIENNE, Rapporteur
M. Roel LEUS , Rapporteur
M. Stéphane DAUZERE-PERES, Examinateur
Mme Rosa FIGUEIREDO, Examinatrice
M. Pierre LOPEZ, Directeur de thèse
M. Laurent HOUSSIN, Co-directeur de thèse

Remerciements

Tout d'abord, je souhaite remercier les rapporteurs de cette thèse, Roel LEUS et Boris DETIENNE, pour leur expertise et leurs commentaires, qui ont contribué à l'amélioration de ce manuscrit. Je tiens également à remercier les membres du jury, Stéphane DAUZERE-PEREZ et Rosa FIGUEIREDO, pour l'intérêt qu'ils ont porté à mon travail et pour les échanges pertinents lors de la soutenance.

Ensuite, je tiens à remercier mes directeurs de thèse, Pierre LOPEZ et Laurent HOUSSIN. Leur implication, leur disponibilité, leurs conseils, ainsi que leur bienveillance, ont grandement contribué à la réussite de ce projet.

Je remercie également tous les membres de l'équipe ROC, pour leur bonne humeur, leur gentillesse, ainsi que pour les moments de convivialité partagés.

Enfin, merci à ma famille et à mes amis pour leurs encouragements et pour les moments de bonheur passés ensemble, en particulier à Sébastien, pour sa patience et son soutien au quotidien.

Méthodes hybrides pour l'ordonnancement disjonctif avec flexibilité de ressources et considération de robustesse

Résumé : Dans cette thèse, nous étudions les problèmes d'ordonnancement disjonctif, en examinant deux types spécifiques de problèmes : ceux impliquant la flexibilité des ressources et ceux dont les durées des tâches sont entachées d'incertitude. Nous proposons des approches hybrides qui combinent différentes méthodes de résolution, exploitant ainsi les avantages de chacune d'elles.

Dans un premier temps, nous étudions le problème de job-shop flexible, un problème d'atelier qui consiste à déterminer sur quelle machine et dans quel ordre doivent être traitées les opérations. Nous proposons un schéma de décomposition du problème, ainsi qu'une méthode de décomposition de Benders basée sur la logique pour le résoudre. Nous comparons cette approche avec des méthodes de résolution basées sur des formulations directes pour les versions non préemptive et préemptive du problème.

Dans un second temps, nous nous intéressons à plusieurs problèmes d'ordonnancement robuste dans lesquels les durées des tâches sont incertaines. Nous nous concentrons sur la résolution de problèmes à deux étapes, dont l'objectif est de fixer la séquence des opérations sur chaque machine, tout en permettant aux dates de traitement des tâches de s'adapter à leurs durées effectives. Nous proposons différentes formulations directes des problèmes, ainsi que des méthodes de décomposition de Benders basées sur la logique et de génération de colonnes et de contraintes. Des résultats numériques sont présentés afin d'évaluer l'efficacité de chacune de ces méthodes.

Mots clés : Ordonnancement disjonctif, programmation mathématique, programmation par contraintes, robustesse, recherche arborescente

Hybrid Methods for Disjunctive Scheduling with Resource Flexibility and Robustness Consideration

Abstract : In this thesis, we study disjunctive scheduling problems, focusing on two specific types of problems: those involving resource flexibility and those where task durations are subject to uncertainty. We propose hybrid approaches that combine different solution methods, exploiting the advantages of each of them.

First, we study the flexible job-shop scheduling problem, where for each task there exists a set of eligible machines. We propose a decomposition scheme of the problem as well as a logic-based Benders decomposition method to solve it. We compare this approach with resolutions based on a direct formulation for the non-preemptive and preemptive versions of the problem.

Second, we consider several robust scheduling problems in which task durations are uncertain. We focus on solving two-stage problems, the objective of which is to determine the sequence of operations on each machine while allowing the processing date of the tasks to be adapted to their real durations. We propose different direct formulations of the problem as well as logic-based Benders decomposition and column and constraint generation methods. Numerical results are presented to illustrate the effectiveness of each of these methods.

Keywords : Disjunctive scheduling, mathematical programming, constraint programming, robustness, tree search

Table des matières

Table des figures

Liste des tableaux

Liste des algorithmes

Notations

$\mathcal{J}$	Ensemble de travaux
$\mathcal{M}$	Ensemble de machines
n_i	Nombre d'opérations du travail i
n	Nombre total d'opérations
$O_{i,j}$	$j^{ième}$ opération du travail i
$\mathcal{M}_i$	Ensemble de machines éligibles au traitement de la tâche i
$\mathcal{I}_m$	Ensemble de tâches pouvant être traitées par la machine m
p_i	Durée de traitement de la tâche i
r_i	Date de disponibilité de la tâche i
d_i	Date d'échéance de la tâche i
q_i	Durée de latence de la tâche i
t_i	Date de début de traitement de la tâche i
C_i	Date de fin de traitement de la tâche i
$C_{\max}$	Makespan
$\mathcal{U}$	Ensemble d'incertitude
ξ	Scénario
$\bar{\rho}$	Valeur nominale du paramètre incertain ρ
$\hat{\rho}$	Déviation maximale du paramètre incertain ρ

JSSP	Problème de job-shop
FJSSP	Problème de job-shop flexible
PLNE	Programmation linéaire en nombres entiers
PPC	Programmation par contraintes
LBBD	Décomposition de Benders basée sur la logique
BI	Borne inférieure
BS	Borne supérieure

Introduction

Contexte

Parmi les différents problèmes d'optimisation combinatoire, les problèmes d'ordonnancement occupent une place importante et se présentent sous diverses formes en fonction de leurs caractéristiques. Peu de ces problèmes sont aisés à résoudre et il est souvent indispensable de mettre en oeuvre des méthodes de résolution avancées, issues de la programmation mathématique, pour déterminer un ordonnancement optimal, quand la taille de problème le permet. Les problèmes d'ordonnancement dits "disjonctifs" regroupent une famille de problèmes où chaque tâche nécessite une ou plusieurs ressources de manière exclusive pendant toute sa durée d'exécution. Ainsi, toute autre tâche nécessitant également ces ressources ne peut être exécutée simultanément avec cette tâche. Ces modèles sont particulièrement pertinents pour représenter de nombreux problèmes pratiques tels que la gestion des machines-outils, la gestion de projets avec des ressources humaines ou encore les systèmes informatiques avec des processeurs.

Dans de nombreux contextes réels, les données d'entrée pour l'ordonnancement sont entachées d'incertitude. Des variations imprévues peuvent survenir dans les temps de traitement des tâches ou la disponibilité des ressources. La prise en compte de l'incertitude est donc essentielle pour assurer la robustesse d'un plan d'ordonnancement. Ainsi, dans le cadre d'une approche proactive, un ordonnancement robuste peut faire face aux perturbations en anticipant les éventuels scénarios défavorables.

Dans cette thèse, nous nous concentrons sur le développement de méthodes hybrides pour l'ordonnancement. Notre objectif est de proposer des approches combinant différentes techniques de résolution afin d'améliorer l'efficacité des méthodes existantes pour résoudre les problèmes d'ordonnancement déterministes ou soumis à l'incertitude.

Organisation du manuscrit

Ce manuscrit est composé de trois parties. Dans la partie I, le chapitre 1 présente les définitions de base et l'état de l'art dans le domaine de l'ordonnancement. Nous décrivons les différents problèmes d'ordonnancement, en mettant l'accent sur les problèmes d'ordonnancement d'atelier. Nous passons ensuite en revue les outils et les méthodes de résolution couramment utilisés dans la littérature. Enfin, nous examinons les instances de la littérature pour les problèmes de flow-shop, de job-shop et de job-shop flexible. Dans le chapitre 2, nous nous concentrons sur l'ordonnancement robuste, qui vise à trouver des solutions résistantes aux variations des données. Nous présentons dans un premier temps les concepts clés de l'optimisation robuste. Nous en décrivons ensuite les méthodes de résolution. Enfin, nous

passons en revue la littérature relative à l'ordonnancement robuste et proposons une définition formelle des problèmes étudiés dans cette thèse.

Dans la partie II, nous proposons des méthodes de résolution exactes pour les problèmes de job-shop flexible. Nous commençons, dans le chapitre 3, par présenter une approche de décomposition de Benders basée sur la logique, qui permet de décomposer le problème. Nous décrivons, en particulier, une formulation du problème maître, des relaxations du sous-problème, ainsi que des coupes de Benders logiques. Ensuite, dans les chapitres 4 et 5 nous abordons respectivement les problèmes de job-shop flexible non préemptif et préemptif. Pour chacun d'entre eux, nous présentons des formulations directes du problème, ainsi que des méthodes de résolution du sous-problème dans le cadre d'une approche utilisant l'algorithme de décomposition de Benders basée sur la logique. Des expérimentations numériques sont menées pour évaluer ces méthodes. Ces travaux ont été présentés dans les conférences *EURO 2021*, *ROADEF 2022* [Juvin 2022a] et *PMS 2022* [Juvin 2022b]. Ils ont également fait l'objet d'une publication dans le journal *Computers and Operations Research* [Juvin 2023f].

Dans la partie III, nous examinons différentes méthodes de résolution pour les problèmes d'ordonnancement robuste. Nous présentons différentes formulations des problèmes et proposons des méthodes de résolution exactes adaptées. Nous évaluons l'ensemble de ces méthodes en effectuant une analyse comparative approfondie de ces approches. Dans le chapitre 6, nous nous concentrons sur le problème de job-shop robuste et nous montrons comment l'ensemble de ces méthodes peut être adapté à la résolution du problème de job-shop flexible. Dans le chapitre 7, notre attention se porte sur le problème de flow-shop de permutation. Nous nous intéressons aussi à la complexité de ce problème dans le cas particulier du problème à deux machines. Cette partie a fait l'objet de publications dans les conférences internationales avec actes *ICORES 2023* [Juvin 2023c] et *CPAIOR 2023* [Juvin 2023e] ainsi qu'une présentation lors de la conférence *ROADEF 2023* [Juvin 2023b].

Le manuscrit se termine par des conclusions générales et les perspectives de ce travail.

Première partie

Définitions et état de l'art

Chapitre 1

Ordonnancement

Sommaire

L'objectif de ce chapitre n'est pas de présenter une revue exhaustive de la littérature, mais plutôt de fournir les connaissances nécessaires pour comprendre les chapitres suivants de ce travail.

1.1 Introduction

L'ordonnancement constitue une branche de l'optimisation combinatoire, visant à allouer de manière efficace les ressources disponibles dans le temps pour l'exécution de différentes tâches [Pinedo 2012].

Dans le contexte de ce document, le terme **tâche** est utilisé comme un terme générique pour désigner une entité élémentaire d'activité dans un problème d'ordonnancement. On distingue deux types de tâches : le **travail** et l'**opération**.

Un **travail** représente l'ensemble des traitements requis pour un même produit, pouvant être exécutés sur plusieurs ressources. Une **opération**, quant à elle, est une entité de traitement spécifique qui est exécutée sur une seule ressource. Un travail

peut donc être composé de plusieurs opérations. Les opérations d'un même travail sont liées entre elles par des contraintes temporelles.

Une **ressource** est un moyen de production nécessaire pour l'exécution des tâches. Les ressources peuvent être classées en fonction de leur nature **renouvelable** ou **consommable**. Une ressource renouvelable est une ressource qui est de nouveau disponible en même quantité après avoir été utilisée par une tâche, c'est le cas par exemple des opérateurs, des machines ou encore des processeurs. Au contraire, une ressource est dite consommable si elle est disponible en quantité limitée au cours du temps et qui diminue après avoir été utilisée par une tâche, c'est le cas par exemple pour un budget ou pour des matières premières. Par ailleurs, les ressources peuvent être **disjonctives** ou **cumulatives**. Une ressource disjonctive est une ressource qui ne peut être utilisée que par une seule tâche à la fois. En revanche, une ressource cumulative est une ressource qui peut être utilisée simultanément par plusieurs tâches, elle peut cependant avoir une capacité limitée. Dans cette thèse, nous nous concentrons sur les ressources renouvelables et disjonctives, que nous appelons **machines**. Cependant, l'ensemble des concepts et des méthodes présentés dans cette thèse peuvent également être appliqués à des ressources humaines, appelées **opérateurs**.

Dans cette thèse, on se concentre plus spécifiquement sur les problèmes d'ordonnancement d'atelier. Dans la suite de ce chapitre, nous définissons, dans un premier temps, les différents types de problèmes d'atelier. Nous nous intéressons ensuite aux outils et méthodes de résolution pour les problèmes d'ordonnancement. Enfin, nous introduisons les jeux d'instances de la littérature pour les problèmes d'atelier étudiés.

1.2 Problèmes d'ordonnancement

Nous considérons l'ordonnancement de travaux sur des machines. Cela consiste à l'allocation d'un ou plusieurs intervalles de temps de production sur une ou plusieurs machines pour chaque travail. Chaque machine est considérée comme une ressource disjonctive, ce qui signifie qu'elle ne peut exécuter qu'une seule tâche à la fois. En outre, un travail peut être composé de plusieurs opérations. Ces opérations peuvent être soumises à des contraintes mettant en jeu le temps et la quantité de ressources disponible. Un ordonnancement est optimal s'il minimise (ou maximise) un critère donné.

Formellement, un problème est défini par un ensemble de machines, noté $\mathcal{M}$, devant traiter un ensemble de travaux, noté $\mathcal{J}$. Chaque travail $i \in \mathcal{J}$ peut être composé de plusieurs étapes appelées opérations. Dans ce cas, on note n_i le nombre d'opérations dans le travail i et $O_{i,j}$ la $j^{ième}$ opération du travail i. De plus, n représente le nombre total d'opérations dans le problème $n = \sum_{i \in \mathcal{J}} n_i$.

La durée de traitement d'une tâche i est notée p_i dans le cas d'une opération unique, $p_{i,j}$, $p_{i,m}$ ou $p_{i,j,m}$ si la durée dépend aussi de la position de l'opération $j \in i \dots n_i$ dans le travail i et/ou de la machine $m \in \mathcal{M}$ sur laquelle elle est traitée.

En ordonnancement, l'objectif est de déterminer le moment où une tâche est exécutée. On utilise généralement la notation t_i (ou $t_{i,j}$) pour représenter la date de début d'une tâche et C_i (ou $C_{i,j}$) pour représenter sa date de fin.

La notation de Graham [Graham 1979], également connue sous le nom de notation à trois champs $\alpha|\beta|\gamma$, est une représentation couramment utilisée pour classifier les problèmes d'ordonnancement. La composante α permet de spécifier l'environnement machine, la composante β représente les contraintes sur les tâches et la composante γ la fonction objectif ou critère à optimiser.

Nous introduisons les problèmes étudiés selon ces trois champs dans la suite du paragraphe. Cependant, cette notation ne sera pas utilisée dans ce document.

1.2.1 Environnement machine

L'environnement machine est défini par le type de ressource disponible et le type de cheminement devant être effectué pour le traitement des travaux. Nous introduisons ici les problèmes rencontrés dans la suite de cette thèse.

1.2.1.1 Problème à machine unique

Le problème à machine unique est le problème d'ordonnancement le plus simple. Dans ce cas, un ensemble de tâches doivent être planifiées et exécutées sur une seule machine $\mathcal{M} = \{M_1\}$. Bien que le problème à machine unique ne soit pas l'un des problèmes spécifiquement étudiés dans le cadre de cette thèse, il est parfois utilisé comme un cas particulier pour obtenir une relaxation des problèmes d'atelier plus complexes.

1.2.1.2 Flow-shop

Le flow-shop est un type de problème d'atelier où les travaux doivent suivre un cheminement unique à travers les machines. Cela signifie que chaque travail doit passer par les différentes machines dans un ordre fixe et identique pour tous les travaux.

Dans le cadre de cette thèse, nous nous intéressons plus particulièrement au flow-shop de permutation. Dans ce cas, les travaux doivent être exécutés dans le même ordre sur chaque machine.

1.2.1.3 Job-shop

Le job-shop (JSSP pour *job-shop scheduling problem*) est un autre type de problème d'atelier qui se distingue par un cheminement multiple des travaux à travers les machines. Cela signifie que chaque travail suit un ordre spécifique sur les différentes machines, mais cet ordre peut varier d'un travail à l'autre.

1.2.1.4 Job-shop flexible

Le problème de job-shop flexible (*flexible job-shop scheduling problem*, FJSSP) est une généralisation du job-shop dans le sens où chaque opération peut être réalisée sur un ensemble de machines et il faut décider quelle machine allouer à chaque opération.

Il existe d'autres types de problèmes d'atelier tels que l'open-shop, pour lequel l'ordre des opérations dans un travail n'est pas fixé. Ce type de problème ne sera pas traité dans cette thèse.

En plus de l'environnement machine, le problème peut être défini en fonction des caractéristiques spécifiques des travaux. Celles considérées dans ce document sont définies dans le paragraphe suivant.

1.2.2 Caractéristiques des travaux

Dans le contexte de ces problèmes d'ordonnancement, il est possible de prendre en compte diverses caractéristiques supplémentaires. Dans ce paragraphe, nous introduisons uniquement celles qui sont pertinentes pour les problèmes étudiés.

Il peut s'agir de **contraintes temporelles** telles que des **dates de disponibilité**, des **dates d'échéance** et/ou des **durées de latence**. La date de disponibilité d'une tâche i est le moment à partir duquel cette tâche peut être traitée. Elle est notée r_i. La contrainte $t_i \geq r_i$ doit alors être satisfaite. La date d'échéance, notée d_i, d'une tâche est la date à laquelle le traitement de celle-ci doit être terminé. Il peut s'agir d'une contrainte stricte, quand cette échéance ne peut être dépassée ($C_i \leq d_i$), ou d'une contrainte souple, on cherchera dans ce cas à minimiser les retards donnés par $T_i = \max(0, C_i - d_i)$. La durée de latence q_i d'une tâche i représente le délai ou l'attente, après la fin de l'exécution de la tâche avant que celle-ci puisse être considérée comme terminée. La tâche ne consomme aucune ressource durant ce délai.

Dans certains cas, une tâche peut être morcelable, ce qui signifie qu'elle peut être interrompue et reprise ultérieurement. Cela correspond à un problème **préemptif**, où l'interruption d'une tâche est possible pour donner la priorité à une autre tâche sur la même ressource. En revanche, dans les problèmes **non préemptifs**, les tâches ne peuvent pas être interrompues et doivent être exécutées de manière continue.

Dans un problème de job-shop, la notion de **recirculation** se réfère à la possibilité pour un travail de passer plusieurs fois par la même machine. Cela signifie qu'un même travail peut avoir plusieurs opérations exécutées sur une même machine. Par conséquent, le nombre d'opérations pour un travail peut dépasser le nombre de machines disponibles.

1.2.3 Critère d'optimisation

Le critère d'optimisation est utilisé pour évaluer et comparer différentes solutions dans un problème d'optimisation. Il permet de déterminer la qualité d'une solution

par rapport à une autre. Plusieurs critères existent dans la littérature.

L'un des plus courants, et celui qui est considéré dans l'ensemble de nos contributions, est le makespan, c'est-à-dire le temps total entre le début et la fin de la production. Il est noté $C_{\max}$.

Il existe cependant d'autres critères, qui ne seront pas pris en compte ici tels que la somme des dates de fin des travaux, notée $\sum_{i\in\mathcal{J}} C_i$. Si les travaux possèdent une date d'échéance, il est aussi possible de considérer la somme des retards, notée $\sum_{i\in\mathcal{J}} T_i$ ou encore le retard maximal $T_{\max}$. Ces critères, appelés **critères réguliers**, sont des fonctions décroissantes des dates de fin des opérations. Cependant, il existe également des **critères irréguliers** qui ne suivent pas cette tendance. Par exemple, la charge de traitement sur les machines est indépendante des dates d'exécution des tâches.

L'optimisation de la **fonction objectif** d'un problème d'ordonnancement est la minimisation, ou la maximisation, de l'un, ou de la combinaison de plusieurs, de ces critères.

Pour l'ensemble des problèmes étudiés dans cette thèse, nous considérons la minimisation du makespan $C_{\max}$. On s'intéresse plus particulièrement au problème de job-shop flexible (dans l'ensemble de la partie II et dans le chapitre 6), de job-shop (dans le chapitre 6) et de flow-shop de permutation (dans le chapitre 7).

1.3 Outils et méthodes de résolution

Dans ce paragraphe, nous introduisons, dans un premier temps, des outils permettant de représenter un problème d'ordonnancement et/ou une solution à ce problème. Ensuite, nous présentons les méthodes de résolution utilisées dans la littérature. Nous proposons un aperçu, non exhaustif, de l'état de l'art de ces méthodes relatif aux problèmes étudiés.

1.3.1 Représentation

Au registre des outils utilisés pour représenter les problèmes d'ordonnancement, nous présentons ici le graphe disjonctif et le diagramme de Gantt.

1.3.1.1 Graphe disjonctif

Un problème d'ordonnancement peut être représenté par un graphe disjonctif $\mathcal{G} = (G, \mathcal{D})$ où $G = (\mathcal{N}, \mathcal{A})$ est un graphe conjonctif et $\mathcal{D}$ un ensemble de disjonctions [Roy 1964]. Dans le graphe conjonctif G, $\mathcal{N}$ est un ensemble de nœuds représentant les tâches du problème ainsi que deux tâches fictives *debut* et fin représentant respectivement le début et la fin de l'ordonnancement, $\mathcal{A}$ est l'ensemble des arcs conjonctifs, ils représentent les relations de précédence entre les opérations. On note $(u, v) \in \mathcal{A}$ un arc d'un nœud $u \in \mathcal{N}$ vers un noeud $v \in \mathcal{N}$, son coût est la durée de traitement de l'opération représentée par le nœud u (dans un formalisme

graphe potentiels-tâches). Il signifie que la tâche représentée par le nœud u doit être terminée avant que la tâche représentée par v puisse débuter, par exemple deux opérations successives d'un même travail. Un couple d'arcs conjonctifs $(u, v) \in \mathcal{D}$ et $(v, u) \in \mathcal{D}$ relient deux tâches qui ne peuvent être traitées simultanément, par exemple deux tâches devant être traitées par une même ressource disjonctive. Ces deux arcs sont mis en disjonction, cela signifie qu'ils ne peuvent pas coexister dans une solution.

Exemple 1 *Considérons une instance de job-shop avec 3 travaux et 2 machines. Les temps de traitement $p_{i,j}$ des opérations $O_{i,j}$, $\forall i \in \mathcal{J}$, $\forall j \in 1, \ldots, n_i$, sont donnés dans le tableau 1.1. On considère ici que la préemption n'est pas autorisée.*

		$M1$	$M2$
$J1$	$O_{1,1}$		7
	$O_{1,2}$	5	
$J2$	$O_{2,1}$	3	
	$O_{2,2}$		4
$J3$	$O_{3,1}$	10	
	$O_{3,2}$		10

TABLEAU 1.1 – Exemple numérique d'une instance de job-shop

La figure 1.1 représente le graphe disjonctif pour cette instance.

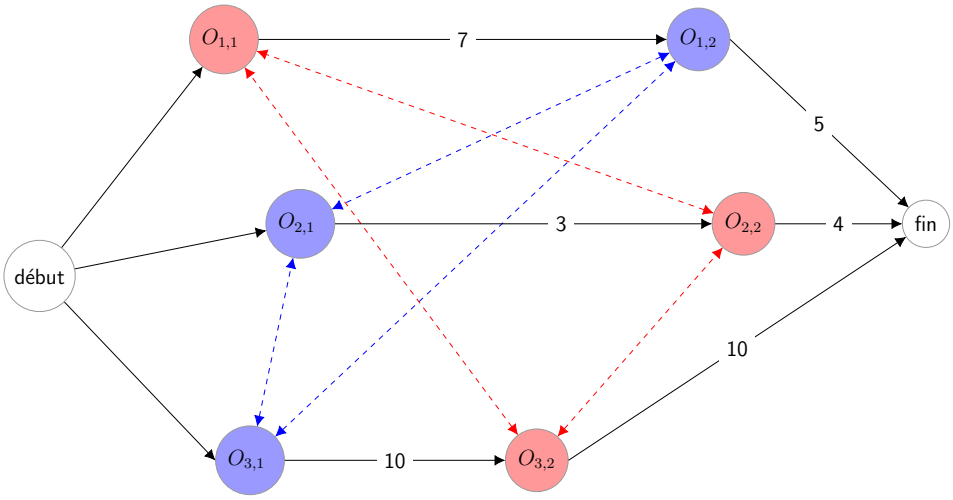


FIGURE 1.1 – Exemple 1 : graphe disjonctif $\mathcal{G}$ pour une instance de job-shop à 3 travaux et 2 machines

Les nœuds représentant les opérations d'un même travail sont sur une même ligne et sont liés par des arcs noirs représentant les relations de précédence induites par le problème. Des arcs noirs sont aussi présents entre le nœud debut et la première opération, ainsi qu'entre la dernière opération et le nœud fin pour chaque

travail. Chaque nœud est représenté en couleur, avec les nœuds bleus correspondant aux opérations qui doivent être traitées sur la machine M_1, et les nœuds rouges correspondant aux opérations qui doivent être traitées sur la machine M_2. Les nœuds de même couleur sont reliés par des arêtes en pointillés, qui représentent les contraintes disjonctives entre les tâches qui ne peuvent pas être exécutées simultanément sur la même ressource.

Une solution du problème correspond à une sélection complète d'arcs disjonctifs sans cycle. Une sélection est complète si un et un seul arc est sélectionné pour chaque disjonction du problème. Le makespan d'une solution correspond alors à la longueur du plus long chemin, du nœud *debut* au nœud fin, dans ce graphe.

Exemple 2 *Une solution complète et admissible pour le problème de l'exemple précédent est donnée par le graphe représenté dans la figure 1.2.*

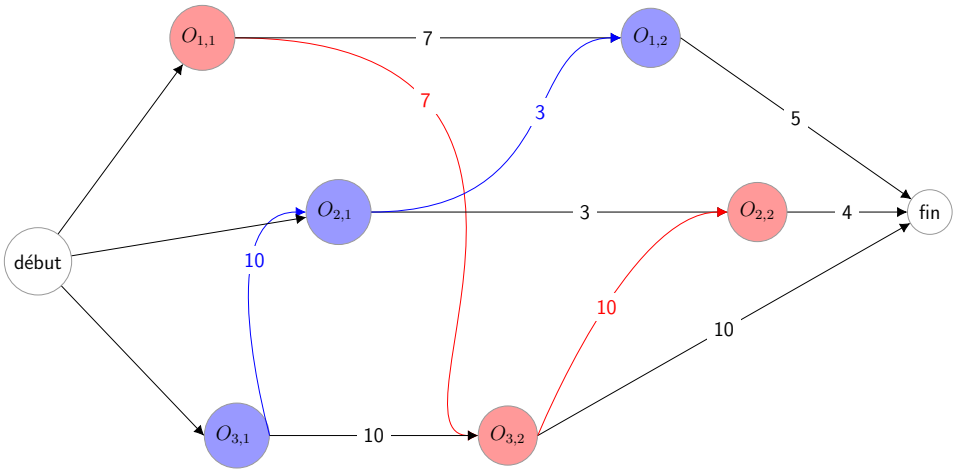


FIGURE 1.2 – Exemple 1 : solution admissible pour une instance de job-shop à 3 travaux et 2 machines ; représentation par un graphe conjonctif

Cette solution consiste à traiter les opérations $O_{3,1}$, $O_{2,1}$ puis $O_{1,2}$ dans cet ordre, sur la machine M_1, et les opérations $O_{1,1}$, $O_{3,2}$ puis $O_{2,2}$ dans cet ordre, sur la machine M_2.

1.3.1.2 Diagramme de Gantt

Un diagramme de Gantt est une représentation graphique d'une solution à un problème d'ordonnancement. Il utilise un axe horizontal pour représenter le temps et un axe vertical pour représenter les différentes ressources. Les tâches sont illustrées sous la forme de barres horizontales dont la longueur est proportionnelle la durée.

Exemple 3 *La solution donnée dans l'exemple précédent est représentée à l'aide d'un diagramme de Gantt dans la figure 1.3. Ce diagramme permet de visualiser les différentes tâches, les ressources qui les effectuent et la période pendant laquelle*

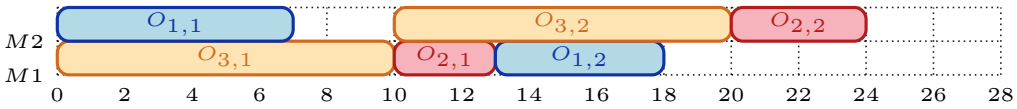


FIGURE 1.3 – Exemple 1 : solution admissible pour une instance de job-shop à 3 travaux et 2 machines ; représentation par un diagramme de Gantt

elles sont exécutées. Chaque travail est représenté par une couleur différente. De plus, il est facile de visualiser le respect des contraintes de précédence entre les opérations d'un même travail (par exemple l'opération $O_{3,1}$ termine avant le début de l'opération $O_{3,2}$) et de lire le makespan de la solution proposée, soit 24 dans cet exemple.

1.3.2 Méthodes de recherche arborescente

Les méthodes de recherche arborescente sont des techniques utilisées pour explorer l'ensemble des solutions admissibles d'un problème en utilisant une représentation des solutions sous forme d'arbre. Chaque nœud de l'arbre représente un sous-ensemble de solutions réalisables ou une solution partielle du problème.

La méthode débute avec un nœud racine qui représente l'ensemble des solutions du problème. Une opération de branchement est réalisée, elle consiste à partitionner cet ensemble en appliquant une décision à l'une des variables. On crée alors plusieurs branches correspondant à des contraintes différentes ajoutées au problème. Ces contraintes peuvent être l'affectation d'une valeur à la variable sélectionnée ou l'ajout d'inégalités pour la valeur de celle-ci. En répétant ce processus jusqu'à ce que l'ensemble des variables aient une affectation, on obtient un arbre dont chaque feuille représente une solution ou une inconsistance.

Il existe différentes stratégies pour choisir l'ordre dans lequel l'arbre des solutions est exploré. La recherche en profondeur d'abord explore les nœuds en profondeur, jusqu'aux feuilles avant de revenir en arrière pour explorer les autres branches de l'arbre. La recherche en largeur d'abord explore les nœuds de chaque niveau de l'arbre avant de passer aux niveaux suivants.

Un algorithme de séparation et évaluation, ou *branch-and-bound*, permet d'éviter d'énumérer l'ensemble des solutions d'un problème. Pour cela, une fonction d'évaluation est utilisée pour juger de la pertinence d'un sous-ensemble de solutions représenté par un nœud. Pour un problème de minimisation, elle peut être utilisée pour trouver une borne inférieure (BI), calculée à l'aide d'une relaxation du problème, c'est-à-dire une version simplifiée du problème avec certaines contraintes relâchées. Cette fonction permet d'élaguer le sous-arbre résultant, si la BI trouvée est trop grande par rapport à la valeur de la meilleure solution déjà explorée, évitant ainsi l'exploration de solutions non prometteuses. Cette méthode est par exemple utilisée pour résoudre le problème de flow-shop de permu-

tation [McMahon 1967, Carlier 1996]. Une telle méthode est aussi proposée dans [Giffler 1960] pour le problème de job-shop. De nombreuses approches arborescentes basées sur représentation du problème par un graphe disjonctif (présenté dans le paragraphe 1.3.1.1) sont proposées dans la littérature [Balas 1969, Carlier 1989].

Il est possible de créer des méthodes de recherche arborescente *ad hoc*, spécifiques pour des problèmes particuliers. Cependant, les techniques de recherche arborescente sont également utilisées dans la recherche de solutions à l'aide de la programmation linéaire en nombres entiers (PLNE) et de la programmation par contraintes (PPC) introduites dans les paragraphes suivants.

1.3.3 Programmation linéaire en nombres entiers

La programmation linéaire en nombres entiers (PLNE) [Wolsey 1998] est une approche couramment utilisée pour modéliser les problèmes d'ordonnancement. Elle permet de décrire le problème sous forme d'inéquations linéaires, où certaines variables peuvent prendre uniquement des valeurs entières.

1.3.3.1 Modélisation

Il existe trois grandes catégories de formulations PLNE pour les problèmes d'ordonnancement [Ku 2016]. La première regroupe les formulations basées sur des variables indexées sur le temps, comme, par exemple, des variables d'activation : $x_{i,t}$ égale à 1 si la tâche i est en cours de traitement à la date t [Bowman 1959], ou des variables de "pulse" : $x_{i,t}$ égale à 1 si la tâche i débute son traitement à la date t [Kondili 1988]. Le deuxième type de formulation est basé sur l'utilisation de variables de rang pour représenter l'ordre d'exécution des tâches sur les machines : $x_{i,l}$ égale à 1 si la tâche i est traité en $l^{ième}$ position [Wilson 1989, Wagner 1959]. Enfin, la troisième catégorie comprend les formulations basées sur les variables de précédence : $x_{i,i'}$ égale à 1 si la tâche i est traitée avant la tâche i' [Manne 1960, Liao 1992].

Le choix de la formulation peut avoir un impact important sur les performances de résolution. Chaque formulation a ses avantages selon le type de problème à résoudre. Les formulations indexées sur le temps sont particulièrement utiles pour les problèmes préemptifs, car elles permettent de modéliser facilement l'interruption des tâches. Cependant, pour les problèmes non préemptifs, elles peuvent être moins efficaces car elles nécessitent un grand nombre de variables ainsi que la définition d'un horizon de temps. En général dans ce cas, les formulations basées sur les rangs sont plus performantes pour les problèmes de flow-shop de permutation [Tseng 2004], tandis que les modèles basés sur les variables de précédence sont plus adaptés aux problèmes de job-shop [Ku 2016, Meng 2020].

1.3.3.2 Résolution

Lors de la résolution des problèmes de programmation linéaire en nombres entiers, une recherche arborescente est souvent utilisée. À chaque nœud, une borne

inférieure du problème est obtenue en effectuant une relaxation du problème, par exemple une relaxation continue, c'est-à-dire permettant aux variables de prendre des valeurs continues, au lieu de se limiter à des valeurs entières. La stratégie de branchement consiste alors à sélectionner une variable dont la valeur, dans la solution optimale du problème relaxé, n'est pas entière. Deux branches sont créées, l'une avec une contrainte spécifiant que la variable est inférieure ou égale à une valeur donnée, et l'autre avec une contrainte spécifiant que la variable est supérieure à cette valeur.

Dans cette thèse, les modèles PLNE proposés sont résolus à l'aide du solveur commercial *IBM CPLEX*.

1.3.4 Programmation par contraintes

La programmation par contraintes (PPC) [Rossi 2006] est une autre approche largement utilisée dans la résolution des problèmes d'optimisation combinatoire. Elle permet de décrire ces problèmes de manière intuitive et compréhensible, en décrivant les variables et les contraintes qui régissent leurs relations. La PPC s'appuie sur le formalisme des problèmes de satisfaction de contraintes, ou *Constraint Satisfaction Problems* (CSP), définis par un ensemble de variables, de domaines de valeurs dans lesquels s'inscrivent les variables et de contraintes définissant les relations entre les variables.

Dans cette thèse, les modèles PPC proposés sont résolus à l'aide du solveur commercial *IBM CP Optimizer* [Laborie 2018].

1.3.4.1 Modélisation

L'un des principaux avantages de la PPC est qu'elle permet d'utiliser des types de variables et de contraintes très variés. Par exemple, pour modéliser les problèmes d'ordonnancement, il est courant d'utiliser des variables d'intervalle qui spécifient la période pendant laquelle une tâche est en cours de traitement, mais dont la position dans le temps est une inconnue du problème. Ces variables peuvent également être optionnelles, ce qui permet de représenter des modes alternatifs d'exécution d'une tâche, par exemple sur différentes machines éligibles dans le problème de job-shop flexible.

Il est possible de contraindre les positions relatives des variables d'intervalle, par exemple avec la contrainte de précédence $EndBeforeStart$. Il est aussi possible de définir des contraintes portant sur un ensemble de variables, on les appelle des *contraintes globales*. Par exemple, les contraintes disjonctives, difficiles à exprimer à l'aide de la PLNE, peuvent être modélisées en PPC à l'aide de contraintes globales de non chevauchement $NoOverlap$.

Les variables de séquence d'intervalle sont un autre type de variables qui permettent de déterminer l'ordre relatif des intervalles qui les composent. Une variable de séquence est définie par un ensemble de variables d'intervalle. La contrainte $NoOverlap$ appliquée directement à la variable de séquence permet de garan-

tir que ces intervalles ne se chevauchent pas. De plus, en utilisant la contrainte *SameSequence*, il est aussi possible d'imposer le même ordre pour deux séquences différentes.

1.3.4.2 Résolution

La résolution d'un de programmation par contraintes problème consiste à trouver une affectation à toutes les variables qui satisfasse les contraintes. Pour cela, des techniques de *propagation de contraintes* sont combinées à une méthode d'exploration de l'espace de recherche telle que la recherche arborescente. La propagation de contraintes met en œuvre des mécanismes d'inférence afin de réduire l'espace de recherche. Cette propagation se base sur des algorithmes de filtrage, qui éliminent les valeurs incompatibles dans les domaines des variables.

En ordonnancement, les règles dites *edge finding* et *not-first/not-last* sont parmi les algorithmes de filtrage les plus connus pour les contraintes disjonctives. La règle *edge finding* est introduite dans [Carlier 1994]. Elle permet de déterminer si une tâche doit obligatoirement être ordonnancée avant ou après un ensemble de tâches. La règle *not-first/not-last* [Torres 2000] permet quant à elle de détecter les situations où il est impossible qu'une tâche soit ordonnancée avant ou après un ensemble de tâches partageant une même ressource.

De nombreux travaux de la littérature s'intéressent à l'utilisation de la PPC pour résoudre le JSSP [Erschler 1976, Nuijten 1998, Baptiste 1995]. Dans [Ku 2016], une comparaison des performances du solveur de PPC *CP Optimizer* par rapport à un ensemble de modèles PLNE est menée. Les résultats de l'étude concluent que la PPC surpasse les modèles PLNE, à la fois en termes de démonstration d'optimalité et de qualité des solutions obtenues. La même conclusion est apportée dans [Meng 2020] pour le FJSSP. De manière analogue, une étude comparative présentée dans [Naderi 2023] montre que le solveur *IBM CP Optimizer* surpasse les solveurs PLNE examinés pour une grande variété de problèmes d'ordonnancement, à l'exception des problèmes d'affectation pure.

1.3.5 Décomposition de Benders basée sur la logique

1.3.5.1 Principes de la décomposition de Benders

La décomposition de Benders [Benders 1962] est une approche utilisée pour résoudre des problèmes d'optimisation, qui exploite le fait que fixer temporairement la valeur de certaines variables simplifie grandement le problème. Elle décompose donc le problème en un problème maître et un ou plusieurs sous-problèmes, qui sont résolus de manière itérative en utilisant la solution de l'un dans l'autre. Le problème maître est une relaxation du problème original, il ne considère qu'une sous-partie des variables du problème et détermine leur valeur. Pour chaque affectation de ces variables, un sous-problème est créé dont la résolution permet d'obtenir une solution réalisable, ou une preuve d'infaisabilité, qui permet de générer des coupes. Ces coupes sont ensuite ajoutées au problème maître afin d'exclure les

solutions précédemment obtenues. Ces itérations sont répétées jusqu'à ce qu'une solution optimale au problème original soit obtenue. Une revue de la littérature [Rahmaniani 2017] présente l'état de l'art pour cette méthode de décomposition.

La décomposition de Benders a été initialement proposée pour une classe de problèmes pour lesquels, lorsque les variables entières sont fixées, le sous-problème résultant est un programme linéaire, ou non linéaire, continu, et pour lesquels il est possible d'utiliser le dual des sous-problèmes pour obtenir des coupes.

La décomposition de Benders basée sur la logique [Hooker 1995] étend cette approche à toute forme de sous-problème en généralisant la dualité de programmation linéaire à la notion de dualité inférée.

1.3.5.2 Dualité inférée

Une extension de la décomposition de Benders a été proposée en généralisant la notion de dualité de programmation linéaire. Cette approche, connue sous le nom de décomposition de Benders basée sur la logique, ou *Logic-Based Benders Decomposition* (LBBD), a été introduite par Hooker [Hooker 1995] dans le contexte de la vérification des circuits logiques et formalisée dans [Hooker 2000]. Les sous-problèmes peuvent alors prendre différentes formes. Il n'existe cependant pas de mécanisme standard pour générer les coupes. Celles-ci doivent donc être conçues pour chaque classe de problèmes, en analysant la structure du problème, ce qui donne lieu à des coupes dites logiques [Hooker 2007].

L'un des principaux avantages de cette méthode est qu'elle permet d'hybrider l'utilisation de la PLNE et de la PPC afin de tirer profit des avantages de chacune de ces méthodes, pour résoudre par exemple des problèmes d'allocation de ressources et d'ordonnancement. Souvent, le problème maître est alors résolu à l'aide de la PLNE, alors que la solution des sous-problèmes est obtenue grâce à la PPC.

L'efficacité des méthodes LBBD repose aussi en grande partie sur des techniques d'accélération, qui consistent à renforcer le problème maître par une relaxation du sous-problème et par la conception de coupes fortes [Hooker 2019, Karlsson 2022].

Cette approche a été appliquée avec succès à une grande variété de problèmes d'ordonnancement, en particulier ceux qui comportent, en outre, des décisions d'affectation. Dans [Hooker 2003], les auteurs s'intéressent à un problème à machines parallèles avec des dates de disponibilité et d'échéance avec l'objectif de minimiser les coûts de traitement. Les problèmes d'ordonnancement cumulatif sont abordés dans [Hooker 2007]. Le problème du flow-shop flexible est étudié dans [Tan 2018]. Dans [Li 2022], les auteurs se concentrent sur un problème combinant le choix de la localisation d'installations de machines et l'ordonnancement de tâches sur ces machines, tandis que le problème d'ordonnancement de machines parallèles avec temps de préparation dépendant de la séquence est traité dans [Tran 2016]. Un problème d'ordonnancement complexe dans un système électronique avionique est étudié dans [Karlsson 2022]. Dans [Sun 2019], les auteurs abordent un problème d'ordonnancement de grues qui effectuent le chargement et déchargement de conteneurs dans un contexte de transport maritime.

Une vaste littérature existe sur l'application de la LBBD pour résoudre divers problèmes d'optimisation combinatoire, au-delà des problèmes d'ordonnancement. Une étude approfondie de cette littérature est proposée dans [Hooker 2019].

1.3.6 Méthodes approchées

Une méthode approchée vise à obtenir une solution de bonne qualité en un temps raisonnable. Elle évite l'exploration exhaustive du domaine de recherche en exploitant les caractéristiques du problème. Elle ne garantit pas de trouver une solution optimale.

1.3.6.1 Règles de priorité

Une règle de priorité permet de déterminer un ordre de passage des tâches sur une même machine. Il existe diverses règles de priorité, plus au moins efficaces en fonction du problème étudié et de la fonction objectif. Dans [Sels 2012], les performances de diverses combinaisons de règles sont étudiées pour le problème de job-shop pour différents critères d'optimisation. La règle *FIFO* (First-In-First-Out) par exemple consiste à traiter les tâches dans l'ordre dans lequel elles deviennent disponibles. La règle *SPT* (Shortest Processing Time) consiste à traiter la tâche avec le plus petit temps de traitement en premier. Cette règle vise, en général, à minimiser le temps d'achèvement moyen. La règle *LPT* (Longest Processing Time), à l'inverse, consiste à traiter la tâche avec la plus grande durée en premier. La règle *MWR* (Most Work Remaining) donne la priorité aux opérations dont le temps de traitement restant sur le travail est le plus élevé, ce qui peut être utile pour équilibrer la charge de travail en vue de la minimisation du makespan par exemple. La règle *EDD* (Earliest due date) quant à elle consiste à traiter la tâche avec la plus petite date d'échéance en premier.

Dans le cas général, l'utilisation de règles de priorité ne présente aucune garantie sur l'optimalité des solutions obtenues. Cependant, il existe des problèmes pour lesquels de telles règles permettent d'obtenir une solution optimale du problème. C'est le cas par exemple de la règle de Johnson [Johnson 1954] pour le problème de flow-shop à deux machines pour la minimisation du makespan. Cette règle consiste à former deux groupes de travaux. Le premier groupe contient les travaux dont la durée de la première opération est inférieure à celle de la seconde opération, c'est-à-dire avec $p_{i,1} \leq p_{i,2}$. Le second groupe est constitué des travaux restants, c'est-à-dire avec $p_{i,1} > p_{i,2}$. Les travaux du premier groupe sont ordonnancés en premier, dans l'ordre croissant de la durée de la première opération $p_{i,1}$, en suivant la règle *SPT*, ensuite les travaux du second groupe sont ajoutés dans l'ordre décroissant de la durée de la deuxième opération $p_{i,2}$, en suivant la règle *LPT*. Cette règle peut, de manière équivalente, être décrite comme suit : le travail i doit être traité avant le travail i' si $\min(p_{i,1}, p_{i',2}) < \min(p_{i',1}, p_{i,2})$.

1.3.6.2 Métaheuristiques

Les métaheuristiques sont des approches qui offrent une méthodologie pour explorer de manière efficace l'espace de recherche sans recourir à une exploration exhaustive de toutes les solutions possibles. Elles peuvent utiliser des stratégies basées sur l'exploration d'un voisinage local pour améliorer progressivement une solution existante ou exploiter une population de solutions et s'inspirer de comportements naturels ou biologiques pour les faire évoluer.

La recherche locale permet d'explorer l'espace des solutions en effectuant des mouvements locaux à partir d'une solution initiale. L'idée est d'améliorer itérativement la solution en se déplaçant vers les voisins prometteurs dans l'espace de recherche, c'est-à-dire en modifiant un sous-ensemble restreint de décisions qui permettent d'améliorer la valeur de la fonction objectif. Afin d'éviter de rester bloqué dans un optimum local, des méthodes permettent d'accepter des mouvements dégradants tout en évitant de revisiter des solutions précédemment explorées. La recherche tabou [Glover 1989], par exemple, maintient en mémoire la liste des dernières solutions visitées. Elle est largement utilisée pour résoudre le problème de flow-shop de permutation [Reeves 1993, Nowicki 1996], de job-shop [Taillard 1994, Barnes 1995, Nowicki 2005] ou encore de job-shop flexible [Hurink 1994, Shen 2018] Le recuit simulé [Kirkpatrick 1983] quant à lui tire son inspiration d'un processus utilisé en métallurgie. Il simule le processus de refroidissement d'un matériau pour atteindre un état d'équilibre. L'algorithme explore l'espace de recherche en acceptant des solutions moins bonnes, avec une certaine probabilité, permettant ainsi d'éviter les optima locaux. Il est par exemple utilisé dans [Osman 1989, Zegordi 1995] pour résoudre le problème de flow-shop de permutation, dans [Van Laarhoven 1992, Kolonko 1999] pour le problème de job-shop ou dans [Najid 2002, Yazdani 2015] pour le problème de job-shop flexible.

Les algorithmes génétiques s'inspirent de la théorie de l'évolution. Ils utilisent des opérations telles que la sélection, le croisement et la mutation pour faire évoluer une population de solutions vers des solutions de meilleure qualité. Ils sont utilisés aussi bien pour résoudre des problèmes de job-shop [Della Croce 1995] que des problèmes de flow-shop de permutation [Reeves 1995, Chen 1995, Zobolas 2009, Etiler 2004].

D'autres exemples encore s'inspirent du comportement des animaux. Les algorithmes de colonies de fourmis s'inspirent du comportement des fourmis dans la recherche de nourriture. Les fourmis communiquent entre elles en déposant des phéromones sur le chemin qu'elles empruntent. Les algorithmes de colonies de fourmis utilisent une approche similaire pour guider l'exploration de l'espace de recherche. Les solutions prometteuses sont renforcées, ce qui favorise la convergence vers des solutions de qualité. Cette approche est utilisée par exemple pour résoudre le problème de flow-shop de permutation dans [Stützle 1998, Ying 2004, Rajendran 2004] ou de job-shop flexible dans [Rossi 2007].

Les essaims particulaires s'inspirent du comportement des groupes d'oiseaux ou de poissons dans leur déplacement collectif. Chaque individu de l'essaim représente

une solution potentielle, et se déplace dans l'espace de recherche en interagissant avec les autres. Les solutions sont alors ajustées en fonction des performances individuelles et des meilleures solutions trouvées par l'ensemble du groupe. L'optimisation par essaims particulaires est, par exemple, utilisée pour résoudre le problème de flow-shop de permutation dans [Tasgetiren 2004, Lian 2008], de job-shop dans [Lin 2010] ou encore de job-shop flexible multi-objectif dans [Huang 2016].

1.4 Instances de la littérature

Dans le cadre de nos expérimentations numériques portant sur les différents problèmes d'ordonnancement étudiés, nous utilisons des jeux d'instances classiques de la littérature. Les détails concernant ces jeux d'instances sont présentés dans la suite de ce paragraphe.

1.4.1 Flow-shop de permutation

Les instances de flow-shop de permutation les plus connues ont été introduites dans [Taillard 1993], afin de proposer une base de comparaison commune pour l'ensemble des méthodes développées dans la littérature. Le jeu d'instances est composé de 120 éléments, dont le nombre de travaux varie entre 20 et 500, et le nombre de machines entre 5 et 20.

1.4.2 Job-shop

Les instances de référence pour le problème de job-shop ont diverses origines [Fisher 1963, Lawrence 1984a, Adams 1988, Applegate 1991, Storer 1992]. Leurs caractéristiques sont résumées dans le tableau 1.2. Pour chaque jeu d'instances, nous indiquons le nombre d'instances (63 au total) qui le composent, la taille de ces instances (nombre de travaux × nombre de machines) ainsi que les intervalles à partir desquels les temps de traitement sont générés.

Jeu d'instances	Référence	Nombre d'instances	Taille	Durée de traitement
`ft6,10,20`	[Fisher 1963]	3	6×6, 10×10, 20×20	[1,10], [1,99]
`la01-40`	[Lawrence 1984a]	40	10× 5, 15× 5, 20× 5, 10×10, 15×10, 20×10, 30×10, 15×15	[5,99]
`abz5-9`	[Adams 1988]	5	10×10, 20×15	[50,100], [25,100], [11,40]
`orb1-10`	[Applegate 1991]	10	10×10	[5,99]
`swv16-20`	[Storer 1992]	5	50×10	[1,100]

TABLEAU 1.2 – Caractéristiques des instances de job-shop de la littérature

Des informations détaillées sur ces instances sont présentées dans [Jain 1999].

1.4.3 Job-shop flexible

Les instances classiques du problème de job-shop flexible sont présentées dans [Behnke 2018]. Celles proposées par Brandimarte [Brandimarte 1993], Hurink et al. [Hurink 1994], Dauzère-Pérès and Paulli [Dauzère-Pérès 1997], Chambers and Barnes [Barnes 1996] et Fattahi et al. [Fattahi 2007] sont parmi les plus utilisées pour l'évaluation des approches de résolution du FJSSP. Elles sont accessibles dans la collection *OR-LIBRARY* [Beasley 1990, Mastrolilli 1998] qui contient un ensemble de données pour des problèmes de recherche opérationnelle.

Dans [Brandimarte 1993], les données ont été générées de manière aléatoire en utilisant une distribution uniforme entre des bornes données. Le benchmark est composé de dix instances du problème dont le nombre de travaux varie entre 10 et 20. Le nombre de machines varie entre 6 et 15 et le nombre d'opérations par travail entre 5 et 15. Le nombre maximum de machines éligibles par opération varie entre 2 et 6.

Dans [Hurink 1994], les instances ont été générées à partir d'instances du problème de job-shop classique, trois tirées de [Fisher 1963] (*ft06, ft10, ft20)* et 40 de [Lawrence 1984b] (*la01-la40*). Chaque ensemble $M_{i,j}$ est composé de la machine à laquelle l'opération $O_{i,j}$ est affectée dans le problème original, ainsi que certaines autres machines avec une probabilité donnée. Trois ensembles d'instances ont ainsi été construits : *edata, rdata et vdata*, avec, dans l'ordre, des niveaux de flexibilité croissants.

Dans [Dauzère-Pérès 1997], le jeu d'instances est composé de dix-huit instances du problème dont le nombre de travaux varie entre 10 et 20. Le nombre de machines varie entre cinq et dix et le nombre d'opérations par travail entre 5 et 25. L'ensemble de machines éligibles pour chaque opération est construit en y ajoutant une machine avec une probabilité comprise entre 0,1 et 0,5.

Dans [Barnes 1996], les instances ont été construites à partir de trois instances du problème de job-shop classique, *ft10* introduite par [Fisher 1963] et *la24* et *la40* par [Lawrence 1984b]. Elles ont été transformées en instances de FJSP en dupliquant les machines, selon plusieurs politiques basées sur deux critères : le temps de traitement total nécessaire sur une machine et le nombre d'opérations critiques sur une machine. Le temps de traitement d'une opération sur une machine dupliquée est identique à l'original. Le jeu d'instances est constitué de vingt-et-une instances. Les instances notées *Kacem* sont introduites dans [Kacem 2002]. Ces instances sont totalement flexibles, cela signifie que chaque machine est capable de traiter chaque opération. Ce jeu d'instances se distingue aussi par sa petite taille, il est composé de seulement quatre instances. Dans [Fattahi 2007], le jeu d'instances est composé de dix instances de petite taille (*SFJS1-SFJS10*) et dix instances de taille moyenne (*MFJS1-MFJS10*).

Les caractéristiques de ces jeux d'instances sont résumées dans le tableau 1.3. Pour chaque jeu d'instances, nous indiquons le nombre d'instances qui le composent (276 au total), la taille de ces instances (nombre de travaux et nombre de machines), ainsi que le degré de flexibilité, défini comme le nombre moyen de machines éligibles

par opération.

Jeu d'instances	Référence	Nombre d'instances	Taille		Flexibilité
			Travaux	Machines	
BrandimarteMk	[Brandimarte 1993]	15	[10 . . .30]	[4 . . .15]	[1.4 . . .4.1]
HurinkEdata	[Hurink 1994]	66	[6 . . .30]	[4 . . .15]	[1.1 . . .1.2]
HurinkRdata	[Hurink 1994]	66	[6 . . .30]	[4 . . .15]	[1.9 . . .2.1]
HurinkVdata	[Hurink 1994]	66	[6 . . .30]	[4 . . .15]	[2.1 . . .6.7]
DPpaulli	[Dauzère-Pérès 1997]	18	[10 . . .20]	[5 . . .10]	[1.1 . . .5]
ChambersBarnes	[Barnes 1996]	21	[10 . . .15]	[11. . .18]	[1 . . .1.3]
Kacem	[Kacem 2002]	4	[4 . . .15]	[5 . . .10]	[5 . . .10]
Fattahi	[Fattahi 2007]	20	[2 . . .12]	[2 . . .8]	[1.5 . . .2.7]

TABLEAU 1.3 – Caractéristiques des instances de job-shop flexible

1.5 Conclusion

Ce chapitre nous a permis de poser les bases nécessaires à la compréhension des problèmes d'ordonnancement d'atelier et des méthodes de résolution associées. Nous avons introduit les différents problèmes d'ordonnancement d'atelier étudiés dans cette thèse, à savoir les problèmes de flow-shop de permutation, de job-shop et de job-shop flexible. Nous avons également exposé différentes méthodes de résolution adaptées à ces problèmes. Enfin, nous avons présenté les instances classiques de la littérature qui seront étudiées dans la suite de ce manuscrit pour évaluer les méthodes de résolution proposées.

Dans le chapitre suivant, nous abordons les concepts liés à l'ordonnancement robuste qui sont nécessaires à la compréhension de la partie III de ce document.

Chapitre 2

Ordonnancement robuste

Sommaire

2.1 Introduction

En pratique, lors de la résolution d'un problème d'ordonnancement, de nombreuses sources d'incertitudes existent, telles que la durée des opérations ou les pannes machines. Prendre ces incertitudes en compte est essentiel, car considérer une version déterministe d'un tel problème peut mener à une solution de très mauvaise qualité, voire infaisable [Ben-Tal 2000]. Pour tenir compte de ces incertitudes, on trouve principalement deux types d'approches. La première est l'optimisation stochastique. Elle repose sur l'utilisation de distributions de probabilités pour modéliser les valeurs incertaines des paramètres. Dans ce cadre, l'objectif est généralement d'optimiser des quantités statistiques, telles que l'espérance, afin d'obtenir une solution qui est bonne en moyenne sur l'ensemble des réalisations possibles des données incertaines. Cette approche peut être pertinente lorsque le processus à optimiser est répété de nombreuses fois et que les conséquences d'une performance médiocre pour une itération n'ont pas d'impact majeur. Elle ne permet pas, en général, de garantir qu'aucune des contraintes du problème ne sera violée pour l'ensemble des réalisations de l'incertitude.

Cependant, il existe des situations où les conséquences d'une mauvaise réalisation de l'incertitude sont plus importantes. De plus, il peut arriver que les probabilités de distribution des données incertaines ne soient pas disponibles. Dans de tels cas, l'optimisation robuste est préférée. L'optimisation robuste considère que les données incertaines appartiennent à un ensemble d'incertitude et vise à optimiser les performances en tenant compte du scénario dans le pire des cas, dit **scénario pire cas** de cet ensemble. L'objectif est donc de trouver une solution qui reste réalisable pour toutes les réalisations possibles des données incertaines, tout en offrant des performances satisfaisantes dans le pire des cas. Dans le cadre de cette thèse, c'est cette approche d'optimisation robuste qui est considérée. Une étude approfondie de la littérature relative à l'optimisation robuste est présentée dans [Ben-Tal 2009].

Dans ce chapitre, nous définissons dans un premier temps les termes relatifs à l'optimisation robuste et les différents types de problèmes rencontrés dans la littérature. Ensuite, nous nous intéressons aux méthodes de résolution couramment utilisées pour résoudre ces problèmes. Enfin, nous présentons plus particulièrement la littérature relative aux problèmes d'ordonnancement robuste.

2.2 Problème d'optimisation robuste

Nous considérons que l'**incertitude** concerne les données du problème. Les données peuvent alors être inexactes, incomplètes ou soumises à des aléas, c'est-à-dire susceptibles d'être modifiées par des événements imprévus. Pour traiter cette incertitude, nous utilisons le concept d'**ensemble d'incertitude**, qui regroupe toutes les configurations possibles des données incertaines. Chaque élément de cet ensemble $\xi \in \mathcal{U}$ est appelé un **scénario** ou une **réalisation de l'incertitude**, et représente des valeurs particulières pour chaque paramètre incertain.

Dans la suite, on note $\bar{\rho}$ la valeur nominale d'un paramètre incertain ρ, c'est-à-dire sa valeur habituelle ou attendue et $\hat{\rho}$ la déviation maximale par rapport à cette valeur, et on considère $\rho \in [\bar{\rho}, \bar{\rho} + \hat{\rho}]$. L'objectif de l'optimisation robuste est de déterminer la meilleure solution **pire cas**, c'est-à-dire une solution qui reste réalisable quelque soit le scénario dans $\mathcal{U}$ et qui optimise les performances dans le pire des scénarios (ce dernier dépendant lui même de la solution choisie).

La qualité d'une solution est généralement mesurée en termes de **performance pire cas**. Elle consiste à évaluer la performance d'une solution fixée pour l'ensemble des scénarios, et retenir la pire d'entre elles comme indicateur de robustesse de cette solution. Par exemple, en ordonnancement, si on considère la totalité des scénarios d'un ensemble d'incertitude, et le makespan obtenu pour chacun d'entre eux pour une solution fixée, alors le makespan pire cas de cette solution correspond au plus grand makespan parmi tous les scénarios. On parle alors de critère *min-max*. La performance peut également être évaluée en termes de **regret maximal**. Pour un scénario donné, le regret représente la différence entre la performance obtenue par la solution et celle de la meilleure solution possible pour ce scénario. Le regret maximal est donc le plus grand regret parmi tous les scénarios. Il mesure ainsi la

perte maximale par rapport à la meilleure performance réalisable. On parle alors de critère *min-max-regret*. Ce critère demande un effort calculatoire supplémentaire car il suppose de connaître la solution optimale pour chaque scénario ; il ne sera pas étudié ici. Dans le cadre de cette thèse, nous nous intéressons seulement au critère de performance pire cas.

2.2.1 Optimisation robuste statique

Dans le cadre de l'optimisation robuste statique, l'ensemble des décisions doivent être prises avant de connaître la réalisation de l'incertitude. Formellement, si on considère un problème d'optimisation

$$\min_{x \in X} f(x) \tag{2.1}$$

où f est la fonction objectif à minimiser et X est l'ensemble des solutions réalisables, la contrepartie robuste du problème peut être formulée comme suit :

$$\min_{x \in X} \max_{\xi \in \mathcal{U}} f(x, \xi) \tag{2.2}$$

avec $f(x, \xi) = +\infty$ si la solution x est irréalisable pour le scénario ξ. Pour une solution x^* fixée, le problème de maximisation $\max_{\xi \in \mathcal{U}} f(x^*, \xi)$ est appelé **problème adverse**. L'objectif est alors de trouver une solution x avec les meilleures performances pire cas, et réalisable quel que soit le scénario $\xi \in \mathcal{U}$.

Dans certains contextes, un tel modèle, à une seule étape, n'est pas réaliste et peut conduire à des solutions pessimistes. C'est pourquoi une méthode d'optimisation multi-niveau est étudiée.

2.2.2 Optimisation robuste multi-étapes

L'optimisation robuste à deux étapes est introduite dans [Ben-Tal 2004]. Ainsi, certaines décisions peuvent être prises lorsque les données incertaines sont connues et peuvent, de cette manière, être adaptées au scénario. Dans ce cas, on sépare les variables en deux ensembles, les variables qui doivent être déterminées avant la réalisation du scénario, appelées **décisions "*here-and-now*"** ou **décisions de premier niveau**, et celles qui peuvent être ajustées à la réalisation de l'incertitude, appelées **décisions "*wait-and-see*"**, **décisions de second niveau** ou **variables de recours**. On parle alors d'**optimisation robuste ajustable**. Un problème d'optimisation robuste ajustable peut être écrit sous la forme :

$$\min_{x \in X} \max_{\xi \in \mathcal{U}} \min_{y \in Y(x,\xi)} f(x, \xi, y) \tag{2.3}$$

où $Y(x, \xi)$ est l'ensemble des réalisations possibles pour les décisions de second niveau y, étant donné une solution de premier niveau $x \in X$ et un scénario $\xi \in \mathcal{U}$.

L'objectif de ce problème est donc de trouver une affectation des variables de premier niveau x telle qu'il existe des variables de second niveau y pour chaque scé-

nario $\xi \in \mathcal{U}$, permettant de satisfaire l'ensemble des contraintes, tout en minimisant la solution pire cas.

2.2.3 Ensemble d'incertitude

L'approche traditionnelle de l'optimisation robuste, telle qu'introduite dans [Soyster 1973], vise à se protéger contre le scénario le plus défavorable où tous les paramètres incertains peuvent prendre simultanément leur pire valeur. Par exemple, dans un **ensemble d'incertitude de type intervalle**, chaque paramètre incertain est donné sous la forme d'un intervalle de valeurs et toutes les combinaisons de ces valeurs sont réalisables. L'ensemble d'incertitude est alors le produit cartésien des intervalles définissant les valeurs possibles pour les variables incertaines $\mathcal{U} = \{\xi \in [0,1] \times [0,1] \times \dots\}$ avec $\rho_i = \bar{\rho}_i + \xi_i \hat{\rho}_i$.

Dans de nombreux cas, le problème peut alors être résolu avec un modèle déterministe, où chaque paramètre incertain prend sa valeur pire cas. Cependant, considérer ce type d'ensemble d'incertitude rend souvent la solution trop pessimiste, c'est-à-dire exagérément prudente, au détriment de la qualité de la fonction objectif, alors que ce scénario est éventuellement peu probable.

Pour surmonter cette limitation, il existe différents types d'ensemble d'incertitude. Le choix de celui-ci peut avoir des conséquences à la fois sur la complexité du problème robuste résultant, mais aussi sur le conservatisme de la solution obtenue et donc la qualité de la valeur de la fonction objectif ainsi que le niveau de protection contre la variation des paramètres incertains [Ben-Tal 1999].

Il est possible de considérer un **ensemble d'incertitude discret**, c'est-à-dire un ensemble fini de scénarios $\mathcal{U} = \{\xi^1, \dots, \xi^k\}$, dont chaque scénario est exprimé explicitement, en fonction des valeurs des paramètres incertains pour ce scénario. Ce type d'ensemble conduit souvent à des problèmes robustes difficilement traitables. Pour de nombreux problèmes d'optimisation combinatoire pouvant être résolus en temps polynomial dans leur version déterministe, par exemple le problème de flowshop de permutation à deux machines [Johnson 1954], leurs équivalents robustes sont *NP-difficiles*, même en considérant seulement deux scénarios [Kasperski 2012].

Un autre type d'ensemble d'incertitude considéré dans la littérature est l'**ensemble d'incertitude ellipsoïdal** [Ben-Tal 2000], qui vise à réduire le conservatisme en évitant les points extrêmes, c'est-à-dire lorsque toutes les variables prennent simultanément leur valeur extrême. Dans ce cas, l'ensemble d'incertitude est exprimé sous la forme :

$$\mathcal{U} = \left\{ (\xi_i)_{i \in 1,\dots,L} \mid \sqrt{\sum_{i=1}^{L} \xi_i^2} \leq \Omega,\ \xi_i \in [-1,1] \right\}$$

où ξ_i représente la déviation du paramètre ρ_i par rapport à sa valeur nominale pour une réalisation ξ donnée ; on a alors $\rho_i = \bar{\rho}_i + \xi_i \times \hat{\rho}_i$. Cette approche permet

de gérer le niveau de protection en faisant varier le paramètre Ω. Cependant, ce type d'ensemble conduit généralement à des problèmes non linéaires plus coûteux en termes de temps de calcul que les modèles linéaires.

Une autre approche permettant un compromis entre protection contre l'incertitude et qualité de la solution est l'utilisation d'un **ensemble d'incertitude discret**, défini par un **budget d'incertitude** [Bertsimas 2004]. Dans ce cas, on suppose qu'au plus $\Gamma \in \mathbb{N}$ paramètres peuvent s'écarter de leur valeur nominale. L'ensemble d'incertitude $\mathcal{U}$ est alors décrit comme suit :

$$\mathcal{U} = \Big\{(\xi_i)_{i\in 1,\dots,L} \mid \sum_{i=1}^{L} \xi_i \leq \Gamma,\ \xi_i \in \{0,1\}\Big\}.$$

Le modèle introduit dans [Soyster 1973] est un cas particulier du modèle d'incertitude de type budget avec $\Gamma = L$, L désignant le nombre de paramètres incertains. Le cas $\Gamma = 0$ correspond au problème déterministe, où l'ensemble des paramètres prennent leur valeur nominale. Cette approche a l'avantage de conduire à une formulation linéaire du problème robuste, comme décrit dans le paragraphe 2.3.2.

Ce type d'ensemble d'incertitude, qui repose sur un budget d'incertitude, est celui qui est étudié dans la suite de cette thèse.

2.3 Méthodes de résolution

Dans ce paragraphe nous présentons différentes méthodes de résolution d'un problème d'optimisation mettant en jeu des ensembles d'incertitude polyédraux.

2.3.1 Modèle étendu

Une approche naturelle pour résoudre un problème d'optimisation robuste consiste à formuler le problème déterministe à l'aide de techniques telles que la PLNE ou la PPC, et à dupliquer les contraintes pour chaque scénario $\xi \in \mathcal{U}$.

On note que si la fonction objectif est soumise à l'incertitude, il suffit d'introduire une nouvelle variable z et reformuler le problème (2.2) comme suit :

$$\min z \tag{2.4}$$

$$\text{t.q.} \quad z \geq f(x,\xi) \qquad \forall \xi \in \mathcal{U} \tag{2.5}$$

$$x \in X \tag{2.6}$$

Cependant, pour un ensemble d'incertitude de type budget, le nombre de scénarios croît de manière exponentielle avec la taille du problème. Cette méthode devient donc rapidement inapplicable en pratique.

2.3.2 Modèle compact

Lorsqu'un problème d'optimisation déterministe peut être formulé comme un PL (ou PLNE) et le problème adverse comme un autre PL, il est possible de formuler

le problème robuste à l'aide du modèle compact. Le principe du modèle compact est alors d'intégrer le dual du problème adverse dans la formulation du problème. Cette approche est basée sur le principe *"primal worst equals dual best"* introduit dans [Beck 2009].

Considérons une contrainte de la forme :

$$\sum_{i=1}^{L} \rho_i \times x_i \leq b \tag{2.7}$$

et un ensemble d'incertitude de type budget, avec au plus Γ paramètres incertains ρ_i qui dévient de leur valeur nominale. Une solution robuste x^* doit satisfaire la contrainte suivante :

$$\sum_{i=1}^{L} \bar{\rho_i} \times x_i^* + \max_{\xi \in \mathcal{U}} \{\sum_{i=1}^{L} \hat{\rho_i} \times x_i^* \times \xi_i\} \leq b. \tag{2.8}$$

Bertsimas et Sim [Bertsimas 2004] ont montré que cette contrainte peut être linéarisée en considérant le dual du problème adverse $\max_{\xi \in \mathcal{U}} \{\sum_{i=1}^{L} \hat{\rho_i} \times x_i^* \times \xi_i\}$. Ce dernier consiste à sélectionner le scénario ξ maximisant la partie gauche de l'inéquation. Pour une solution x^* fixée, il est équivalent à :

$$\max \sum_{i=1}^{L} \hat{\rho_i} \times x_i^* \times \xi_i \tag{2.9}$$

$$\text{t.q.} \quad \sum_{i=1}^{L} \xi_i \leq \Gamma \tag{2.10}$$

$$0 \leq \xi_i \leq 1 \qquad \forall i \in 1, \ldots, L \tag{2.11}$$

dont le dual peut être formulé comme suit :

$$\min \sum_{i=1}^{L} s_i + z \times \Gamma \tag{2.12}$$

$$\text{t.q.} \quad z + s_i \geq \hat{\rho_i} \times x_i^* \qquad \forall i \in 1, \ldots, L \tag{2.13}$$

$$s_i \geq 0 \qquad \forall i \in 1, \ldots, L \tag{2.14}$$

$$z \geq 0 \tag{2.15}$$

En substituant le problème de maximisation dans la contrainte (2.8) par son

dual, celle-ci peut être remplacée par :

$$\sum_{i=1}^{L} \bar{\rho}_i \times x_i^* + \min \sum_{i=1}^{L} s_i + z \times \Gamma \leq b \tag{2.16}$$

$$z + s_i \geq \hat{\rho}_i \times x_i^* \quad \forall i \in 1, \ldots, L \tag{2.17}$$

$$s_i \geq 0 \quad \forall i \in 1, \ldots, L \tag{2.18}$$

$$z \geq 0 \tag{2.19}$$

Dans [Bertsimas 2003], les mêmes auteurs appliquent une reformulation similaire pour des problèmes d'optimisation combinatoire.

2.3.3 Algorithme de génération de coupes

Une autre approche pour éviter d'avoir un grand nombre de contraintes découlant d'une formulation étendue du problème consiste à considérer une version relaxée du problème et à ajouter les contraintes *"à la volée"* (avec utilisation de *lazy constraints*), c'est-à-dire seulement si elles sont violées par l'une des solutions explorées. Cette méthode peut être assimilée à une décomposition de Benders, telle qu'elle est présentée dans le paragraphe 1.3.5.1.

Dans [Bertsimas 2016] une étude vise à comparer les performances d'une reformulation compacte d'un problème et sa résolution à l'aide de génération de coupes pour des problèmes statiques. Les auteurs montrent qu'aucune méthode ne domine l'autre pour l'ensemble des problèmes considérés.

2.3.4 Algorithme de génération de colonnes et de contraintes

La méthode de génération de colonnes et de contraintes a été introduite par Zeng et Zhao [Zeng 2013] pour résoudre les problèmes d'optimisation robuste à deux étapes. La procédure divise le problème en un problème maître et un sous-problème adverse. L'idée est de résoudre la contrepartie robuste (ou problème maître), pour un sous-ensemble limité de scénarios. Cela permet de fixer les variables de premier niveau. Un sous problème adverse permet d'identifier un scénario qui, le cas échéant, rend la solution trouvée dans le problème maître infaisable. Ensuite, ce scénario est inclus dans le problème maître en générant, à la volée, les variables de décision de recours correspondantes ainsi que les contraintes associées. Ce processus se répète jusqu'à ce qu'une solution réalisable pour tous les scénarios soit trouvée. La figure 2.1 illustre le schéma de l'algorithme de génération de colonnes et de contraintes.

Dans le pire des cas, tous les scénarios sont générés et cela revient à la formulation étendue présentée dans le paragraphe 2.3.1.

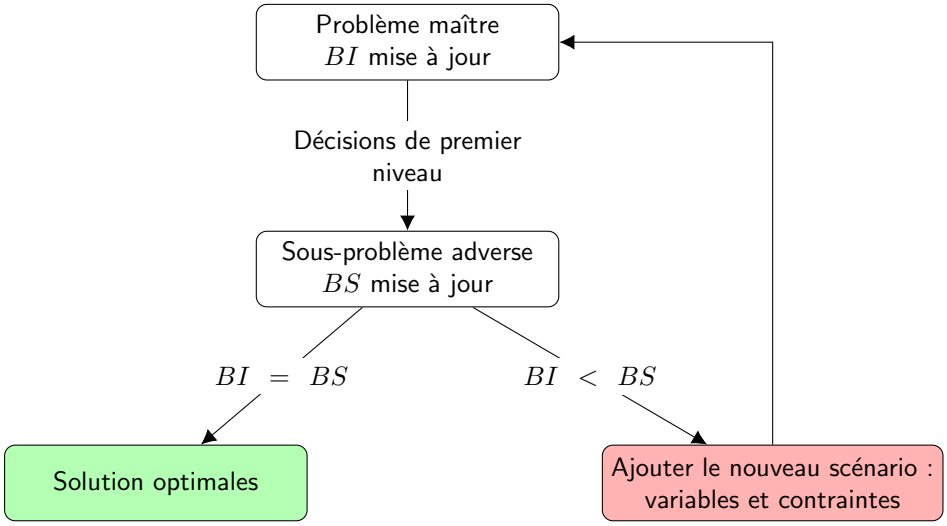


FIGURE 2.1 – Schéma de génération de colonnes et de contraintes (BI = borne inférieure, BS = borne supérieure)

2.4 Problèmes d'ordonnancement robuste

Dans ce paragraphe, nous examinons la littérature relative aux problèmes d'ordonnancement robuste. Ensuite, nous présentons les hypothèses qui sont prises en compte pour l'ensemble des problèmes d'ordonnancement robuste étudiés dans la suite de la thèse.

2.4.1 Littérature connexe

La robustesse d'une solution à un problème d'ordonnancement réside dans sa capacité à s'adapter face à l'incertitude. Elle peut se traduire à travers des modifications possibles dans l'ordonnancement de référence, telles que des changements d'affectation des opérations aux machines, des ajustements des dates de début et de fin des opérations, ou encore des modifications de l'ordre des opérations pour s'adapter aux différents scénarios.

Parmi les méthodes d'ordonnancement sous incertitude, on trouve des approches d'ordonnancement dynamique, où la séquence des opérations est déterminée en temps réel pendant l'exécution des tâches. Cela permet d'adapter l'ordonnancement en fonction de l'évolution des conditions et des événements imprévus [Subramaniam 2000]. Une autre approche consiste à remettre en cause la séquence des opérations prédéterminée à chaque fois qu'un événement inattendu survient. On parle alors de méthodes réactives, qui visent à calculer rapidement de nouvelles solutions [Van de Vonder 2007]. Toutefois, il n'est pas possible de garantir la valeur de la fonction objectif des solutions obtenues avec ces méthodes.

À l'inverse, il existe des méthodes proactives où l'ordonnancement est totalement

construit *a priori* en ajoutant des temps morts ou des marges pour amortir les déviations potentielles liées à l'incertitude [Jamili 2016]. Ces méthodes peuvent être assimilées à l'optimisation robuste statique, où aucune décision ne peut être ajustée à la réalisation de l'incertitude, ce qui peut entraîner une forte dégradation des performances.

Il existe également des méthodes permettant de pré-calculer un ensemble de solutions, ce qui permet de sélectionner la meilleure solution en fonction du scénario qui se présente [Ourari 2015].

Dans la suite de ce paragraphe, nous nous concentrons uniquement sur les approches visant à déterminer une séquence d'opérations unique pour chaque machine, et où les dates d'exécution des tâches peuvent s'adapter au scénario, de manière à minimiser le pire des cas en termes de critère d'optimisation étudié.

Les problèmes d'ordonnancement à une machine et à machines parallèles sont étudiés dans [Bougeret 2019], [Bougeret 2021] et [Bougeret 2023], considérant que les durées de traitement peuvent prendre n'importe quelle valeur dans un ensemble d'incertitude budgétisé. Une étude de la complexité de ces problèmes ainsi que des algorithmes d'approximation sont présentés.

Le problème de flow-shop à deux machines, avec l'objectif de minimiser le makespan dans le pire des cas (critère *min-max*), est étudié dans plusieurs articles [Kasperski 2012, Ying 2015, Levorato 2022]. Dans [Kasperski 2012], les auteurs considèrent un ensemble d'incertitude discret pour les durées de traitement et proposent un algorithme d'approximation pour résoudre le problème. Dans [Ying 2015], les durées de traitement incertaines sont définies par un budget d'incertitude. Les auteurs utilisent des métaheuristiques, notamment un algorithme de recuit simulé et un algorithme de recherche itérative, pour résoudre le problème sur un ensemble de 300 instances générées aléatoirement. Dans l'article [Levorato 2022], un algorithme de génération de colonnes et de contraintes est proposé pour le même problème. Cet algorithme est basé sur plusieurs formulations PLNE du problème déterministe. Il est détaillé dans le paragraphe 7.4.2 de ce document et sert de base de comparaison pour les méthodes de résolution proposées dans cette thèse.

Le problème de flow-shop robuste à deux machines, avec un objectif de minimisation du pire regret (*min-max-regret*), est abordé dans l'article [Kouvelis 2000]. Les auteurs considèrent deux types d'ensembles d'incertitude pour les durées des tâches : des ensembles discrets et des ensembles de type intervalle. Pour chacun de ces types, ils développent un algorithme de *branch-and-bound* ainsi qu'une méthode de résolution heuristique. Un algorithme génétique est proposé dans [Ćwik 2015], pour résoudre le problème à trois machines. Pour le cas général, sans limitation sur le nombre de machines considérées, un algorithme glouton est proposé dans [Ćwik 2018]. D'autre part, le problème de minimisation de la somme pondérée des date de fin pire cas est étudié dans [Levorato 2023].

Le problème de job-shop cyclique robuste à deux niveaux est considéré dans [Hamaz 2023, Hamaz 2018]. Un algorithme de *branch-and-bound*, une méthode de décomposition de Benders et une méthode de génération de colonnes et de contraintes

sont présentés.

Le problème d'ordonnancement de projets sous contrainte de ressources robuste à deux niveaux est étudié dans [Bruni 2017] et [Bold 2021]. L'objectif est de minimiser le makespan dans le pire des cas, en prenant en compte des durées de traitement incertaines.

Dans [Bruni 2017], les auteurs proposent un algorithme dynamique de complexité polynomiale pour résoudre le problème adverse, lorsque les durées des tâches sont incluses dans un ensemble d'incertitudes budgétisées. Ce problème adverse consiste à trouver le scénario qui entraîne le plus grand makespan pour une séquence de tâches donnée. De plus, ils présentent une méthode de décomposition de Benders pour résoudre le problème d'ordonnancement robuste.

Pour le même problème, une reformulation du problème adverse sous la forme d'un problème de plus long chemin dans un graphe est proposée dans [Bold 2021]. Cela leur permet d'utiliser le dual de ce problème adverse dans une formulation compacte. Ils démontrent expérimentalement la supériorité de cette formulation par rapport à la décomposition de Benders en utilisant des instances de référence de la littérature. De plus, dans [Bold 2022], les auteurs étendent cette formulation compacte à une version multi-mode du problème d'ordonnancement robuste. Ils observent également la supériorité de la formulation compacte par rapport à la décomposition de Benders proposée dans [Balouka 2021] pour cette extension du problème.

2.4.2 Définition formelle des problèmes étudiés

Dans la partie III, nous étudions différents problèmes d'ordonnancement, le job-shop, le job-shop flexible et le flow-shop de permutation, dans leur version robuste. Pour cela, nous considérons que les durées de traitement des opérations sont soumises à l'incertitude.

2.4.2.1 Budget d'incertitude

Afin de pouvoir contrôler la robustesse des solutions obtenues et ainsi trouver un équilibre entre la minimisation du makespan et la prise en compte de l'incertitude, nous définissons l'ensemble d'incertitude considéré à l'aide de l'approche proposée dans [Bertsimas 2004]. Celle-ci est basée sur la notion de budget d'incertitude (présentée dans le paragraphe 2.2.3) qui permet une restriction sur le nombre de déviations pouvant se produire dans un même scénario.

Étant donné ξ un scénario, on définit $p_{i,j}(\xi)$, la durée d'une opération $O_{i,j}$ dans un scénario ξ comme suit :

$$p_{i,j}(\xi) = \bar{p}_{i,j} + \xi_{i,j}\hat{p}_{i,j} \tag{2.20}$$

où $\xi_{i,j}$ est égal à 1 si la durée de l'opération dévie dans le scénario ξ, 0 sinon.

Nous étudions deux types de budget d'incertitude :

1. Un budget global Γ, qui désigne le nombre d'opérations dont la durée de traitement peut dévier, sur l'ensemble des machines. Dans ce cas, l'ensemble d'incertitude est exprimé sous la forme :

$$\mathcal{U} = \Big\{(\xi_{i,j})_{i\in\mathcal{J},j\in\{1,\dots,n_i\}} \mid \sum_{i\in\mathcal{J}}\sum_{j=1}^{n_i} \xi_{i,j} \leq \Gamma,\ \xi_{i,j} \in \{0,1\}\Big\}.$$

2. Un budget dépendant des machines $\Gamma = (\Gamma_1, \dots, \Gamma_{|\mathcal{M}|})$ où Γ_m désigne le nombre maximal d'opérations dont les temps de traitement peuvent dévier simultanément sur la machine $m \in \mathcal{M}$. Dans ce cas, l'ensemble d'incertitude est exprimé sous la forme :

$$\mathcal{U} = \Big\{(\xi_{i,j})_{i\in\mathcal{J},j\in\{1,\dots,n_i\}} \mid \sum_{O_{i,j}\in\mathcal{I}_m} \xi_{i,j} \leq \Gamma_m,\ \forall m \in \mathcal{M},\ \xi_{i,j} \in \{0,1\}\Big\}$$

où $\mathcal{I}_m$ est l'ensemble des opérations traitées sur la machine $m \in \mathcal{M}$.

Un budget d'incertitude dépendant des machines, $\Gamma = (\Gamma_1, \dots, \Gamma_{|\mathcal{M}|})$, vise à éviter de prendre en compte les scénarios extrêmes, où toutes les déviations se concentrent sur une seule machine [Song 2022]. Pour chaque machine $m \in \mathcal{M}$, il est alors possible d'adapter un budget Γ_m au niveau de confiance accordé à la machine.

2.4.2.2 Optimisation robuste à deux étapes

Les problèmes d'ordonnancement robuste considérés se prêtent naturellement à un processus de décision en deux étapes, où les décisions de séquencement des opérations doivent être prises en amont, avant que les durées d'opérations incertaines ne soient connues, mais où les dates de début des opérations peuvent s'adapter à la réalisation de l'incertitude. Sans connaître à l'avance les durées des opérations non terminées *(non clairvoyance)*, il est possible de trouver les dates de début de traitement optimales pour une séquence et un scénario donnés. Il suffit pour cela de commencer le traitement d'une opération dès que toutes les opérations qui la précèdent (à la fois sur le travail et sur la machine) sont terminées. Cette approche permet alors d'obtenir un ordonnancement semi-actif, dominant pour la minimisation du makespan.

Par conséquent, les décisions relatives à la séquence des opérations sur les machines sont les décisions de premier niveau, tandis que les dates de début des opérations constituent l'ensemble des décisions de second niveau de nos problèmes.

2.5 Conclusion

Dans ce chapitre, nous avons présenté les concepts clés de l'optimisation sous incertitude, en mettant l'accent sur l'optimisation robuste. Nous avons commencé par définir les problèmes statiques et multi-niveaux, ainsi que les ensembles d'in-

certitudes couramment considérés dans la littérature. Ensuite, nous nous sommes penchés sur les différentes méthodes de résolution utilisées en optimisation robuste. Enfin, nous avons examiné la littérature relative à l'ordonnancement robuste, en mettant en évidence les différentes approches proposées et les problèmes spécifiques étudiés. Nous avons également proposé une définition formelle des ensembles d'incertitude considérés.

L'ensemble de ces concepts sont utilisés dans la partie III de ce document, pour résoudre les problèmes à deux étapes de job-shop et de flow-shop de permutation robustes.

Deuxième partie

Méthodes de résolution exacte pour les problèmes de job-shop flexible

Chapitre 3

Approche par décomposition de Benders basée sur la logique

Sommaire

3.1 Introduction

Le problème de job-shop flexible peut être naturellement décomposé en un problème d'affectation de ressources et un problème d'ordonnancement. Dans cette thèse, nous proposons une méthode de décomposition de Benders basée sur la logique pour résoudre les problèmes de job-shop flexible, qu'ils soient préemptifs ou non préemptifs.

Dans le cas du problème non préemptif, une décomposition similaire a été présentée dans l'article [Naderi 2021]. Cependant, nous proposons une nouvelle formulation du problème maître ainsi que des coupes différentes de celles proposées dans la littérature. De plus, nous ajoutons une étape supplémentaire dans le schéma de décomposition pour accélérer la recherche.

Pour résoudre le problème maître, nous proposons un modèle de programmation linéaire en nombres entiers, qui est présenté dans le paragraphe 3.3. Ensuite, nous abordons la question de la génération de coupes d'optimalité dans le paragraphe 3.4.

Enfin, nous présentons un algorithme de décomposition dans le paragraphe 3.5. Ces travaux ont fait l'objet des publications [Juvin 2022b] et [Juvin 2023f].

La résolution des sous-problèmes est traitée dans des chapitres spécifiques, à savoir le chapitre 4 pour le problème non préemptif et le chapitre 5 pour le problème préemptif.

3.2 Définition du problème

Dans le cadre du problème de job-shop flexible (défini dans le paragraphe 1.2.1.4), on considère un ensemble de travaux $\mathcal{J}$ et un ensemble de machines $\mathcal{M}$. Chaque travail $i \in \mathcal{J}$ est constitué d'une séquence de n_i opérations. On note $O_{i,j}$, la $j^{ième}$ opération du travail $i \in \mathcal{J}$. Pour chaque opération $O_{i,j}$, il existe un ensemble de machines, notée $\mathcal{M}_{i,j}$, capables de la traiter. La durée de traitement de l'opération $O_{i,j}$ sur une machine $m \in \mathcal{M}_{i,j}$ est notée $p_{i,j,m}$.

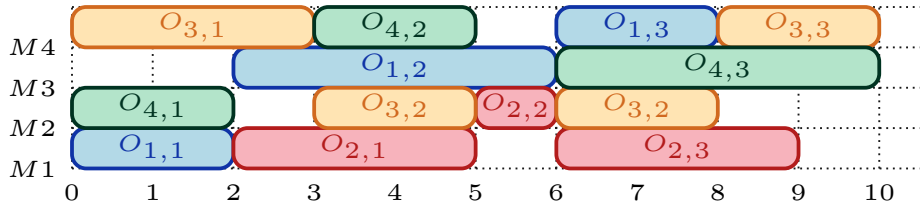
Exemple 4 *On considère une instance de job-shop flexible à quatre travaux et quatre machines. Chaque travail est constitué de trois opérations. Les temps de traitement $p_{i,j,m}$ de chaque opération $\{O_{i,j}|j = 1, \ldots, n_i\}_{i\in\mathcal{J}}$, sur chaque machine éligible $m \in \mathcal{M}_{i,j}$ sont donnés par le tableau 3.1. Un exemple d'affectation des tâches aux machines est donné en gras : les opérations $O_{1,1}$, $O_{2,1}$ et $O_{2,3}$ sont affectées à la machine $M1$, les opérations $O_{2,2}$, $O_{3,2}$ et $O_{4,1}$ à la machine $M2$, $O_{1,2}$ et $O_{4,3}$ à la machine $M3$ et les opérations $O_{1,3}$, $O_{3,1}$, $O_{3,3}$, $O_{4,2}$ à la machine $M4$.*

		$M1$	$M2$	$M3$	$M4$
$J1$	$O_{1,1}$	**2**	3		
	$O_{1,2}$		4	**4**	5
	$O_{1,3}$	4	2	2	**2**
$J2$	$O_{2,1}$	**3**	4		
	$O_{2,2}$		**1**	1	2
	$O_{2,3}$	**3**		2	
$J3$	$O_{3,1}$	3		5	**3**
	$O_{3,2}$	2	**4**	4	
	$O_{3,3}$			4	**2**
$J4$	$O_{4,1}$	3	**2**	2	
	$O_{4,2}$	2	4	2	**2**
	$O_{4,3}$	4	4	**4**	2

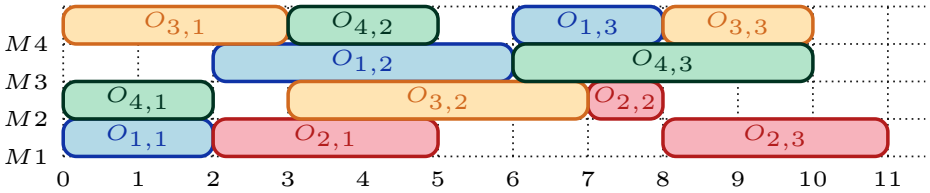
TABLEAU 3.1 – Exemple d'une instance de job-shop flexible : temps de traitement (en gras, une affectation possible)

La figure 3.1 représente, à l'aide de diagrammes de Gantt, un ordonnancement préemptif (3.1a) et non préemptif (3.1b) respectant les affectations données en gras dans le tableau 3.1.

On note que, dans l'ordonnancement préemptif, le traitement de l'opération $O_{3,2}$ est interrompue après deux unités de temps pour permettre l'exécution de l'opération $O_{2,2}$ sur la machine M_2. Une fois cette opération terminée, le traitement de



(a) Ordonnancement préemptif, makespan 10



(b) Ordonnancement non préemptif, makespan 11

FIGURE 3.1 – Diagrammes de Gantt de solutions d'ordonnancement préemptif et non préemptif pour l'exemple 4 (job-shop flexible 4 travaux × 4 machines)

l'opération $O_{3,2}$ reprend sur la même machine. Dans cet exemple, l'utilisation de la préemption permet de réduire le makespan de 11 *à* 10.

Le problème de job-shop flexible peut naturellement être décomposé en un problème d'affectation et un problème d'ordonnancement. Ainsi, dans le problème maître, chaque opération est affectée à exactement une machine parmi l'ensemble de ses machines éligibles. Résoudre ce problème maître permet alors de déterminer une borne inférieure BI du problème général. À chaque itération h, une fois les affectations x^h fixées par le problème maître, le sous-problème consiste à résoudre un problème de job-shop non flexible. La solution de ce sous-problème est une solution admissible du problème général et permet donc d'obtenir une borne supérieure (BS). La résolution du sous-problème permet aussi de déduire des coupes d'optimalité. Ces coupes sont ajoutées au problème maître qui est alors résolu à nouveau. La figure 3.2 illustre ce schéma de décomposition.

3.3 Problème maître

Le problème maître est une relaxation du problème général. Il consiste simplement à affecter chaque opération à une machine en capacité de la réaliser. Ainsi, le problème maître implique seulement des variables d'affectation, notées $x_{i,j,m}$. Sa

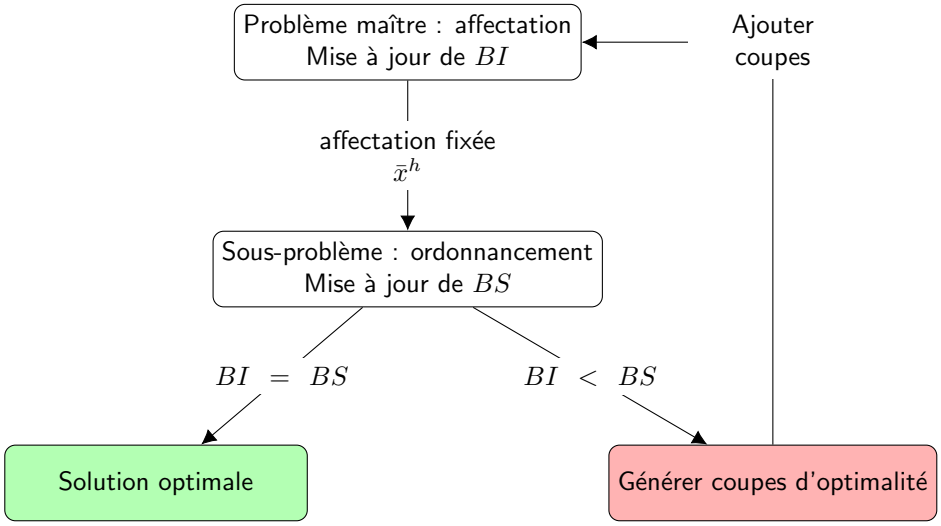


FIGURE 3.2 – Schéma de décomposition de Benders pour le problème de job-shop flexible

formulation est :

$$\min C_{\max} \tag{3.1}$$

$$\sum_{m \in \mathcal{M}_{i,j}} x_{i,j,m} = 1 \qquad \forall j \in \{1, \ldots, n_i\}, \forall i \in \mathcal{J} \tag{3.2}$$

$$\text{Coupes de Benders} \tag{3.3}$$

$$x_{i,j,m} \in \{0, 1\} \tag{3.4}$$

L'objectif est la minimisation du makespan (3.1). Les contraintes (3.2) assurent que chaque opération est affectée à exactement une machine. Les contraintes (3.3) représentent les coupes qui sont ajoutées itérativement au problème maître. Les détails de la génération de ces coupes sont donnés dans le paragraphe (3.4). Dans cette formulation, le seul lien entre l'objectif $C_{\max}$ et les variables de décision $x_{i,j,m}$ sont les contraintes induites par les coupes de Benders (3.3). Or, ces coupes sont ajoutées successivement au cours de la recherche. Ainsi, en début de processus, peu (ou aucune, lors de la première itération) d'informations permettent de guider la recherche. Afin de répondre à cette problématique, une relaxation du sous-problème est ajoutée au problème maître. Cette relaxation permet d'obtenir une borne inférieure plus précise sur la valeur optimale du problème, ce qui augmente les chances de trouver des affectations de bonne qualité.

3.3.1 Relaxation à un problème à un travail

La relaxation du sous-problème a pour but de guider le choix des variables d'affectation $x_{i,j,m}$, mais aussi d'obtenir une borne inférieure du makespan $C_{\max}$ de bonne qualité.

Un premier ensemble d'inégalités valides découle directement des relations de précédence entre les opérations consécutives d'un même travail :

$$C_{\max} \geq \sum_{j=1}^{n_i} \sum_{m \in \mathcal{M}_{i,j}} x_{i,j,m} p_{i,j,m} \qquad \forall i \in \mathcal{J} \tag{3.5}$$

Les contraintes (3.5) expriment le fait que le temps de traitement total est au moins égal aux temps de traitement cumulés de l'ensemble des opérations d'un même travail.

Exemple 5 *Considérons l'instance présentée dans l'exemple 4 et reprenons l'affectation indiquée en gras dans le tableau 3.1.*

Les contraintes (3.5) nous donnent :

- *Durée de traitement du travail $J1$:* $p_{1,1,1} + p_{1,2,3} + p_{1,3,4} = 2+4+2 = 8 \implies C_{\max} \geq 8$;
- *Durée de traitement du travail $J2$:* $p_{2,1,1} + p_{2,2,2} + p_{2,3,1} = 3+1+3 = 7 \implies C_{\max} \geq 7$;
- *Durée de traitement du travail $J3$:* $p_{3,1,4} + p_{3,2,2} + p_{3,3,4} = 3+4+2 = 9 \implies C_{\max} \geq 9$;
- *Durée de traitement du travail $J4$:* $p_{4,1,2} + p_{4,2,4} + p_{4,3,3} = 2+2+4 = 8 \implies C_{\max} \geq 8$.

Par conséquent, l'affectation considérée dans cet exemple conduit à un makespan d'au moins 9, donnée par la durée cumulée des opérations du travail $J3$.

3.3.2 Relaxation à un problème à une machine

De la même façon, les contraintes disjonctives sur les machines donnent l'ensemble d'inégalités valides suivant :

$$C_{\max} \geq \sum_{O_{i,j} \in \mathcal{I}_m} x_{i,j,m} p_{i,j,m} \qquad \forall m \in \mathcal{M} \tag{3.6}$$

avec $\mathcal{I}_m = \{O_{i,j} |\ i \in \mathcal{J},\ j \in \{1, \ldots, n_i\}\ ,\ m \in \mathcal{M}_{i,j}\}$ l'ensemble des opérations pouvant être traitées par une machine m.

Les contraintes (3.6) traduisent le fait que le temps de traitement total est supérieur aux temps de traitement cumulés de l'ensemble des opérations affectées à une même machine.

Exemple 6 *En reprenant le même exemple que précédemment, les contraintes (3.6) amènent :*

— *Durée de traitement sur la machine* $M1$ *:* $p_{1,1,1} + p_{2,1,1} + p_{2,3,1} = 2+3+3 = 8 \implies C_{\max} \geq 8$ *;*
— *Durée de traitement sur la machine* $M2$ *:* $p_{4,1,2} + p_{2,2,2} + p_{3,2,2} = 2+1+4 = 7 \implies C_{\max} \geq 7$ *;*
— *Durée de traitement sur la machine* $M3$ *:* $p_{1,2,3} + p_{4,3,3} = 4+4 = 8 \implies C_{\max} \geq 8$ *;*
— *Durée de traitement sur la machine* $M4$ *:* $p_{3,1,4} + p_{4,2,4} + p_{1,2,4} + p_{3,3,4} = 3+2+2+2 = 9 \implies C_{\max} \geq 9$.

En conséquence, on obtient un makespan d'au moins 9, valeur donnée par la durée cumulée des opérations sur la machine $M4$.

3.3.3 Relaxation à un problème à une machine avec dates de disponibilité et durées de latence

Il est possible d'affiner ces contraintes en considérant un problème à une machine obtenu en relaxant l'ensemble des contraintes disjonctives, sauf sur une machine. Cette idée est tirée de [Carlier 1989] pour le calcul de borne inférieure pour le problème de job-shop non flexible. On définit pour cela la date de disponibilité d'une opération $O_{i,j}$ (respectivement la durée de latence) comme la somme des durées minimales des opérations qui la précèdent (respectivement qui la succèdent) dans le travail i : $r_{i,j}^{min} = \sum_{j'<j} \min_{m \in \mathcal{M}_{i,j'}} p_{i,j',m}$ (respectivement $q_{i,j}^{min} = \sum_{j'>j} \min_{m \in \mathcal{M}_{i,j'}} p_{i,j',m}$).

Pour chaque machine m et pour tout sous-ensemble, non vide, d'opérations $\mathcal{I}'_m \subseteq \mathcal{I}_m$ pouvant être traitées par la machine m, on considère :

— r'_m : la date minimale de disponibilité du sous-ensemble $\mathcal{I}'_m$,

$$r'_m = \min_{O_{i,j} \in \mathcal{I}'_m} r_{i,j}^{min} \tag{3.7}$$

— q'_m : la durée minimale entre la fin de traitement du sous-ensemble $\mathcal{I}'_m$ et la fin de l'ordonnancement,

$$q'_m = \min_{O_{i,j} \in \mathcal{I}'_m} q_{i,j}^{min} \tag{3.8}$$

pour former l'ensemble d'inégalités valides suivant :

$$C_{\max} \geq r'_m + \sum_{O_{i,j} \in \mathcal{I}'_m} x_{i,j,m} p_{i,j,m} + q'_m \quad \forall m \in \mathcal{M}, \forall \mathcal{I}'_m \subseteq \mathcal{I}_m. \tag{3.9}$$

Pour tout sous-ensemble d'opérations $\mathcal{I}'_m \subseteq \mathcal{I}_m$ pouvant être traitées par une machine m, les contraintes (3.9) obligent le makespan à être au moins égal à la somme des temps de traitement des opérations de ce sous-ensemble affectées à la machine m à laquelle on ajoute les durées minimales avant et après le traitement de ce sous-ensemble d'opérations.

Le nombre de contraintes (3.9) croît de manière exponentielle avec le nombre

d'opérations dans le problème. Cependant, en pratique, il n'est pas nécessaire de considérer tous les sous-ensembles de $\mathcal{I}_m$, comme on le montre ci-dessous.

Théorème 1 *Soient $\mathcal{I}'_m$ et $\mathcal{I}''_m$ deux sous-ensembles de $\mathcal{I}_m$ tels que $\mathcal{I}''_m \subset \mathcal{I}'_m$ et $\exists O_{i,j} \in \mathcal{I}''_m$ avec $r^{min}_{i,j} = r'_m$ et $\exists O_{i',j'} \in \mathcal{I}''_m$ avec $q^{min}_{i',j'} = q'_m$, alors la contrainte $C_{\max} \geq r'_m + \sum_{O_{i,j} \in \mathcal{I}'_m} x_{i,j,m} p_{i,j,m} + q'_m$ domine la contrainte $C_{\max} \geq r''_m + \sum_{O_{i,j} \in \mathcal{I}''_m} x_{i,j,m} p_{i,j,m} + q''_m$.*

Preuve 1 *Pour montrer qu'une contrainte notée $C1$ domine une autre contrainte notée $C2$, il suffit de prouver que $C1$ implique $C2$.*

Par hypothèse, $r'_m = r''_m$, $q''_m = q'_m$ et, pour toute affectation, $\sum_{O_{i,j} \in \mathcal{I}'_m} x_{i,j,m} p_{i,j,m} \geq \sum_{O_{i,j} \in \mathcal{I}''_m} x_{i,j,m} p_{i,j,m}$. Ainsi, $r'_m + \sum_{O_{i,j} \in \mathcal{I}'_m} x_{i,j,m} p_{i,j,m} + q'_m \geq r''_m + \sum_{O_{i,j} \in \mathcal{I}''_m} x_{i,j,m} p_{i,j,m} + q''_m$.

On en déduit que :
$C_{\max} \geq r'_m + \sum_{O_{i,j} \in \mathcal{I}'_m} x_{i,j,m} p_{i,j,m} + q'_m \implies C_{\max} \geq r''_m + \sum_{O_{i,j} \in \mathcal{I}''_m} x_{i,j,m} p_{i,j,m} + q''_m$.
□

Ainsi, sur la base du Théorème 1, l'ensemble de contraintes (3.9) peut être remplacé par les contraintes (3.10) sans perte de précision. Cela permet de réduire considérablement le nombre de contraintes nécessaires.

$$C_{\max} \geq r_a + \sum_{O_{i,j} \in \mathcal{I}_a} x_{i,j,m} p_{i,j,m} + q_a \quad \forall m \in \mathcal{M},\ \forall a = (O_{i,j}, O_{i',j'}) \in \mathcal{I}^2_m \tag{3.10}$$

avec $r_a = \min(r^{min}_{i,j}, r^{min}_{i',j'})$, $q_a = \min(q^{min}_{i,j}, q^{min}_{i',j'})$ et $\mathcal{I}_a = \{O_{i,j} \in \mathcal{I}_m \,|\, r^{min}_{i,j} \geq r_a,\ q^{min}_{i,j} \geq q_a\}$.

Exemple 7 *En considérant toujours le même exemple ainsi que le sous-ensemble $\mathcal{I}_a \subset \mathcal{I}_1$ d'opérations pouvant être traitées sur la machine M_1 avec $a = (O_{1,1}, O_{2,1})$, on obtient :*

$$r_a = \min(r^{min}_{1,1}, r^{min}_{2,1}) = \min(0, 0) = 0$$
$$q_a = \min(q^{min}_{1,1}, q^{min}_{2,1}) = \min(p^{min}_{1,2} + p^{min}_{1,3}, p^{min}_{2,2} + p^{min}_{2,3}) = \min(4 + 2, 1 + 2) = 3$$

Le sous-ensemble $\mathcal{I}_a$ est l'ensemble des opérations pouvant être traitées par la ma-

chine M_1 telles que $r^{min}_{i,j} \geq r_a = 0$ et $q^{min}_{i,j} \geq q_a = 3$, d'où :

$$\begin{aligned}
r^{min}_{1,1} = 0 \geq r_a \quad et \quad q^{min}_{1,1} = 6 \geq q_a &\implies O_{1,1} \in \mathcal{I}_a \\
q^{min}_{1,3} = 0 < q_a &\implies O_{1,3} \notin \mathcal{I}_a \\
r^{min}_{2,1} = 0 \geq r_a \quad et \quad q^{min}_{2,1} = 3 \geq q_a &\implies O_{2,1} \in \mathcal{I}_a \\
q^{min}_{2,3} = 0 < q_a &\implies O_{2,3} \notin \mathcal{I}_a \\
r^{min}_{3,1} = 0 \geq r_a \quad et \quad q^{min}_{3,1} = 6 \geq q_a &\implies O_{3,1} \in \mathcal{I}_a \\
r^{min}_{4,2} = 2 \geq r_a \quad et \quad q^{min}_{4,2} = 4 \geq q_a &\implies O_{4,2} \in \mathcal{I}_a \\
&\implies \mathcal{I}_a = \{O_{1,1}, O_{2,1}, O_{3,1}, O_{4,2}\}
\end{aligned}$$

D'après les contraintes (3.10), on obtient donc :

$$\begin{aligned}
C_{\max} &\geq 0 + p_{1,1,1}x_{1,1,1} + p_{2,1,1}x_{2,1,1} + p_{3,1,1}x_{3,1,1} + p_{4,2,1}x_{4,2,1} + 3 \\
&\geq 0 + 2 \times 1 + 3 \times 1 + 3 \times 0 + 4 \times 0 + 3 = 8.
\end{aligned}$$

On notera que, si l'on considère l'ensemble des machines et l'ensemble des paires d'opérations, les contraintes (3.10) nous donnent $C_{\max} \geq 10$ (pour $\mathcal{I}_{a'} = \{O_{1,2}, O_{4,3}\}$ sur la machine M_3), ce qui montre que la solution d'ordonnancement préemptif représentée par la figure 3.1a est optimale pour les affectations considérées dans cet exemple.

Pour chaque itération h, on obtient une solution du problème maître, à savoir une affectation x^h des opérations aux machines. On note $x^h_{i,j,m}$ la valeur de la variable d'affectation $x_{i,j,m}$ à l'itération h. On définit $\mathcal{P}^h = \{(i,j,m) \,|\, x^h_{i,j,m} = 1,\ i \in \mathcal{J},\ j \in \{1,\ldots,n_i\},\ m \in \mathcal{M}_{i,j}\}$ l'ensemble des affectations qui déterminent entièrement une solution du problème maître et $\mathcal{P}^h_m = \{(i,j) \,|\, x^h_{i,j,m} = 1,\ i \in \mathcal{J},\ j \in \{1,\ldots,n_i\}\}$ l'ensemble des opérations affectées à la machine m.

3.4 Coupes de Benders logiques

Après chaque itération, des coupes logiques sont ajoutées au problème maître. Ces coupes sont déduites à partir d'informations obtenues par la résolution d'un sous-problème. Elles permettent d'exclure des affectations non optimales et forcent le problème maître à explorer d'autres solutions. On considère dans ce chapitre que la solution optimale de ce sous-problème est obtenue à l'aide d'un oracle.

Soit z^h le makespan optimal, trouvé par la résolution d'un sous-problème, pour une affectation $\mathcal{P}^h$ donnée. On en déduit la coupe de Benders logique triviale suivante :

$$C_{\max} \geq z^h\Big(1 - \sum_{(i,j,m)\in\mathcal{P}^h} (1 - x_{i,j,m})\Big). \tag{3.11}$$

La contrainte (3.11) indique que le makespan ne peut être inférieur à z^h que si

l'affectation d'au moins une opération est différente de celle fixée par le problème maître à l'itération h. En effet, si les mêmes affectations sont choisies, c'est-à-dire $x_{i,j,m} = x^h_{i,j,m},\ \forall i \in \mathcal{J},\ \forall j \in \{1,\dots,n_i\},\ \forall m \in \mathcal{M}_{i,j}$, alors la somme est nulle et la partie droite de l'inégalité est égale à z^h. Cette contrainte est inactive (la partie droite est nulle ou négative) si au moins une des variables $x_{i,j,m}$ prend une valeur différente.

L'inégalité (3.11) est valide ; elle n'exclut aucune solution faisable. Cependant, chaque coupe de ce type ne permet d'exclure qu'une seule solution du problème maître puisqu'elle devient inactive dès qu'une des affectations est modifiée. Ainsi, son efficacité est limitée. Le but est alors de rendre ces coupes plus générales en identifiant un sous-ensemble d'opérations dont l'affectation conduit à un makespan identique.

$$C_{\max} \geq z^h\Big(1 - \sum_{(i,j,m)\in\mathcal{Q}^h} (1 - x_{i,j,m})\Big) \tag{3.12}$$

avec $\mathcal{Q}^h \subset \mathcal{P}^h$ un sous-ensemble d'affectations qui aboutit à la valeur de la fonction objectif z^h.

3.4.1 Chemin critique

Afin de trouver une coupe logique impliquant seulement un sous-ensemble d'opérations, on s'intéresse dans un premier temps aux opérations constituant le chemin critique dans la solution retournée par le sous-problème.

Proposition 1 *Soient $\mathcal{C}^h_c$ l'ensemble des opérations formant l'un des chemins critiques dans une solution optimale du sous-problème et z^h le makespan de cette solution. La coupe suivante :*

$$C_{\max} \geq z^h\Big(1 - \sum_{(i,j,m)\in\mathcal{C}^h_c} (1 - x_{i,j,m})\Big) \tag{3.13}$$

n'est pas toujours valide.

Preuve 2 *Contre-exemple 8.*

Exemple 8 *On considère l'instance de job-shop flexible présentée dans l'exemple 4 et les affectations, fixées par le problème maître, suivantes : $x^h_{1,1,2} = x^h_{1,2,4} = x^h_{1,3,1} = x^h_{2,1,2} = x^h_{2,2,4} = x^h_{2,3,1} = x^h_{3,1,4} = x^h_{3,2,1} = x^h_{3,3,3} = x^h_{4,1,1} = x^h_{4,2,2} = x^h_{4,3,4} = 1$. Un ordonnancement optimal pour ces affectations est donné par la figure 3.3. Cette solution a pour makespan 15 et possède un chemin critique $\mathcal{C}^h_c$ constitué des opérations $O_{1,1}$, $O_{1,2}$, $O_{1,3}$ et $O_{2,3}$.*

Il est possible de trouver un ordonnancement de makespan inférieur à 15, avec toutes les opérations du chemin critique $\mathcal{C}^h_c$ affectées aux mêmes machines ($O_{1,1}$ sur la machine M_2, $O_{1,2}$ sur la machine M_4, $O_{1,3}$ et $O_{2,3}$ sur la machine M_1). Un tel ordonnancement est représenté par la figure 3.4.

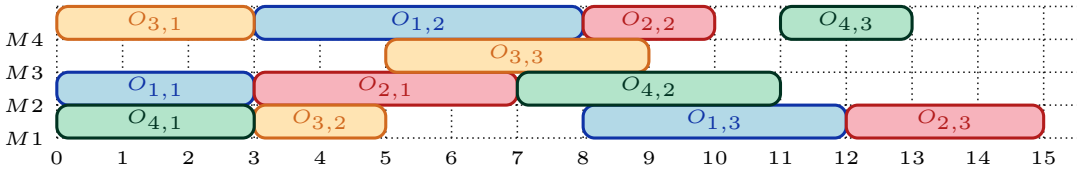


FIGURE 3.3 – Diagramme de Gantt : ordonnancement optimal pour les affectations de l'exemple 8 avec un makespan de 15 et un chemin critique composé des opérations $O_{1,1}$, $O_{1,2}$, $O_{1,3}$ et $O_{2,3}$

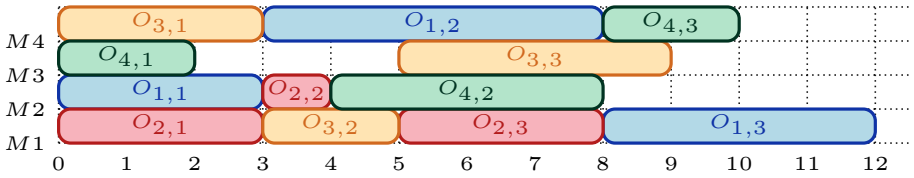


FIGURE 3.4 – Diagramme de Gantt : un ordonnancement avec les affectations $x_{1,1,1} = x_{1,2,4} = x_{1,3,1} = x_{2,3,1} = 1$ de l'exemple 8

Fixer l'affectation des opérations d'un chemin critique n'entraîne donc pas nécessairement un makespan supérieur ou égal à la longueur de ce chemin et la coupe (3.13) n'est pas toujours valide. □

3.4.2 Borne supérieure de l'influence d'une affectation sur la valeur du makespan

Dans [Naderi 2021], les auteurs proposent la coupe suivante :

$$C_{\max} \geq z^h - \sum_{(i,j,m)\in\mathcal{P}^h} p_{i,j,m}(1 - x_{i,j,m}). \tag{3.14}$$

Tout comme la coupe (3.11), si aucune des affectations ne change, le makespan ne peut être inférieur à z^h. Sinon, le makespan de la nouvelle solution est supérieur ou égal à z^h auquel on retire la somme des durées des opérations dont l'affectation change. Ils justifient la validité de cette coupe en affirmant que si une opération est retirée d'un ordonnancement, alors le makespan est au plus réduit de la durée de cette opération. Or cette affirmation est fausse comme on peut le voir dans l'exemple suivant :

Exemple 9 *Considérons l'instance de job-shop flexible présentée dans l'exemple 4 et les affectations fixées par le problème maître suivant :* $x^h_{1,1,2} = x^h_{1,2,2} = x^h_{1,3,3} = x^h_{2,1,1} = x^h_{2,2,3} = x^h_{2,3,1} = x^h_{3,1,4} = x^h_{3,2,3} = x^h_{3,3,4} = x^h_{4,1,1} = x^h_{4,2,3} = x^h_{4,3,2} = 1$.

Un ordonnancement optimal pour ces affectations est donné par la figure 3.5. Le makespan de cette solution est de 13.

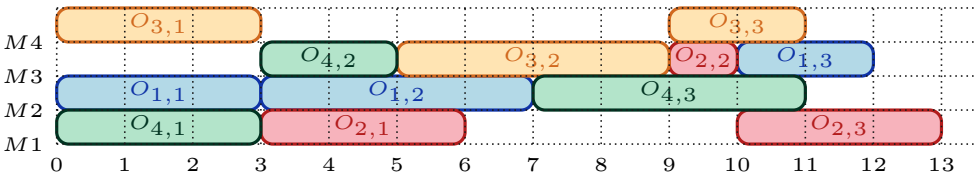


FIGURE 3.5 – Diagramme de Gantt : ordonnancement non préemptif optimal pour les affectations de l'exemple 9 avec un makespan de 13

D'après la coupe donnée par l'inégalité (3.14), si l'ensemble des affectations fixées restent les mêmes sauf pour l'opération $O_{2,2}$, le makespan devrai être supérieur à $13 - p_{2,2,3} = 13 - 1 = 12$.

Or, la figure 3.6 représente un ordonnancement avec ces caractéristiques de makespan 11.

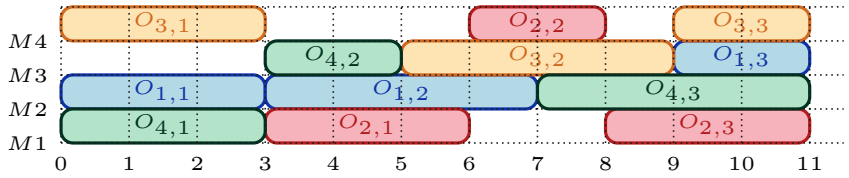


FIGURE 3.6 – Diagramme de Gantt : ordonnancement avec les affectations de l'exemple 9, sauf pour l'opération $O_{2,2}$, avec un makespan de 11

L'inégalité (3.14) n'est donc pas toujours une coupe de Benders valide.

3.4.3 Problème à une machine avec dates de disponibilité et durées de latence

Comme dans le paragraphe 3.3.3, l'idée de [Carlier 1989] est utilisée, à savoir relâcher le problème de job-shop en un problème à une machine avec dates de disponibilité et durées de latence pour trouver une borne inférieure. À chaque itération h, après avoir résolu le problème maître, il est possible d'évaluer la qualité de l'affectation donnée par ce dernier, avant même de résoudre le sous-problème.

En effet, l'affectation des opérations aux machines étant totalement déterminée, nous connaissons la durée de traitement $p^h_{i,j} = \sum_{m \in \mathcal{M}_{i,j}} p_{i,j,m} x^h_{i,j,m}$ de chaque opération $O_{i,j}$ (et non plus une borne inférieure de celle-ci). Il est ainsi possible de redéfinir la date de disponibilité minimale (respectivement la durée de latence minimale) d'une opération $O_{i,j}$ comme $r^{min,h}_{i,j} = \sum_{j'<j} p^h_{i,j}$ (respectivement $q^{min,h}_{i,j} =$

$\sum_{j'>j} p^h_{i,j}$). Le calcul d'une borne inférieure plus précise de la valeur du makespan est alors accessible. Soient m une machine et $a = (O_{i,j}, O_{i',j'}) \in \mathcal{P}^h_m \times \mathcal{P}^h_m$ une paire d'opérations affectées à la machine m. On redéfinit la plus petite date de disponibilité $r_a = min(r^{min,h}_{i,j}, r^{min,h}_{i',j'})$ et la durée de latence minimale $q_a = min(q^{min,h}_{i,j}, q^{min,h}_{i',j'})$ ainsi que $\mathcal{P}_a = \{O_{i,j} \in \mathcal{P}^h_m \,|\, m \in \mathcal{M}_{i,j},\, r^{min,h}_{i,j} > r_a,\, q^{min,h}_{i,j} > q_a\}$ l'ensemble des opérations affectées à la machine m dont la date de disponibilité et la durée de latence sont supérieures ou égales, respectivement, à r_a et q_a. Alors, $BI_a = r_a + \sum_{O_{i,j} \in \mathcal{P}_a} p_{i,j,m} + q_a$ est une borne inférieure du makespan z^h dans le sous-problème. Ainsi, si BI_a est supérieure ou égale au makespan de la meilleure solution connue (noté BS), il est inutile de résoudre le sous-problème car il n'améliorera pas la solution. Afin de tirer parti de cette information, il convient d'ajouter la coupe suivante au problème maître :

$$C_{\max} \geq BI_a\Big(1 - \sum_{m \in \mathcal{M}} \sum_{O_{i,j} \in \mathcal{S}_a \cap \mathcal{P}^h_m} (1 - x^h_{i,j,m})\Big). \tag{3.15}$$

avec $\mathcal{S}_a$ l'ensemble des opérations de $\mathcal{P}_a$ ainsi que les opérations des travaux dont au moins une opération appartient à cet ensemble (car celles-ci ont une influence sur le calcul de r_a ou q_a).

Exemple 10 *Considérons à nouveau l'instance présentée dans l'exemple 4, avec les mêmes affectations que dans l'exemple 7, à savoir :* $x^h_{1,1,1} = x^h_{1,2,3} = x^h_{1,3,4} = x^h_{2,1,1} = x^h_{2,2,2} = x^h_{2,3,1} = x^h_{3,1,4} = x^h_{3,2,2} = x^h_{3,3,4} = x^h_{4,1,2} = x^h_{4,2,4} = x^h_{4,3,3} = 1$.

Étudions encore une fois le sous-ensemble formé à partir de la paire $a = (O_{1,1}, O_{2,1})$. *On a alors :*

$$r_a = \min(r^h_{1,1}, r^h_{2,1}) = \min(0, 0) = 0$$
$$q_a = \min(q^h_{1,1}, q^h_{2,1}) = \min(p_{1,2,3} + p_{1,3,4}, p_{2,2,2} + p_{2,3,1}) = \min(4 + 2, 1 + 3) = 4.$$

Le sous-ensemble $\mathcal{P}_a$ *est l'ensemble des opérations* $O_{i,j}$ *affectées à la machine* M_1 *telles que* $r^h_{i,j} \geq r_a = 0$ *et* $q^h_{i,j} \geq q_a = 4$, *d'où* $\mathcal{P}_a = \{O_{1,1}, O_{2,1}\}$ *et* $\sum_{O_{i,j} \in \mathcal{P}_a} p_{i,j,1} = p_{1,1,1} + p_{2,1,1} = 2 + 3 = 5$. *Ainsi,* $BI_a = r_a + \sum_{O_{i,j} \in \mathcal{P}_a} p_{i,j,1} + q_a = 9$.

Bien que la paire d'opérations $a = (O_{1,1}, O_{2,1})$ *considérée dans cet exemple soit la même que celle étudiée dans l'exemple 7, la borne obtenue ici est de meilleure qualité car les valeurs de* r_a *et* q_a *sont obtenues différemment.*

Les affectations étant prédéterminées par le problème maître, seules les durées de traitement associées à celle-ci sont prises en considération pour calculer la plus petite date de disponibilité $r^{min,h}_{i,j}$ *et la durée de latence minimale* $q^{min,h}_{i,j}$ *d'une opération* $O_{i,j}$ *(il n'est pas nécessaire de prendre en compte toutes les affectations possibles des opérations antérieures et ultérieures).*

3.5 Algorithme de décomposition

Sur la base des composantes décrites dans les paragraphes précédents, nous proposons l'algorithme 1. Celui-ci est utilisé dans les chapitres 4 et 5 pour la méthode de décomposition de Benders basée sur la logique.

Algorithme 1 Algorithme de décomposition

Initialisation :
Problème maître ← ensembles de contraintes (3.2), (3.4), (3.5) et (3.10) ;
$meilleureSolution$ ← solution de **Heuristique 1** ;
$h \leftarrow 0$, $BI \leftarrow 0$, $BS \leftarrow meilleureSolution.makespan$;
Étape 1 Problème maître :
$\mathcal{P}^h \leftarrow$ affectation donnée par le problème maître ;
BI mise à jour ;
Étape 2 Relaxations du sous-problème :
$affectationPrometteuse \leftarrow$ **vrai**
pour tout $m \in \mathcal{M}$ **faire**
  pour tout $a \leftarrow (O_{i,j}, O_{i',j'}) \in \mathcal{P}_m^h \times \mathcal{P}_m^h$ **faire**
    $r_a \leftarrow min(r_{i,j}^{min,h}, r_{i',j'}^{min,h})$
    $q_a \leftarrow min(q_{i,j}^{min,h}, q_{i',j'}^{min,h})$
    $\mathcal{P}_a \leftarrow \{O_{i,j} \in \mathcal{P}_m^h \mid r_{i,j}^{min,h} > r_a,\ q_{i,j}^{min,h} > q_a\}$
    $BI_a \leftarrow r_a + \sum_{O_{i,j} \in \mathcal{P}_a} p_{i,j,m} + q_a$
    si $BI_a \geq BS$ **alors**
      Ajout de la coupe (3.15) au problème maître ;
      $affectationPrometteuse \leftarrow$ **faux**
si $(affectationPrometteuse =$ **faux**$)$ **alors**
  Retour à **Étape 1** ;
Étape 3 Sous-problème :
$s^h, z^h \leftarrow$ Solution optimale du sous-problème pour l'affectation $\mathcal{P}^h$ et makespan associé
si $z^h < BS$ **alors**
  $meilleureSolution \leftarrow s^h$
  $BS \leftarrow z^h$
si $BI < BS$ **alors**
  Ajout de la coupe (3.11) au problème maître ;
  Retour à **Étape 1** ;
sinon
  Retourner $meilleureSolution$;

Lors de la phase d'initialisation, le problème maître est formé à partir des contraintes d'affectation des opérations aux machines (contraintes (3.2)) ainsi que d'une relaxation du sous-problème (contraintes (3.5) et (3.10)). Pour obtenir une première solution admissible, une heuristique naïve, notée **Heuristique 1**, est utilisée. Elle consiste en un algorithme glouton qui utilise la règle de priorité *MWR* (*most work remaining*). Cette règle donne la priorité aux opérations dont le temps

de traitement restant sur le travail est le plus élevée. L'affectation aux machines se fait en choisissant à chaque fois la machine pouvant terminer l'exécution au plus tôt.

L'étape 1 consiste en la résolution du problème maître et permet d'obtenir une affectation $\mathcal{P}^h$ des opérations aux machines.

Plutôt que de résoudre directement le sous-problème résultant, la solution $\mathcal{P}^h$ est évaluée à l'aide de la relaxation présentée dans le paragraphe 3.4.3. Ainsi, pour chaque machine et pour chaque paire d'opérations qui lui sont affectées, une borne inférieure du makespan est calculée. Si celle-ci est supérieure ou égale au makespan de la meilleure solution déjà obtenue, alors l'affectation n'a aucune chance d'améliorer la solution et une coupe d'optimalité est ajoutée au problème maître.

Sinon, comme dans un schéma de décomposition classique, le sous-problème est évalué et permet d'obtenir la solution optimale pour l'affectation $\mathcal{P}^h$ et de mettre à jour la meilleure solution obtenue.

Enfin, l'optimalité de la solution est testée. Si elle est atteinte, la solution optimale est retournée. Sinon, une coupe d'optimalité est générée, et le problème maître est résolu à nouveau.

3.6 Conclusion

Dans ce chapitre, nous proposons un schéma de décomposition de Benders basé sur la logique pour résoudre le problème de job-shop flexible. Ce problème se décompose en un problème maître d'affectation des opérations aux machines et un sous-problème d'ordonnancement. Nous présentons un modèle de PLNE pour le problème maître, renforcé par une relaxation du sous-problème.

Nous décrivons également comment, à partir d'une solution optimale du sous-problème pour une affectation donnée, il est possible de générer des coupes logiques permettant, itérativement, d'améliorer la borne inférieure et donc d'orienter le problème maître vers la génération de meilleures solutions.

En outre, nous proposons d'ajouter une étape supplémentaire au schéma de décomposition classique, à savoir une relaxation du sous-problème pour chaque affectation fixée par le problème maître. Cette relaxation permet d'évaluer rapidement la qualité des affectations. Elle permet ainsi de limiter le nombre de sous-problèmes résolus en évitant de résoudre inutilement des instances peu prometteuses.

La résolution des sous-problèmes est abordée dans les chapitres suivants, notamment le chapitre 4 pour le problème non préemptif et le chapitre 5 pour le problème préemptif.

Chapitre 4

Le problème de job-shop flexible non préemptif

Sommaire

4.1 Introduction

Brucker et Schlie [Brucker 1990] sont les premiers à s'intéresser au problème de FJSSP en 1990. Ils proposent alors un algorithme polynomial permettant de résoudre le problème à deux travaux ($n = 2$). Pour le cas général, en tant qu'extension du JSSP, le FJSSP est *NP-difficile*. En raison de cette complexité, de nombreuses métaheuristiques ont été développées, par exemple la recherche taboue [Shen 2018, Hurink 1994], les algorithmes génétiques [Pezzella 2008, Xianzhou 2011] ou le recuit simulé [Najid 2002, Yazdani 2015].

Des méthodes exactes ont également été conçues, incluant une majorité de modèles mathématiques [Özgüven 2010, Ziaee 2018]. Dans [Meng 2020] quatre modèles de programmation linéaire en nombres entiers mixtes et un modèle de programmation par contraintes sont comparés. Les résultats expérimentaux montrent que la formulation PLNE la plus efficace est celle basée sur les variables de précédence. Cette même étude montre qu'une résolution à l'aide de la PPC permet d'obtenir de meilleure performance et résout des instances jusqu'à 30 travaux et 10 machines ou 15 travaux et 18 machines. Une revue de la littérature répertoriant les différentes méthodes de résolution est présentée dans [Chaudhry 2016].

Dans ce chapitre, on s'intéresse au méthodes exactes pour le problème de job-shop flexible non préemptif. Dans un premier temps, des formulations mathématiques directes permettant de fixer à la fois les décisions d'affectation et d'ordonnancement sont présentées dans le paragraphe 4.2. Ces formulations sont basées sur la PLNE (4.2.1) et la PPC (4.2.1). Ensuite, dans le paragraphe 4.3, nous présentons des méthodes de résolution du sous-problème dans le cadre d'une approche utilisant l'algorithme de décomposition de Benders basée sur la logique exposé dans le chapitre 3. Enfin, des expérimentations numériques permettant de comparer les performances ces différentes méthodes sont présentées dans le paragraphe 4.4.

4.2 Formulations directes

Dans ce paragraphe, nous présentons deux formulations mathématiques directes pour la résolution du problème de job-shop flexible. Contrairement à la méthode de décomposition présentée dans le chapitre 3, ces modèles permettent de résoudre le problème en une seule itération en déterminant ensemble l'affectation et l'ordonnancement des opérations sur les machines.

4.2.1 Programmation linéaire en nombres entiers

La formulation PLNE présentée ici est basée sur une formulation disjonctive. Elle est inspirée du modèle proposé dans [Shen 2018] pour le problème de job-shop flexible avec temps de préparation dépendant de la séquence. Ici, elle est simplement adaptée au problème étudié en ne tenant pas compte de ces temps de préparation.

On définit $t_{i,j}$ comme la variable entière représentant la date de début de traitement de l'opération $O_{i,j}$, $\forall i \in \mathcal{J}$, $\forall j \in \{1, \dots, n_i\}$, $C_{\max}$ la variable entière représentant le makespan, ainsi que les variables binaires suivantes :

— variables d'affectation :

$$x_{i,j,m} = \begin{cases} 1 & \text{si } O_{i,j} \text{ affectée à la machine } m \\ 0 & \text{sinon.} \end{cases} \quad \forall i \in \mathcal{J}, \forall j \in \{1, \dots, n_i\}, \forall m \in \mathcal{M}_{i,j}$$

— variables disjonctives :

$$y_{i,j,i',j'} = \begin{cases} 1 & \text{si } O_{i,j} \text{ traitée avant } O_{i',j'} \\ 0 & \text{sinon.} \end{cases} \quad \begin{array}{l} \forall (i,i') \in \mathcal{J}^2, \forall j \in \{1, \dots, n_i\}, \\ \forall j' \in \{1, \dots, n_{i'}\} \text{ t.q. } O_{i',j'} \neq O_{i,j}, \end{array}$$

Le problème de job-shop flexible peut alors s'exprimer sous la forme du modèle

suivant :

$$\min C_{\max} \tag{4.1}$$

$$\sum_{m \in \mathcal{M}_{ij}} x_{i,j,m} = 1 \qquad \forall i \in \mathcal{J},\ \forall j \in \{1, \ldots, n_i\} \tag{4.2}$$

$$t_{i,j} \geq t_{i,j-1} + \sum_{m \in \mathcal{M}_{i,j-1}} x_{i,j-1,m} p_{i,j-1,m} \qquad \forall i \in \mathcal{J},\ \forall j \in \{2, \ldots, n_i\} \tag{4.3}$$

$$t_{i,j} \geq t_{i',j'} + p_{i',j',m} - (2 - x_{i,j,m} - x_{i',j',m} + y_{i,j,i',j'})H \qquad \forall m \in \mathcal{M}_{i,j} \cap \mathcal{M}_{i'j'},\ \forall (O_{i,j}, O_{i',j'}) \in \mathcal{I}_m^2 \text{ t.q. } O_{i',j'} \neq O_{i,j} \tag{4.4}$$

$$t_{i',j'} \geq t_{i,j} + p_{i,j,m} - (3 - x_{i,j,m} - x_{i',j',m} - y_{i,j,i',j'})H \qquad \forall m \in \mathcal{M}_{i,j} \cap \mathcal{M}_{i'j'},\ \forall (O_{i,j}, O_{i',j'}) \in \mathcal{I}_m^2 \text{ t.q. } O_{i',j'} \neq O_{i,j} \tag{4.5}$$

$$C_{\max} \geq t_{i,n_i} + \sum_{m \in \mathcal{M}_{i,n_i}} x_{i,n_i,m} p_{i,n_i,m} \qquad \forall i \in \mathcal{J} \tag{4.6}$$

$$x_{i,j,m} \in \{0, 1\} \qquad \forall i \in \mathcal{J},\ \forall j \in \{1, \ldots, n_i\} \tag{4.7}$$

$$y_{i,j,i',j'} \in \{0, 1\} \qquad \forall i \in \mathcal{J},\ \forall j \in \{1, \ldots, n_i\} \tag{4.8}$$

avec H une borne supérieure du makespan pire cas.

La fonction objectif (4.1) consiste à minimiser le makespan. Les contraintes (4.2) assurent que chaque opération est affectée à exactement une machine éligible. Les contraintes (4.3) permettent de satisfaire les relations de précédence entre les opérations consécutives d'un même travail. Les contraintes (4.4) et (4.5) arbitrent les disjonctions entre opérations affectées à une même machine. Enfin, les contraintes (4.6) définissent le makespan, les dernières contraintes (4.7 et 4.8) précisant la nature binaire des variables disjonctives et d'affectation.

4.2.2 Programmation par contraintes

Le modèle de PPC présenté ici est celui proposé par IBM ILOG CPLEX CP Optimizer comme exemple illustratif [IBM 2022].

Cette formulation utilise des variables d'intervalle pour modéliser la période pendant laquelle une tâche est en cours de traitement. Les variables d'intervalle peuvent également être optionnelles, ce qui permet, par exemple, de représenter les différents modes dans lesquels une opération peut être réalisée sur différentes machines. Dans ce modèle, les variables d'intervalle relatives à une opération sont dupliquées pour chaque machine éligible, et une seule variable est prise en compte dans la solution, en fonction de l'affectation optimale.

Pour modéliser le problème, la variable continue $C_{\max}$ représente le makespan et les variables d'intervalle suivantes sont définies :

— $task_{i,j}$: variable représentant l'intervalle entre le début et la fin de l'opération $O_{i,j}$, $\forall i \in \mathcal{J}$, $\forall j \in \{1, \ldots, n_i\}$;

— $mode_{i,j,m}$: variable optionnelle représentant l'intervalle entre le début et la fin de l'opération $O_{i,j}$ sur la machine m, $\forall i \in \mathcal{J}$, $\forall j \in \{1, \ldots, n_i\}$, $\forall m \in \mathcal{M}_{i,j}$.

Le modèle est le suivant :

$$\min C_{\max} \tag{4.9}$$

$$\text{t.q.} \quad Alternative(task_{i,j}, mode_{i,j,m} : \forall m \in \mathcal{M}_{i,j}) \quad \forall i \in \mathcal{J}, \forall j \in \{1, \ldots, n_i\} \tag{4.10}$$

$$EndBeforeStart(task_{i,j}, task_{i,j+1}) \quad \forall i \in \mathcal{J}, \forall j \in \{1, \ldots, n_i - 1\} \tag{4.11}$$

$$NoOverlap(mode_{i,j,m} : \forall O_{i,j} \in \mathcal{I}_m) \quad \forall m \in \mathcal{M} \tag{4.12}$$

$$C_{\max} \geq task_{i,n_i}.end \quad \forall i \in \mathcal{J} \tag{4.13}$$

La fonction objectif (4.9) consiste à minimiser le makespan. Les contraintes (4.10) assurent que chaque opération est affectée à exactement une machine éligible. Les contraintes (4.11) permettent de satisfaire les relations de précédence entre les opérations consécutives d'un même travail. Les contraintes (4.12) arbitrent les disjonctions entre opérations affectées à une même machine. Enfin, les contraintes (4.13) permettent de définir le makespan.

4.3 Décomposition de Benders basée sur la logique

Dans le chapitre 3, nous avons présenté un algorithme de décomposition de Benders basé sur la logique pour résoudre les problèmes de job-shop flexible. Nous avons alors supposé que les sous-problèmes étaient résolus par un oracle.

Dans ce paragraphe, nous proposons deux méthodes de résolution du sous-problème dans le cas non préemptif. Le sous-problème est un problème de job-shop classique. Pour le résoudre, de nombreuses méthodes ont été proposées dans la littérature. Nous présentons ici deux formulations mathématiques, l'une en PLNE et l'autre en PPC.

4.3.1 Programmation linéaire en nombres entiers

Des études comparatives ont montré que les modèles disjonctifs sont les plus efficaces pour résoudre les problèmes de job-shop ([Ku 2016, Meng 2020]). Ainsi, notre modèle est basé sur celui présenté dans [Manne 1960]. Il implique les variables de décision suivantes : $t_{i,j}$ la variable continue représentant la date de début de traitement de chaque opération $O_{i,j}$, $\forall i \in \mathcal{J}$, $\forall j \in \{1, \ldots, n_i\}$, $C_{\max}$ la variable continue représentant le makespan, ainsi que les variables de précédence binaires

suivantes :

$$y_{i,j,i',j'} = \begin{cases} 1 & \text{si } O_{i,j} \text{ traitée avant } O_{i',j'} \\ 0 & \text{sinon.} \end{cases} \quad \begin{array}{l} \forall (i,i') \in \mathcal{J}^2, \forall j \in \{1,\ldots,n_i\}, \\ \forall j' \in \{1,\ldots,n'_i\} \text{ t.q. } O_{i',j'} \neq O_{i,j}, \end{array}$$

Le modèle disjonctif est énoncé comme suit :

$$\min C_{\max} \tag{4.14}$$

$$\text{t.q.} \quad C_{\max} \geq t_{i,n_i} + p_{i,n_i} \qquad \forall i \in \mathcal{J} \tag{4.15}$$

$$t_{i,j+1} \geq p_{i,j} + t_{i,j} \qquad \forall i \in \mathcal{J}, \forall j \in \{1,\ldots,n_i - 1\} \tag{4.16}$$

$$t_{i,j} \geq t_{i',j'} + p_{i',j'} - y_{i,j,i',j'} H \qquad \forall m \in \mathcal{M}, \forall (O_{i,j}, O_{i,j'}) \in \mathcal{I}_m^2 \tag{4.17}$$

$$t_{i',j'} \geq t_{i,j} + p_{i,j} - (1 - y_{i,j,i',j'}) H \qquad \forall m \in \mathcal{M}, \forall (O_{i,j}, O_{i,j'}) \in \mathcal{I}_m^2 \tag{4.18}$$

4.3.2 Programmation par contraintes

Le modèle de PPC utilisé pour résoudre le sous-problème de JSSP est basé sur l'exemple illustratif fourni par *IBM ILOG CPLEX CP Optimizer* [IBM 2022], similaire au modèle PPC pour le FJSSP présenté dans le paragraphe 4.2.2.

En particulier, le modèle comprend une variable continue $C_{\max}$ qui représente le makespan, ainsi que des variables d'intervalle $task_{i,j}$ représentant l'intervalle entre le début et la fin de chaque opération $O_{i,j}$, $\forall i \in \mathcal{J}$, $\forall j \in \{1,\ldots,n_i\}$.

Il est formulé comme suit :

$$\min C_{\max} \tag{4.19}$$

$$\text{t.q.} \quad C_{\max} \geq task_{i,n_i}.end \qquad \forall i \in \mathcal{J} \tag{4.20}$$

$$EndBeforeStart(task_{i,j}, task_{i,j+1}) \qquad \forall i \in \mathcal{J},\ \forall j \in \{1,\ldots,n_i - 1\} \tag{4.21}$$

$$NoOverlap(task_{i,j} : \forall O_{i,j} \in \mathcal{P}_m^h) \qquad \forall m \in \mathcal{M} \tag{4.22}$$

L'objectif (4.19) et les contraintes (4.20) assurent la minimisation du makespan. Les contraintes (4.21) définissent les contraintes de précédence entre les opérations d'un même job. Les contraintes (4.22) interdisent les chevauchements d'opérations sur une machine.

4.4 Expérimentations numériques

Dans ce paragraphe, nous étudions les performances de différentes approches pour la résolution du problème de job-shop flexible non préemptif.

Toutes les expériences sont réalisées sur trois nœuds de cluster avec Intel Xeon E5-2695 v4 CPU à 2,1 GHz. Tous les algorithmes présentés sont implémentés en C++, en utilisant CPLEX 12.10 pour les modèles PLNE et CP Optimizer (CPO) 12.10 pour les modèles PPC. Le temps CPU et la RAM sont respectivement limités à 1 heure et 16 Go.

Les instances considérées ici sont les instances classiques de job-shop flexible présentées dans le paragraphe 1.4.3.

Les performances des différentes méthodes sont évaluées en termes d'écart d'optimalité, de nombre de solutions optimales trouvées, de nombre de meilleures solutions trouvées et de déviation par rapport à la meilleure solution.

Les résultats sont présentés en moyenne, par type de jeux d'instances afin de refléter de la manière la plus synthétique possible les performances des différentes méthodes testées sur une grande variété de problèmes.

4.4.1 Analyse des méthodes de décomposition de Benders basée sur la logique

Nous considérons deux méthodes de décomposition de Benders basée sur la logique, notée $LBBD_{PPC}$ et $LBBD_{PLNE}$ utilisant le schéma de décomposition présenté dans le chapitre 3. Les sous-problèmes sont respectivement résolus à l'aide du modèle de PPC présenté dans le paragraphe 4.3.2 et du modèle PLNE présenté dans le paragraphe 4.3.1.

Dans le tableau 4.1, nous indiquons, pour ces deux méthodes, la proportion de temps passé à résoudre les sous-problèmes ("SP(%)"), le nombre moyen de sous-problèmes résolus ("#SP"), et le temps moyen passé à résoudre un sous-problème ("temps/SP" en secondes) pour chaque jeu d'instances étudié.

Benchmark	$LBBD_{PLNE}$			$LBBD_{PPC}$		
	SP(%)	#SP	temps/SP(s)	SP(%)	#SP	temps/SP(s)
BrandimarteMk	100	4	700	75	2211	0.5
HurinkEdata	100	6	521	100	416	4.5
HurinkRdata	100	320	11.1	85	6209	0.4
HurinkVdata	100	189	16.6	40	6035	0.2
DPpaulli	100	1	3600	100	1	2173
ChambersBarnes	100	1	3600	100	2	1554
Kacem	71	60	0.3	1	60	0.1
Fattahi	89	373	1.4	0.5	1037	0.1

Tableau 4.1 – Répartition du temps de calcul pour les méthodes de décomposition, pour chaque jeu d'instances

La seule différence entre les deux méthodes est la façon de résoudre le sous-problème, le nombre d'itérations pour résoudre une instance de façon optimale est donc le même pour les deux méthodes (voir par exemple le jeu d'instances *Kacem* pour lequel toutes les instances sont résolues à l'optimum). Il a été démontré dans [Ku 2016] que la PPC est plus performante que la PLNE pour résoudre les problèmes de job-shop, que ce soit en termes de démonstration d'optimalité ou de qualité des solutions. Il n'est donc pas surprenant que le temps passé par sous-problème est beaucoup plus faible pour la méthode utilisant la PPC. De plus, pour

les instances, qui ne sont pas résolues de manière optimale, le nombre d'itérations est plus grand pour $LBBD_{PPC}$ car tant que la solution optimale n'est pas atteinte, la recherche se poursuit en itérant jusqu'à la limite de temps fixée. Nous remarquons également que pour tous les jeux d'instances, à l'exception de ceux dont les sous problèmes sont résolus très rapidement (*HurinkVdata*, *Kacem* et *Fattahi* pour la méthode $LBBD_{PPC}$) , la quasi-totalité du temps de calcul est consacrée à la résolution des sous-problèmes. Enfin, nous observons que pour les instances les plus compliquées (*DPpaulli* et *ChambersBarnes*, très peu de sous-problèmes sont considérés (1 seul dans la plupart des cas), ce qui signifie que, quelle que soit la méthode utilisée, la limite de temps imposée d'une heure n'est pas suffisante pour résoudre ces sous-problèmes de manière optimale.

Le tableau 4.2 présente les indicateurs de performance pour ces deux méthodes, à savoir la proportion d'instances pour lesquelles l'optimalité est prouvée (Opti. (%)), l'écart d'optimalité moyen (Gap(%)) ainsi que la proportion d'instances pour lesquelles la meilleures solutions réalisables est trouvées par chaque méthode.

Benchmark	$LBBD_{PLNE}$			$LBBD_{PPC}$		
	Opti. (%)	Gap(%)	Best (%)	Opti. (%)	Gap(%)	Best (%)
BrandimarteMk	7	11	7	60	3	100
HurinkEdata	11	10	16	47	5	100
HurinkRdata	0	14	6	30	8	98
HurinkVdata	12	5	62	39	3	98
DPpaulli	0	17	38	0	8	100
ChambersBarnes	0	12	0	0	9	100
Kacem	100	0	100	100	0	100
Fattahi	85	2	89	85	1	100

Tableau 4.2 – Performances des méthodes de décomposition

Pour tous les indicateurs de performance considérés et pour l'ensemble des jeux d'instances étudiés, il est clair que la méthode $LBBD_{PPC}$ est supérieure à la méthode $LBBD_{PLNE}$. Cette observation est cohérente avec le fait que la résolution des sous-problèmes par la PPC est plus efficace que par la PLNE. Par conséquent, pour la suite des résultats, nous nous concentrerons uniquement sur la résolution du sous-problème à l'aide de la PPC pour la méthode de décomposition.

Afin d'étudier l'impact de l'étape 2 sur les performances de l'algorithme de décomposition 1 (présenté dans le paragraphe 3.5), le tableau 4.3 liste pour les 17 instances du jeu d'instances *Fattahi*, résolues de manière optimale par $LBBD_{PPC}$, le nombre d'itérations (#itérations), le nombre de coupes ajoutées au problème maître (#coupes), le nombre de sous-problèmes résolus (#SP), ainsi que le temps passé à résoudre l'instance (temps (s)), avec et sans étape 2. Notons que ces résultats sont présentés pour ce petit sous-ensemble d'instances car nous considérons que cet échantillon est suffisant pour visualiser l'effet de l'étape 2 sur notre algorithme. En

outre, les informations étudiées ici ne sont pas pertinentes pour les instances dont la résolution a été interrompue par la limite de temps. Avec l'étape 2, le nombre

Instance	**sans étape 2**				**avec étape 2**			
	temps (s)	#itérations	#coupes	#SP	temps (s)	#itérations	#coupes	#SP
SFJS1	0	0	0	0	0	0	0	0
SFJS2	0	0	0	0	0	0	0	0
SFJS3	0	2	2	2	0	4	106	2
SFJS4	0	2	2	2	0	4	131	1
SFJS5	0	1	1	1	0	2	33	1
SFJS6	0	3	3	3	0	3	91	2
SFJS7	0	1	1	1	0	1	1	1
SFJS8	0	27	27	27	0	14	115	7
SFJS9	0	4	4	4	0	4	70	4
SFJS10	0	1	1	1	0	2	51	1
MFJS1	0	95	95	95	0	69	10164	41
MFJS2	0	618	618	618	0	211	14593	151
MFJS3	0	238	238	238	0	82	19480	37
MFJS4	16	3929	3929	3929	3	934	256259	739
MFJS5	0	371	371	371	0	73	61344	24
MFJS6	12	3225	3225	3225	1	817	67586	663
MFJS7	3117	45728	45728	45728	9	1640	431142	1209

Tableau 4.3 – Impact de l'étape 2 sur les performances de l'algorithme de décomposition 1 pour les instances *Fattahi* résolues à l'optimum

de sous-problèmes est toujours plus petit que le nombre d'itérations, lui-même plus petit que le nombre de coupes. En effet, à l'étape 2, si au moins une des bornes inférieures calculées est supérieure à la valeur actuelle de la meilleure solution, alors au moins une coupe est générée et aucun sous-problème n'est résolu ; sinon, exactement un sous-problème est résolu qui génère exactement une coupe. Enfin, pour chaque instance, nous observons qu'il y a moins d'itérations et donc moins de sous-problèmes résolus avec l'étape 2. Il n'y a pas de domination claire par l'une ou l'autre méthode en termes de nombre de coupes générées, mais ce facteur n'est pas significatif par rapport au nombre de sous-problèmes qui peuvent être évités avec l'étape 2. En effet, comme nous l'avons vu en étudiant le tableau 4.1, le temps passé à résoudre le problème maître est négligeable par rapport à celui passé dans le sous-problème. Ainsi, l'étape 2 permet d'éviter un nombre considérable de sous-problèmes, ce qui contribue à une meilleure efficacité globale de la méthode de décomposition.

4.4.2 Comparaison des méthodes

Nous étudions également la résolution du problème en utilisant le modèle de PPC décrit dans le paragraphe 4.2.2. Nous désignons cette méthode par PPC. Afin de garantir une comparaison équitable, nous proposons également une variante de cette méthode, appelée $warm\_PPC$, qui utilise une solution initiale obtenue à

l'aide de l'**Heuristique 1** décrite dans le paragraphe 3.5.

Nous ne reportons pas les résultats obtenus par la résolution à l'aide du modèle PLNE présenté dans le paragraphe 4.2.1, car, pour un grand nombre d'instances, le modèle n'est pas parvenu à obtenir une solution réalisable dans les conditions expérimentales fixées.

Les figures 4.1, 4.2, 4.3 et 4.4 offrent une visualisation graphique des indicateurs de performance.

Dans la figure 4.1, les écarts d'optimalité moyens pour chaque méthode et chaque jeu d'instances sont présentés. Nous observons qu'aucune des méthodes ne domine les autres sur l'ensemble des jeux d'instances. Cependant, on constate que les écarts d'optimalité sont plus modérés et réguliers pour la méthode $LBBD_{PPC}$. En effet, cet écart ne dépasse jamais les 9% pour cette méthode en moyenne sur l'ensemble des jeux d'instances, alors qu'il atteint jusqu'à 24% pour les méthodes PPC et $warm\_PPC$, pour les instances *DPpaulli*. Globalement, si nous considérons ensemble les 276 instances, les méthodes PPC et $warm\_PPC$ atteignent en moyenne un écart d'optimalité de 11% alors que la méthode de décomposition $LBBD_{PPC}$ obtient un gap moyen de 5%.

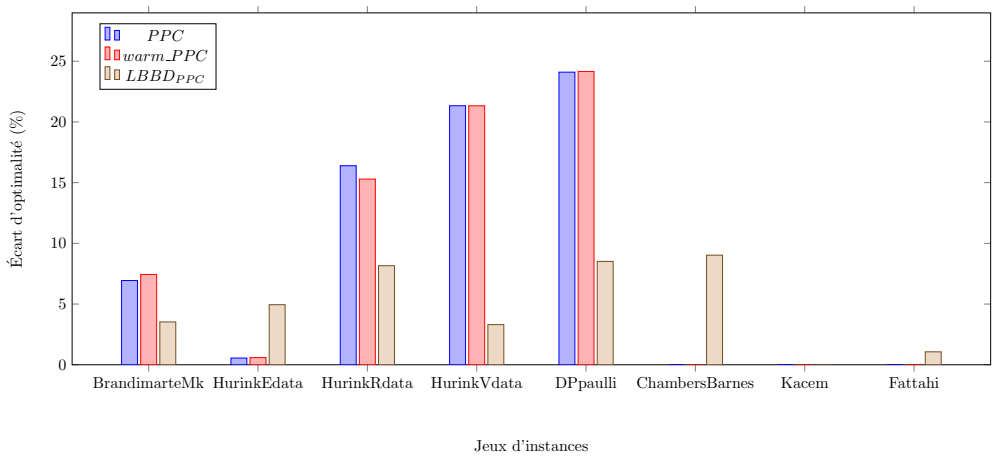


FIGURE 4.1 – Écarts d'optimalité moyen en fonction du jeu d'instances

La figure 4.2 illustre la proportion de solutions optimales trouvées pour chaque jeu d'instances, dans la limite de temps fixée. Pour cet indicateur, les méthodes PPC et $warm\_PPC$ dominent largement la méthode de décomposition. Dans l'ensemble, $warm\_PPC$ est capable de résoudre 175 instances à l'optimalité, suivie de 172 pour PPC et 107 pour $LBBD_{PPC}$. Il convient de souligner que la méthode de décomposition ne parvient pas à résoudre ne serait-ce qu'une instance pour les jeux d'instances *DPpaulli* et *ChambersBarnes*, alors que les méthodes PPC et $warm\_PPC$ parviennent à résoudre l'ensemble des instances *ChambersBarnes*. D'autre part, la méthode $LBBD_{PPC}$ obtient de meilleurs résultats que PPC et $warm\_PPC$ uniquement pour le jeu d'instances *Brandimarte*.

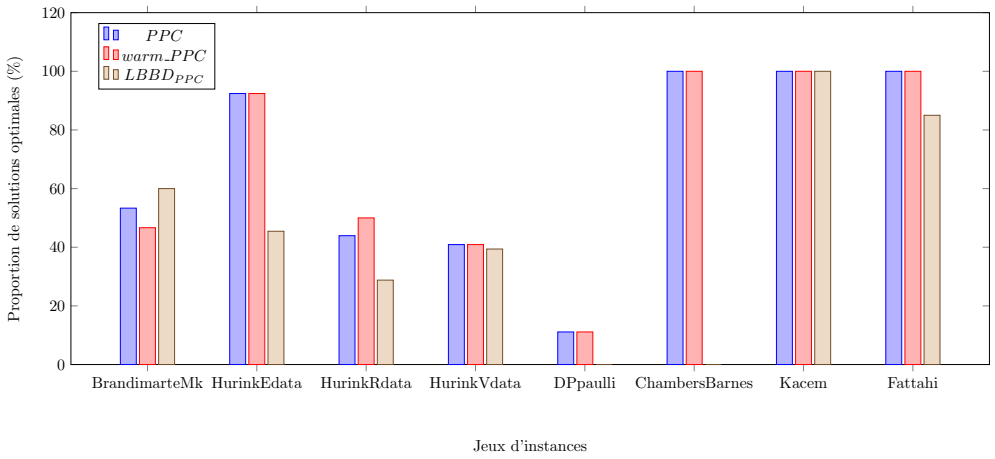


FIGURE 4.2 – Proportion de solutions optimales pour chaque jeu d'instance

Pour chaque instance, nous avons identifié la méthode (ou les méthodes) permettant d'obtenir la solution de meilleure qualité, c'est-à-dire celle avec le plus petit makespan. La figure 4.3 présente la proportion d'instances pour lesquelles chaque méthode fournit la meilleure solution.

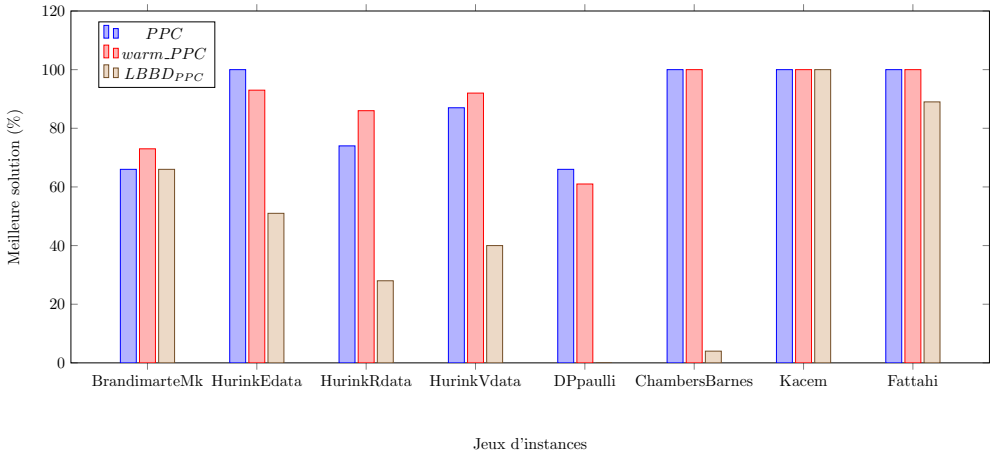


FIGURE 4.3 – Proportion de meilleures solutions trouvées pour chaque jeu d'instance

On constate que les méthodes utilisant uniquement la PPC obtiennent la meilleure solution pour la quasi-totalité des instances. Plus précisément, la méthode $warm_{PPC}$ parvient à fournir la meilleure solution pour 247 des 276 instances, tandis que la méthode PPC le fait pour 240 instances. En revanche, la méthode $LBBD_{PPC}$ ne parvient à fournir la meilleure solution que pour 113 instances.

Au total, la méthode de décomposition obtient une solution strictement meilleure que les autres méthodes pour un nombre limité d'instances, seulement 8, sur l'en-

semble des jeux d'instances.

La figure 4.4 présente, pour chaque méthode, le pourcentage de déviation relative moyenne par rapport à la meilleure solution connue. Pour une instance ι et une méthode μ, elle est calculée comme suit :

$$ER_\iota^\mu = \frac{BS_\iota^\mu - BS_\iota^*}{BS_\iota^*} \tag{4.23}$$

où BS_ι^* désigne la meilleure valeur de la fonction objectif trouvée pour l'instance ι et BS_ι^μ la valeur du makespan obtenu par la méthode μ pour cette instance. Sans

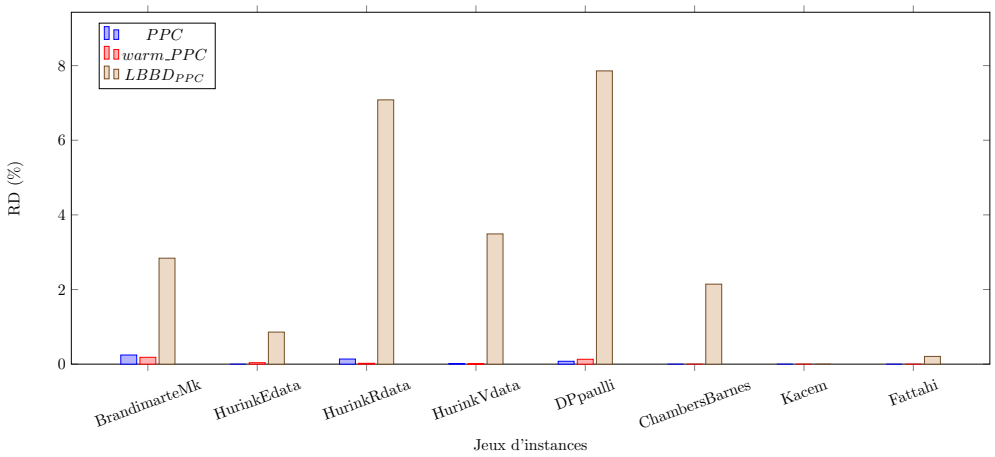


FIGURE 4.4 – Écart relatif à la meilleure solution moyenne pour chaque jeu d'instances

surprise, au regard des résultats présentés dans la figure 4.3, la déviation relative moyenne est très faible pour les méthodes PPC et $warm\_PPC$. Cette observation est cohérente avec le fait que ces méthodes obtiennent la meilleure solution pour une grande majorité des instances. En revanche, pour la méthode de décomposition, elle peut atteindre jusqu'à près de 8% pour le jeu d'instances *DPpaulli*, ce qui montre que les solutions obtenues à l'aide de cette méthode sont parfois de mauvaise qualité.

L'ensemble de ces résultats soulignent la supériorité des méthodes basées sur la PPC pure en termes de qualité des solutions trouvées. Cependant, la méthode de décomposition proposée, qui utilise la PPC pour résoudre le sous-problème ($LBBD_{PPC}$), parvient à fournir une meilleure solution réalisable pour quelques instances (8 au total) et, de manière générale, des bornes inférieures de bonne qualité.

La méthode $LBBD_{PPC}$ permet même d'améliorer certaines bornes inférieures et supérieures de la littérature. Celles-ci sont répertoriées dans le tableau 4.4.

En particulier, la méthode $LBBD_{PPC}$ permet d'améliorer 11 bornes inférieures et 8 bornes supérieures connues pour des instances de la littérature. De plus, elle permet de prouver l'optimalité pour la première fois pour 7 de ces instances.

Jeu	**Instance**	BI	BS
BrandimarteMk	mk6	**51**	-
BrandimarteMk	mk11	**609**	**609***
BrandimarteMk	mk13	**381**	**390**
HurinkRdata	la23	**817**	-
HurinkRdata	la24	**775**	-
HurinkRdata	abz7	**497**	-
HurinkRdata	abz8	**508**	-
HurinkRdata	abz9	**501**	-
HurinkRdata	car1	**5034**	**5034***
HurinkRdata	car2	**5985**	**5985***
HurinkRdata	car3	**5622**	**5622***
HurinkVdata	car1	-	**5005***
HurinkVdata	car2	-	**5929***
HurinkVdata	car4	-	**6514***

Tableau 4.4 – Bornes améliorées par la méthode de décomposition de Benders pour le job-shop flexible

4.5 Conclusion

Dans ce chapitre, nous présentons plusieurs méthodes exactes pour résoudre le FJSSP avec minimisation du makespan. En particulier, nous appliquons le modèle de décomposition de Benders basée sur la logique présenté dans le chapitre 3 à ce problème.

Bien que cette décomposition ne soit pas la méthode la plus performante pour résoudre le problème de job-shop flexible non préemptif, elle offre des avantages tels que la génération de bornes inférieures de qualité et des améliorations par rapport à l'état de l'art pour certaines instances. La PPC pure reste néanmoins plus adaptée pour obtenir des solutions optimales, ou du moins de bonne qualité, de manière efficace.

Chapitre 5

Le problème de job-shop flexible préemptif

Sommaire

5.1 Introduction

Dans ce chapitre, on s'intéresse au problème de job-shop flexible préemptif. Le traitement d'une opération peut alors être interrompu pendant un temps, puis reprendre là où il s'est arrêté sur la même machine. Ces situations sont courantes, par exemple lorsque les ressources sont des ordinateurs ou un ensemble de travailleurs. Quand la préemption est possible, d'un point de vue technique, celle-ci doit être envisagée, car elle offre souvent la possibilité d'améliorer un critère tel que le makespan [Creemers 2019].

Cependant, la préemption rend les problèmes de job-shop plus difficiles à résoudre. En effet, Le problème de job-shop préemptif (pJSSP), avec recirculation est *NP-difficile* dès deux machines et trois travaux [Brucker 1999] alors que la version non préemptive du problème peut être résolue en temps polynomial [Kravchenko 1995].

La prise en compte conjointe de la préemption et de la flexibilité des ressources complique donc encore le problème classique déjà difficile à résoudre. Par conséquent, les études sur le problème de job-shop flexible et préemptif (pFJSSP) sont rares. Trois métaheuristiques sont présentées dans [Zhang 2016] pour le FJSSP avec

des caractéristiques spéciales (jours de travail flexibles, chevauchement et possibilités de préemption limitées), tandis que [Jansen 2005] a proposé plusieurs algorithmes d'approximation pour le FJSSP lorsque le nombre de machines et le nombre maximal d'opérations par travail sont fixes, dont un pour le cas particulier du pFJSSP. Ces caractéristiques particulières et les méthodes d'approximation sortent du cadre de ce chapitre.

À notre connaissance, aucune méthode exacte n'a été rapportée dans la littérature pour résoudre ce problème. L'objectif de ce chapitre est donc de développer et de comparer plusieurs méthodes exactes pour le pFJSSP avec minimisation du makespan. Dans un premier temps, nous présentons des formulations mathématiques utilisant la PLNE (5.2.1) et la PPC (5.2.2) et permettant une résolution directe du problème. Ensuite, dans le paragraphe 5.3, nous abordons des méthodes de résolution du sous-problème dans le cadre d'une approche utilisant l'algorithme de décomposition de Benders basée sur la logique exposé dans le chapitre 3. Enfin, nous présentons les résultats des expérimentations numériques dans le paragraphe 5.4.

5.2 Formulations directes

Dans ce paragraphe, nous présentons deux formulations mathématiques directes pour la résolution du problème de job-shop flexible préemptif. Contrairement à la méthode de décomposition présentée dans le chapitre 3, ces modèles permettent de résoudre le problème en une seule itération en déterminant ensemble l'affectation et l'ordonnancement des opérations sur les machines.

5.2.1 Programmation linéaire en nombres entiers

Le modèle de PLNE présenté dans ce paragraphe est basé sur une formulation indexée sur le temps proposée dans [Bowman 1959] pour résoudre le problème de job-shop préemptif. Il est adapté ici pour y intégrer la notion de flexibilité des ressources.

On définit $\mathcal{T} = \{1, 2, 3, \ldots, \tau\}$ un horizon de temps discret, ainsi que les variables de décision suivantes :

$$x_{i,j,m} = \begin{cases} 1 & \text{si } O_{i,j} \text{ affectée à la machine } m \\ 0 & \text{sinon.} \end{cases} \quad \forall i \in \mathcal{J}, \forall j \in \{1, \ldots, n_i\}, \forall m \in \mathcal{M}_{i,j}$$

et

$$y_{i,j,t} = \begin{cases} 1 & \text{si l'opération } O_{i,j} \text{ en cours à l'instant } t \\ 0 & \text{sinon.} \end{cases} \quad \forall i \in \mathcal{J}, \forall j \in \{1, \ldots, n_i\}, \forall t \in \mathcal{T}$$

Le modèle mathématique est le suivant :

$$\min C_{\max} \tag{5.1}$$

$$\text{t.q.} \quad \sum_{m \in \mathcal{M}_{i,j}} x_{i,j,m} = 1 \qquad \forall i \in \mathcal{J},\ \forall j \in \{1, \ldots, n_i\} \tag{5.2}$$

$$\sum_{t=1}^{\tau} y_{i,j,t} \geq \sum_{m \in \mathcal{M}_{i,j}} x_{i,j,m} p_{i,j,m} \qquad \forall i \in \mathcal{J},\ \forall j \in \{1, \ldots, n_i\} \tag{5.3}$$

$$\sum_{t'=t}^{\tau} y_{i,j,t'} \leq \max_{m \in \mathcal{M}_{i,j}} p_{i,j,m}(1 - y_{i,j+1,t}) \qquad \forall i \in \mathcal{J},\ \forall j \in \{1, \ldots, n_i - 1\},\ \forall t \in \mathcal{T} \tag{5.4}$$

$$\sum_{i \in \mathcal{J}} \sum_{j=1}^{n_i} x_{i,j,m} y_{i,j,t} \leq 1 \qquad \forall m \in \mathcal{M},\ \forall t \in \mathcal{T} \tag{5.5}$$

$$C_{\max} \geq (t+1) y_{i,n_i,t} \qquad \forall i \in \mathcal{J},\ \forall t \in \mathcal{T} \tag{5.6}$$

$$x_{i,j,m} \in \{0,1\} \qquad \forall i \in \mathcal{J},\ \forall j \in \{1, \ldots, n_i\}, \forall m \in \mathcal{M}_{i,j} \tag{5.7}$$

$$y_{i,j,t} \in \{0,1\} \qquad \forall i \in \mathcal{J},\ \forall j \in \{1, \ldots, n_i\},\ \forall t \in \mathcal{T} \tag{5.8}$$

L'objectif (5.1) est la minimisation du makespan. Les contraintes (5.2) garantissent que chaque opération est traitée par exactement une des machines éligibles. La durée de traitement de chaque opération est respectée grâce aux contraintes (5.3). Les contraintes (5.4) expriment les contraintes de précédence entre les opérations d'un même travail. Soit $i \in \mathcal{J}$ un travail, $O_{i,j}$ et $O_{i,j+1}$ deux opérations successives et t une période donnée. Si l'opération $O_{i,j+1}$ est traitée à l'instant t (c'est-à-dire $y_{i,j+1,t} = 1$), alors l'opération $O_{i,j}$ doit être achevée avant t (c'est-à-dire qu'elle ne doit plus être traitée à partir de t, $\forall t' \geq t$, $y_{i,j,t'} = 0$). Les contraintes (5.5) sont les contraintes disjonctives ; à chaque instant t, une seule opération est en cours de traitement sur chaque machine m. Les contraintes (5.6) permettent de définir le makespan comme la fin de l'exécution de la dernière opération. Enfin, les contraintes (5.7) et (5.8) précisent la nature binaire des variables disjonctives et d'affectation.

Les contraintes (5.5) ne sont pas linéaires. Elles peuvent cependant être facilement linéarisées puisque les variables $x_{i,j,m}$ et $y_{i,j,t}$ sont binaires. Pour cela, on définit l'ensemble de variables binaires suivant :

$$z_{i,j,m,t} = \begin{cases} 1 & \text{si l'opération } O_{i,j} \text{ traitée par la machine } m \text{ à l'instant } t, \\ 0 & \text{sinon.} \end{cases}$$

Ainsi, les contraintes (5.5) peuvent être remplacées par l'ensemble de contraintes

suivant :

$$z_{i,j,m,t} \geq x_{i,j,m} + y_{i,j,t} - 1 \quad \forall i \in \mathcal{J}, \forall j \in \{1, \ldots, n_i\}, \forall m \in \mathcal{M}_{i,j}, \forall t \in \mathcal{T} \tag{5.9}$$

$$z_{i,j,m,t} \leq x_{i,j,m} \quad \forall i \in \mathcal{J}, \forall j \in \{1, \ldots, n_i\}, \forall m \in \mathcal{M}_{i,j}, \forall t \in \mathcal{T} \tag{5.10}$$

$$z_{i,j,m,t} \leq y_{i,j,t} \quad \forall i \in \mathcal{J}, \forall j \in \{1, \ldots, n_i\}, \forall m \in \mathcal{M}_{i,j}, \forall t \in \mathcal{T} \tag{5.11}$$

$$\sum_{i \in \mathcal{J}} \sum_{j=1}^{n_i} z_{i,j,m,t} \leq 1 \quad \forall m \in \mathcal{M}, \forall t \in \mathcal{T} \tag{5.12}$$

Notons que [Meng 2020] ont montré que le modèle basé sur les variables de séquence est le plus efficace pour résoudre le problème de job-shop flexible (sans préemption). Toutefois, ce modèle ne peut pas être appliqué au problème préemptif. En effet, dans ce cas, une solution ne peut pas être totalement décrite par une séquence d'opérations sur chaque machine.

La résolution de ce modèle est très coûteuse en termes de temps de calcul pour les instances de grande taille, car sa dimension dépend de l'horizon de temps en plus du nombre d'opérations et de machines. Les résultats expérimentaux montrent que seules de très petites instances du problème peuvent être résolues. Pour de nombreuses instances utilisées dans le paragraphe 5.4, aucune solution réalisable n'est trouvée. Par conséquent, l'utilisation directe du modèle de PLNE ne sera pas consignée dans les résultats numériques.

5.2.2 Programmation par contraintes

Le modèle de PPC décrit ici utilise les fonctionnalités de modélisation offertes par *IBM CP Optimizer solver*. En particulier, nous utilisons des variables d'intervalle. Il y a deux manières de modéliser une opération préemptive de durée p à l'aide des variables d'intervalle ([Polo-Mejía 2020]) :

1) par un ensemble de variables d'intervalle dont la durée n'est pas fixée, en imposant que la somme de leurs durées soit égale à p ;
2) par une séquence de p variables d'intervalle de durée unitaire.

Ces deux possibilités sont illustrées dans la figure 5.1 pour une opération de durée 3. Ces deux illustrations représentent la même solution, à savoir l'exécution de la tâche durant deux unités de temps, une interruption, puis la fin de traitement de la tâche pour une unité de temps.

FIGURE 5.1 – Deux représentations possibles d'une solution de découpage avec des variables d'intervalle pour une opération préemptive de durée 3

Pour le problème d'ordonnancement de projet sous contraintes de ressources avec compétences multiples (*multi-skill resource-constrained project scheduling pro-*

blem), [Polo-Mejía 2020] montre que la seconde approche est plus efficace. Nous avons également mené des expériences conduisant à cette conclusion pour notre problème spécifique.

Ainsi, chaque opération préemptive est décomposée en parties de durée unitaire dans le modèle de PPC proposé. Nous introduisons les variables de décision suivantes :

— $task_{i,j}$: variable d'intervalle entre le début et la fin de l'opération $O_{i,j}$,
— $mode_{i,j,m}$: variable d'intervalle optionnelle entre le début et la fin de l'opération $O_{i,j}$ sur la machine m,
— $part_{i,j,k,m}$: variable d'intervalle optionnelle de durée unitaire représentant le traitement de la $k^{ième}$ partie de l'opération $O_{i,j}$ sur la machine m.

Le modèle est le suivant :

$$\min C_{\max} \tag{5.13}$$

$$\text{t.q.} \quad C_{\max} \geq task_{i,n_i}.end \qquad \forall i \in \mathcal{J} \tag{5.14}$$

$$EndBeforeStart(task_{i,j}, task_{i,j+1}) \qquad \forall i \in \mathcal{J}, \ \forall j \in \{1, \ldots, n_i - 1\} \tag{5.15}$$

$$EndBeforeStart(part_{i,j,k,m}, part_{i,j,k+1,m}) \qquad \forall i \in \mathcal{J}, \ \forall j \in \{1, \ldots, n_i\}, \ \forall m \in \mathcal{M}_{i,j}, \ \forall k \in 1, \ldots, p_{i,j,m} - 1 \tag{5.16}$$

$$Span(mode_{i,j,m}, part_{i,j,k,m} : \forall k \in 1, \ldots, p_{i,j,m}) \qquad \forall i \in \mathcal{J}, \ \forall j \in \{1, \ldots, n_i\}, \ \forall m \in \mathcal{M}_{i,j} \tag{5.17}$$

$$Alternative(task_{i,j}, mode_{i,j,m} : \forall m \in \mathcal{M}_{i,j}) \qquad \forall i \in \mathcal{J}, \ \forall j \in \{1, \ldots, n_i\} \tag{5.18}$$

$$PresenceOf(mode_{i,j,m}) \Rightarrow PresenceOf(part_{i,j,k,m}) \qquad \forall i \in \mathcal{J}, \ \forall j \in \{1, \ldots, n_i\}, \ \forall m \in \mathcal{M}_{i,j}, \ \forall k \in 1, \ldots, p_{i,j,m} \tag{5.19}$$

$$NoOverlap(part_{i,j,k,m} : \forall O_{i,j} \in \mathcal{I}_m, \ k \in 1, \ldots, p_{i,j,m}) \qquad \forall m \in \mathcal{M} \tag{5.20}$$

La fonction objectif (5.13) consiste à minimiser le makespan. Les contraintes (5.14) permettent de définir le makespan comme la fin de l'exécution de la dernière opération. Les contraintes (5.15) garantissent que les opérations d'un même travail seront traitées en respectant les contraintes de précédence sur les travaux. Les contraintes (5.16) ont pour but d'ordonner les parties de chaque opération et évitent ainsi des symétries. Les contraintes (5.17) et (5.19) permettent de faire le lien entre les variables $part_{i,j,k,m}$ et $mode_{i,j,m}$. Avec les contraintes (5.19), si la variable $mode_{i,j,m}$ est présente, alors l'opération $O_{i,j}$ est traitée sur le machine m et l'ensemble des parties $part_{i,j,k,m}$ $(1 \leq k \leq p_{i,j,m})$ doivent être présentes pour assurer la totalité de la durée de traitement de l'opération. Dans ce même cas,

les contraintes (5.17) assurent que l'intervalle représenté par la variable $mode_{i,j,m}$ s'étend sur l'ensemble des parties $part_{i,j,k,m}$ (c'est-à-dire commence avec $part_{i,j,k,1}$ et termine avec $part_{i,j,k,p_{i,j,m}}$). Les contraintes (5.18) utilisent la contrainte globale *Alternative* qui garantit que chaque opération sera traitée par exactement une des machines éligibles, c'est-à-dire qu'un seul $mode_{i,j,m}$ est présent dans la solution pour chaque opération $O_{i,j}$ et que l'intervalle représenté par la variable $task_{i,j}$ commence et se termine en même temps que ce mode. Avec les contraintes (5.20), les contraintes de disjonction sur les machines sont assurées.

5.3 Décomposition de Benders basée sur la logique

Dans le chapitre 3, nous avons présenté un schéma de décomposition de Benders basée sur la logique pour les problèmes de job-shop flexible. Nous proposons ici d'appliquer cette méthode aux problèmes préemptifs.

Pour les résoudre, nous présentons d'abord un modèle de CP, puis nous décrivons un algorithme de branch-and-bound qui trouve son origine dans [Ebadi 2013].

5.3.1 Programmation par contraintes

Ce modèle est similaire à celui proposé dans le paragraphe 5.2.2. Les variables intervenant dans ce modèle sont les suivantes :

— $task_{i,j}$: variable d'intervalle entre le début et la fin de l'opération $O_{i,j}$,
— $part_{i,j,k}$: variable d'intervalle de durée unitaire représentant le traitement de la $k^{ième}$ partie de l'opération $O_{i,j}$.

On note que les variables $mode_{i,j,m}$ n'apparaissent pas dans ce sous-problème car l'ensemble des affectations est déjà fixées par le problème maître. De même, les variables $part_{i,j,k}$ ne sont pas indexées par m et ne sont pas optionnelles.

Le sous-problème est décrit comme suit :

$$\min z^h \tag{5.21}$$

$$\text{t.q.} \quad z^h \geq task_{i,n_i}.end \qquad \forall i \in \mathcal{J} \tag{5.22}$$

$$EndBeforeStart(task_{i,j}, task_{i,j+1}) \qquad \forall i \in \mathcal{J},\ \forall j \in \{1,\dots,n_i-1\} \tag{5.23}$$

$$EndBeforeStart(part_{i,j,k}, part_{i,j,k+1}) \qquad \forall i \in \mathcal{J},\ \forall j \in \{1,\dots,n_i\},\ \forall k \in 1,\dots,p^h_{i,j}-1 \tag{5.24}$$

$$Span(task_{i,j}, part_{i,j,k} : \forall k \in 1,\dots,p^h_{i,j}) \qquad \forall i \in \mathcal{J},\ \forall j \in \{1,\dots,n_i\} \tag{5.25}$$

$$NoOverlap(part_{i,j,k} : \forall O_{i,j} \in \mathcal{P}^h_m,\ \forall k \in 1,\dots,p^h_{i,j}) \qquad \forall m \in \mathcal{M} \tag{5.26}$$

où $p^h_{i,j} = \sum_{m \in \mathcal{M}_{i,j}} p_{i,j,m} x^h_{i,j,m}$.

5.3.2 Algorithme de branch-and-bound

Ebadi et Moslehi ont développé un algorithme de branch-and-bound pour le problème de job-shop préemptif [Ebadi 2013]. Cette méthode peut donc être utilisée pour la résolution du sous-problème.

Pour cette méthode, chaque opération $O_{i,j}$ est divisée en sous-opérations de durée unitaire $\{O_{i,j,1}, \ldots, O_{i,j,p_{i,j}}\}$. Pour présenter les principes clés de l'algorithme, nous utilisons les notations supplémentaires suivantes :

α	niveau actuel dans l'arbre de recherche,
σ_α	ordonnancement partiel au niveau α,
$\mathcal{V}_\alpha$	ensemble des opérations de durée unitaire disponibles selon σ_α,
$ES_{i,j,k}$	date de disponibilité de l'opération $O_{i,j,k}$,
ES_α	plus petite date de disponibilité des opérations de $\mathcal{V}_\alpha$, $ES_\alpha = \min_{O_{i,j,k} \in \mathcal{V}_\alpha} ES_{i,j,k}$,
m_α	machine sélectionnée au niveau α,
$\mathcal{V}'_\alpha$	sous-ensemble des opérations de $\mathcal{V}_\alpha$ traitées par la machine m_α et disponibles à la date ES_α,
$M_{i,j}$	machine traitant l'opération $O_{i,j}$.

Les auteurs ont montré que l'ensemble des ordonnancements sans délai pour lesquels les séquences sur chaque machine suivent la règle de Jackson préemptive (*preemptive EDD*) est dominant. Les deux règles de dominance suivantes en sont déduites :

Règle de dominance 1 *Quand une opération de durée unitaire $O_{i,j,k} \in \mathcal{V}'_\alpha$ est sélectionnée et ajoutée à l'ordonnancement partiel σ_α, alors la totalité de l'opération $O_{i,j}$ doit être traitée avant le début de n'importe quelle opération de durée unitaire de l'ensemble $\mathcal{V}'_\alpha \setminus \{O_{i,j,k}\}$. Ainsi, une relation de précédence est ajoutée entre l'opération $O_{i,j}$ et l'ensemble des opérations de durée unitaire de $\mathcal{V}'_\alpha \setminus \{O_{i,j,k}\}$.*

La seconde règle de dominance permet de calculer une borne inférieure sur la durée pendant laquelle une opération peut être traitée sans préemption. Cette règle est basée sur le principe selon lequel une interruption ne peut avoir lieu que lorsque qu'une nouvelle opération devient disponible.

Règle de dominance 2 *Quand une opération de durée unitaire $O_{i,j,k} \in \mathcal{V}'_\alpha$ est sélectionnée et ajoutée à l'ordonnancement partiel σ_α, le temps pendant lequel le traitement de l'opération $O_{i,j}$ peut se poursuivre sans préemption est supérieur à X, avec*

$$X = \min(p_{i,j} - k + 1, \min_{i' \in \mathcal{J} \setminus \{i\}} (x(i')) \tag{5.27}$$

et $x(i')$ une borne inférieure du temps pendant lequel le travail i' ne peut pas interrompre le traitement de l'opération $O_{i,j}$. La valeur de $x(i')$ peut être obtenue comme suit.

Soit $O_{i',j',k'} \in \mathcal{V}_\alpha$ la prochaine opération de durée unitaire disponible du travail i'.

• *Si $M_{i',j'} = m_\alpha$, alors :*

$$x(i') = \begin{cases} +\infty & \text{s'il existe une relation de précédence entre } O_{i,j} \\ & \text{et } O_{i',j',k'} \text{ induite par la règle de dominance 1,} \\ ES_{i',j',k'} - ES_\alpha & \text{sinon.} \end{cases} \tag{5.28}$$

• *Si $M_{i',j'} \neq m_\alpha$, soit j'' tel que $M_{i',j''} = m_\alpha$, alors :*

$$x(i') = \begin{cases} (ES_{i',j',k'} - ES_\alpha) + (p_{i',j'} - k' + 1) + \sum_{j'<j'''<j''} p_{i',j'''} & \text{si } j' < j'', \\ +\infty & \text{sinon.} \end{cases} \tag{5.29}$$

Notons que [Ebadi 2013] suppose que chaque travail visite chaque machine au plus une fois (en ordonnancement d'atelier, cette hypothèse est connue sous le nom de *non recirculation*). Cependant, cette hypothèse ne convient pas lorsqu'il s'agit de résoudre un sous-problème du job-shop flexible. En effet, pendant la phase d'affectation, rien n'empêche que deux opérations d'un même travail soient effectuées sur la même machine (dans l'exemple 4 du chapitre 3, les opérations $O_{2,1}$ et $O_{2,3}$ du travail J_2 sont affectées à la même machine M_1). Par conséquent, nous avons adapté la règle de dominance 2 à notre problème spécifique. Pour cela, il suffit de redéfinir j'' utilisé dans l'équation (5.29) comme suit :

$$j'' = \begin{cases} \min(\{j''' \in [1, n_i] \mid j''' > j',\ M_{i',j'''} = m_\alpha\}) & \text{si } \{j''' \in [1, n_i] \mid j''' > j', \\ & M_{i',j'''} = m_\alpha\} \neq \emptyset, \\ 0 & \text{sinon.} \end{cases} \tag{5.30}$$

La recirculation des opérations sur les machines n'affecte pas la règle de dominance 1, qui reste donc valable pour notre problème sans aucune modification.

L'algorithme 2 décrit la procédure incluant ces règles. Il commence au niveau $\alpha = 0$ avec un ordonnancement vide $\sigma_0 = \{\}$, l'ensemble $\mathcal{V}_0$ des opérations de durée unitaire disponibles est formé avec la première opération de durée unitaire de chaque travail. À chaque itération α, la machine m_α qui traite l'opération de durée unitaire dont la date de disponibilité est la plus précoce ES_α (ou l'une de ces machines, s'il y en a plusieurs) est sélectionnée par la fonction *SelectionnerUneMachineParmi()*. L'ensemble des opérations de durée unitaire disponibles pour être traitées à ES_α sur la machine m_α est désigné par $\mathcal{V}'_\alpha$. Pour chaque opération dans $\mathcal{V}'_\alpha$, un nouveau nœud est créé représentant un sous-ensemble de solutions avec un arc disjonctif entre la dernière opération de durée unitaire $O_{i,j,p_{i,j}}$ de l'opération $O_{i,j}$ et l'ensemble des autres opérations de durée unitaire dans $\mathcal{V}'_\alpha$ (*règle de dominance 1*), la date de disponibilité de ces opérations est mise à jour. Une limite inférieure X du temps pendant lequel le traitement de l'opération $O_{i,j}$ peut se poursuivre sans préemption

Algorithme 2 Algorithme de branch-and-bound pour le problème de job-shop flexible préemptif [Ebadi 2013]

Initialisation :

$\alpha \leftarrow 0$, $\sigma_\alpha \leftarrow \{\}$, $V_0 \leftarrow \{O_{i,1,1} \,|\, i \in \mathcal{J}\}$, $BS \leftarrow +\infty$
pour tout $i \in \mathcal{J}$ **faire**
  $ES_{i,0,0} \leftarrow 0$

Étape 1 Sélection de la machine et de la date de début :

$ES_\alpha \leftarrow \min(\{ES_{i,j,k} \,|\, i \in \mathcal{J},\ O_{i,j,k} \in \mathcal{V}_\alpha\})$
$m_\alpha \leftarrow SelectionnerUneMachineParmi(\{m \in \mathcal{M} \,|\, \exists O_{i,j,k} \in \mathcal{V}_\alpha,\ m_{i,j} = m,\ ES_{i,j,k} = ES_\alpha\})$
$\mathcal{V}'_\alpha \leftarrow \{O_{i,j,k} \in \mathcal{V}_\alpha \,|\, M_{i,j,k} = m_\alpha, ES_{i,j,k} = ES_\alpha\}$

Étape 2 Branchement :

pour tout $O_{i,j,k} \in \mathcal{V}'_\alpha$ **faire**
  pour tout $O_{i',j',k'} \in \mathcal{V}'_\alpha \setminus \{O_{i,j,k}\}$ **faire**
    $O_{i',j',k'}$ débute après la fin de $O_{i,j}$; ▷ *Règle de dominance 1*
    $ES_{i',j',k'} \leftarrow ES_{i,j,k} + p_{i,j} - k + 1$
  $X \leftarrow \min(p_{i,j} - k + 1, \min_{i' \in \mathcal{J} \setminus \{i\}}(x(i'))$; ▷ *Règle de dominance 2*
  $\sigma_{\alpha+1} \leftarrow \sigma_\alpha \cup \{O_{i,j,k}, \ldots, O_{i,j,k+X-1}\}$
  $\mathcal{V}_{\alpha+1} \leftarrow (\mathcal{V}_\alpha \setminus \{O_{i,j,k}\})$
  si $X < p_{i,j} - k + 1$ **alors**
    $\mathcal{V}_{\alpha+1} \leftarrow \mathcal{V}_{\alpha+1} \cup \{O_{i,j,k+X}\}$
    $ES_{i,j,k+X} \leftarrow ES_{i,j,k} + X$
  sinon
    si $j < n_i$ **alors**
      $\mathcal{V}_{\alpha+1} \leftarrow \mathcal{V}_{\alpha+1} \cup \{O_{i,j+1,0}\}$
      $ES_{i,j+1,0} \leftarrow ES_{i,j,k} + X$
  si $\mathcal{V}_{\alpha+1} = \{\}$ **alors**
    $BS \leftarrow \max_{i' \in \mathcal{J}}(ES_{i',n'_i,p_{i',n'_i}} + 1)$
  sinon
    $BI \leftarrow CalculerBorneInférieure()$
    si $BI < BS$ **alors**
      $\alpha \leftarrow \alpha + 1$
      Retour à **Étape 1** ;
Retourner BS

est calculée (*règle de dominance 2*). L'opération $O_{i,j,k}$ et les $(X-1)$ opérations de durée unitaire suivantes sont ordonnancées à partir de ES_α sur la machine m_α. L'ensemble des opérations de durée unitaire disponibles est mis à jour ; s'il est vide, une solution admissible du problème a été trouvé et la borne supérieure est mise à jour ; sinon, une borne inférieure BI de l'ordonnancement partiel est calculée pour évaluer ce nouveau nœud. Si BI est supérieure à la valeur de la fonction objectif de la meilleure solution obtenue jusqu'à présent, le nœud est abandonné et on passe à l'opération de durée unitaire de $\mathcal{V}'_\alpha$ suivante. Dans le cas contraire, l'algorithme passe au niveau suivant.

5.4 Expérimentations numériques

Dans ce paragraphe, nous examinons les performances des différentes approches présentées dans ce chapitre pour la résolution du problème de job-shop flexible préemptif. Pour ce faire, nous utilisons les instances classiques de job-shop flexible présentées dans le paragraphe 1.4.3, dans les conditions expérimentales exposées dans le paragraphe 4.4.

Les performances des différentes méthodes sont évaluées en termes d'écart d'optimalité, de nombre de solutions optimales trouvées, de nombre de meilleures solutions trouvées et de déviation par rapport à la meilleure solution connue.

Les résultats sont présentés en moyenne, par jeu d'instances afin de refléter synthétiquement les performances des différentes méthodes testées sur une grande variété de problèmes.

Dans un premier temps, nous évaluons les performances de l'algorithme de décomposition 1, appliqué au problème préemptif. Nous considérons dans un second temps la résolution à l'aide du modèle de PPC présenté dans le paragraphe 5.2.2 afin de comparer ces deux méthodes.

5.4.1 Analyse des méthodes de décomposition de Benders basée sur la logique

Nous considérons deux méthodes de décomposition de Benders basée sur la logique, notées $LBBD_{PPC}$ et $LBBD_{B\&B}$, utilisant le schéma de décomposition présenté dans le chapitre 3. Dans la méthode $LBBD_{PPC}$, le sous-problème est résolu à l'aide du modèle de PPC présenté dans le paragraphe 5.3.1. Dans la méthode $LBBD_{B\&B}$, il est résolu à l'aide de l'algorithme de branch-and-bound présenté dans le paragraphe 5.3.2.

Dans le tableau 5.1, nous indiquons, pour ces deux méthodes, la proportion du temps passé à résoudre les sous-problèmes ("SP(%)"), le nombre moyen de sous-problèmes résolus ("#SP"), et le temps moyen passé à résoudre un sous-problème ("temps/SP" en secondes) pour chaque jeu d'instances étudié.

Il est intéressant de noter que le temps moyen consacré à la résolution des sous-problèmes est considérablement plus faible pour la méthode $LBBD_{B\&B}$ que pour la méthode $LBBD_{PPC}$. Cela confirme l'efficacité de l'algorithme de branch-and-bound

Jeu d'instances	$LBBD_{PPC}$			$LBBD_{B\&B}$		
	SP(%)	#SP	temps/SP (s)	SP(%)	#SP	temps/SP (s)
BrandimarteMk	99.99	31	45.7	99.91	200	6.3
HurinkEdata	100	2	1497.8	100	54	41.1
HurinkRdata	100	8	432.2	82.99	6031	0.4
HurinkVdata	99.72	2	1075.3	51.89	4986	0.3
DPpaulli	100	1	3600	100	1	3600
ChambersBarnes	100	1	3600	100	4	840
Kacem	16.67	46	0.1	0	16	0
Fattahi	100	14	109.2	0.1	2654	0.1

TABLEAU 5.1 – Performances des méthodes de décomposition pour chaque jeu d'instances

pour résoudre le sous-problème de job-shop préemptif. D'autre part, l'efficacité de la méthode $LBBD_{B\&B}$ se traduit par un plus grand nombre d'itérations avant d'atteindre la limite de temps fixée. Ce qui, nous le verrons par la suite, permet à cette méthode d'obtenir de bons résultats en termes de résolution à l'optimum et de qualité des solutions obtenues.

Nous remarquons également que pour tous les jeux d'instances, à l'exception des plus faciles (*Kacem* et *Fattahi*), la quasi-totalité du temps de calcul est consacrée à la résolution des sous-problèmes. Enfin, nous observons que pour les instances les plus compliquées (*DPpaulli* et *ChambersBarnes*), très peu de sous-problèmes sont considérés (un seul dans la plupart des cas), ce qui signifie que, quelle que soit la méthode utilisée, la limite de temps imposée d'une heure n'est pas suffisante pour résoudre ces sous-problèmes de manière optimale.

5.4.2 Comparaison des méthodes

Nous étudions également la résolution du problème de job-shop flexible préemptif à l'aide du modèle de PPC présenté dans le paragraphe 5.2.2. Nous désignons cette méthode par PPC. Afin de garantir une comparaison équitable, nous proposons également une variante de cette méthode, notée $warm\_PPC$. Cette variante utilise une solution obtenue à l'aide de l'**Heuristique 1** décrite dans le paragraphe 3.5, comme point de départ de la recherche.

Les figures 5.2, 5.3, 5.4 et 5.5 offrent une visualisation graphique des indicateurs de performance.

La figure 5.2 présente les écarts d'optimalité moyens pour chaque méthode. Nous observons que les méthodes de décomposition proposées sont plus performantes que les modèles de PPC pour chaque jeu d'instances. Globalement, si nous considérons l'ensemble des 276 instances, la méthode PPC atteint en moyenne un écart d'optimalité de 23% (22% pour $warm\_PPC$) en comparaison avec un écart de 10% pour la méthode $LBBD_{PPC}$ et 6% pour la méthode $LBBD_{B\&B}$.

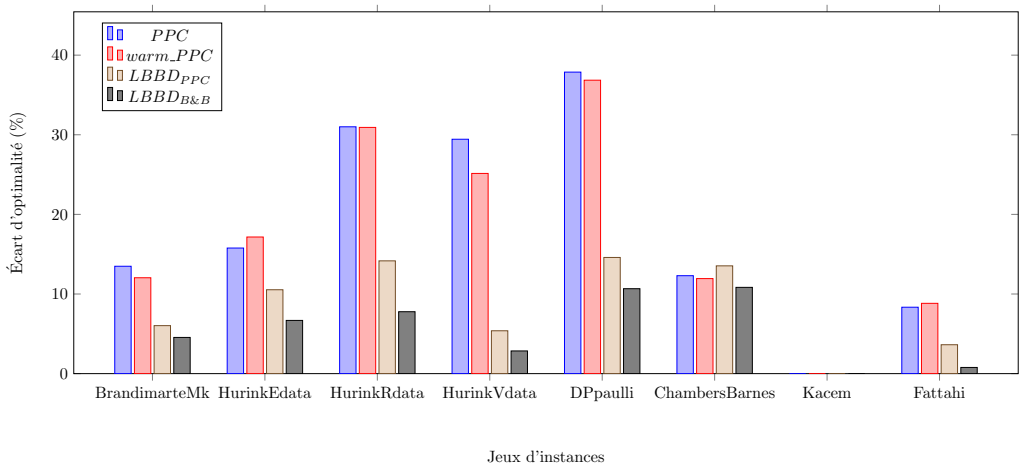


FIGURE 5.2 – Écarts d'optimalité moyens en fonction du jeu d'instances

La figure 5.3 illustre la proportion de solutions optimales trouvées pour chaque jeu d'instances. Dans l'ensemble, $LBBD_{B\&B}$ est capable de résoudre à l'optimalité un plus grand nombre d'instances dans le temps alloué, à savoir 103 contre 52 pour $LBBD_{PPC}$, 36 pour PPC et 41 pour $warm\_PPC$. Il convient de souligner qu'aucune des méthodes étudiées ne parvient à résoudre ne serait-ce qu'une instance pour les jeux d'instances *DPpaulli* et *ChambersBarnes*.

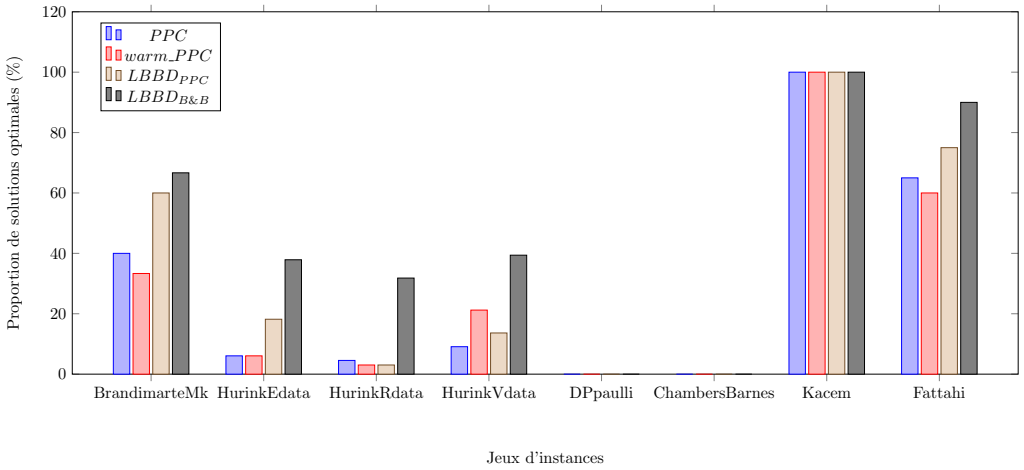


FIGURE 5.3 – Proportion de solutions optimales pour chaque jeu d'instances

La figure 5.4 décrit la proportion de meilleures solutions trouvées parmi les méthodes étudiées. Contrairement à l'écart d'optimalité, la méthode $LBBD_{PPC}$ (avec 60 meilleures solutions trouvées sur toutes les instances) ne domine pas clairement PPC (avec 53 meilleures solutions) et obtient même de moins bons résultats que $warm\_PPC$ (avec 95 meilleures solutions), pour cet indicateur. Cependant, la mé-

thode $LBBD_{B\&B}$ demeure plus performante que ces méthodes (avec 197 meilleures solutions) et trouve la meilleure solution pour au moins la moitié des instances de chaque jeu d'instances. Si nous nous concentrons sur les ensembles de données *HurinkEdata*, *HurinkRdata* et *HurinkVdata*, nous remarquons que la proportion des meilleures solutions obtenues par la méthode $LBBD_{B\&B}$ diminue à mesure que la flexibilité des instances augmente, à l'avantage de la méthode $warm\_PPC$.

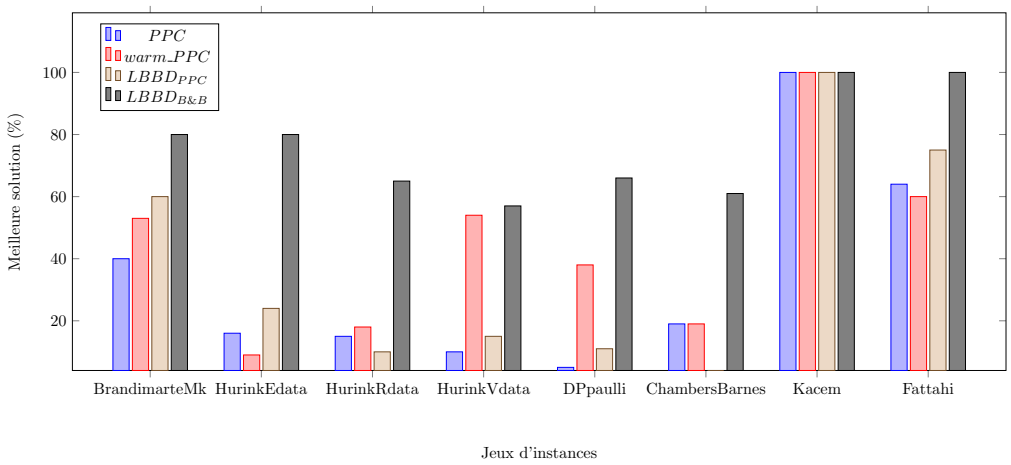


FIGURE 5.4 – Proportion de meilleures solutions trouvées pour chaque jeu d'instances

L'écart relatif à la meilleure solution est illustré par la figure 5.5. Elle montre que $LBBD_{B\&B}$ obtient, en moyenne et pour chaque jeu d'instances, des solutions de bonne qualité. En effet, l'écart relatif moyen des solutions de cette méthode n'excède pas 2% pour aucun des jeux d'instances. Pour les autres méthodes, la qualité des solutions obtenues est très variable en fonction du jeu d'instances étudié.

Sur la base de ces résultats, on en déduit que la décomposition proposée dans le chapitre 3 est bien adaptée au problème de job-shop flexible préemptif. En particulier, la méthode $LBBD_{B\&B}$ domine le modèle de PPC présenté. Pour l'ensemble des jeux d'instances et pour l'ensemble des indicateurs étudiés, elle obtient de meilleurs résultats.

Bien que la méthode $LBBD_{B\&B}$ atteigne de meilleures performances, nous remarquons une grande variabilité des résultats en fonction des jeux d'instances. Nous proposons donc de nous concentrer sur les caractéristiques des instances pouvant influencer les performances de différentes méthodes.

En particulier, la figure 5.6 nous permet d'étudier l'écart d'optimalité en fonction du nombre moyen d'opérations par machine. Chaque type de point (forme et couleur) représente une méthode de résolution et chaque point de ce graphique représente une instance résolue par l'une de ces méthodes, avec en abscisse le nombre moyen d'opérations par machine de l'instance, et en ordonnée l'écart d'optimalité obtenu .

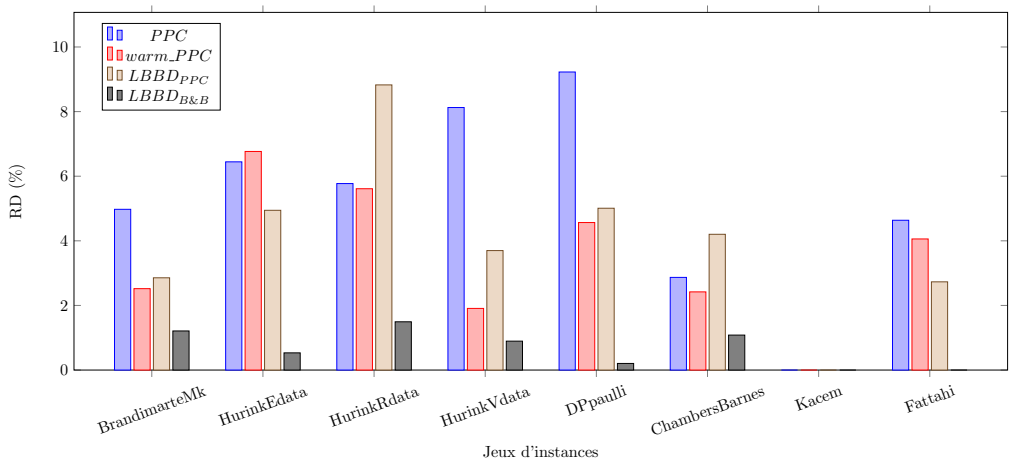


FIGURE 5.5 – Écart relatif à la meilleure solution moyen pour chaque jeu d'instances

L'analyse de la figure 5.6 confirme ce que l'on peut voir dans la figure 5.2, à savoir que les méthodes basées sur la décomposition permettent d'obtenir un meilleur écart d'optimalité que les méthodes de PPC pures. De plus, bien que les données soient assez dispersées en raison de la grande hétérogénéité des instances, l'observation de la régression linéaire effectuée pour chaque méthode est très intéressante. En effet, on constate que les problèmes deviennent plus difficiles à résoudre lorsque le nombre d'opérations par machine augmente. Cependant, l'efficacité des méthodes de décomposition est moins sensible à ce facteur que les méthodes CP.

Toutefois, pour certaines instances (75 sur 276), les méthodes basées sur la PPC ($warm\_PPC$ et PPC) obtiennent de meilleures solutions réalisables que les méthodes basées sur la méthode de décomposition ($LBBD_{PPC}$ et $LBBD_{B\&B}$). Afin d'identifier les caractéristiques des instances pour lesquelles il est le plus approprié d'appliquer une méthode de décomposition, nous utilisons une méthode de classification. Celle-ci est basée sur l'apprentissage de règles de décision, déduites à partir des caractéristiques des données, pour construire un arbre de décision. Pour cela, nous utilisons une bibliothèque gratuite d'apprentissage automatique appelée "*Scikit-Learn*". Nous séparons les instances en trois classes :

Classe 1 : les instances pour lesquelles un meilleur résultat est obtenu avec la méthode de décomposition,

Classe 2 : celles pour lesquelles un meilleur résultat est obtenue avec une méthode de PPC,

Classe 3 : celles pour lesquelles le même meilleur résultat est obtenu par les deux types de méthodes.

Pour chaque instance, nous considérons les paramètres tels que sa taille (nombre de travaux, de machines et d'opérations), sa flexibilité (nombre de machines éligibles par opération) et le temps de traitement moyen de ses opérations. Nous limitons la profondeur maximale de l'arbre à 3 étages. L'arbre résultant est illustré dans

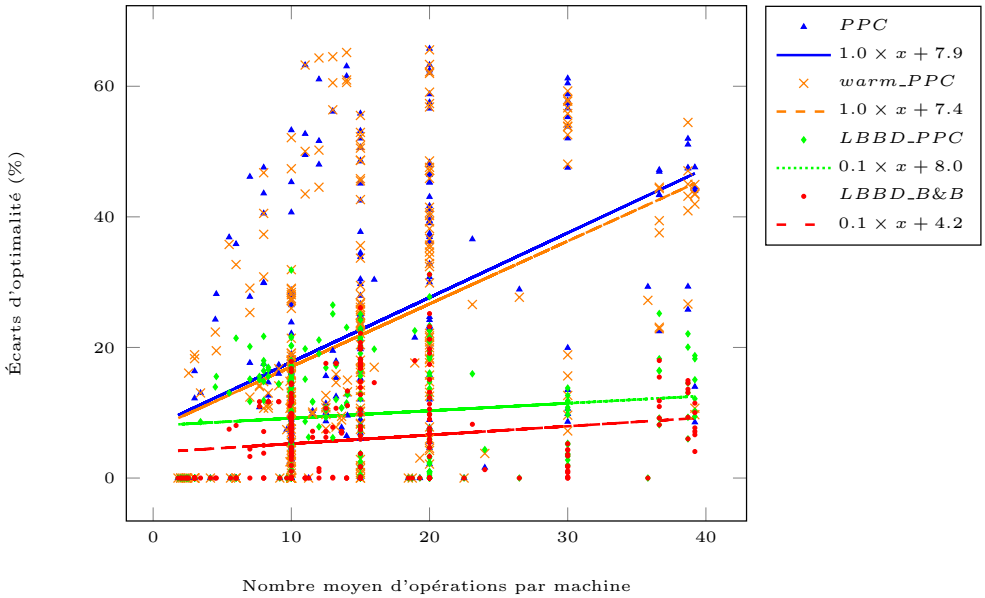


FIGURE 5.6 – Écart d'optimalité en fonction du nombre moyen d'opérations par machine

la figure 5.7. Les nœuds de décision sont représentés par des rectangles, les nœuds feuilles par des ellipses vertes dans lesquelles nous pouvons trouver le nombre d'instances appartenant à chaque classe respectant les conditions fixées par l'ensemble des branches de décision parcourues sur le chemin depuis la racine.

L'arbre de décision représenté par la figure 5.7 nous montre que la caractéristique la plus importante pour discriminer ces classes sont la flexibilité des opérations et le nombre de machines. En effet, nous remarquons que pour toutes les instances ayant une flexibilité moyenne supérieure à 4.403, les méthodes basées sur la PPC obtiennent systématiquement la meilleure solution (c'est-à-dire une BS strictement plus petite, pour 33 d'entre elles, ou la même meilleure solution que les méthodes de décomposition, pour 12 autres). Pour les autres instances, celles ayant plus de 9 machines sont pour la plupart mieux résolues par les méthodes basées sur le décomposition (88 instances ont une BS strictement plus petite avec une méthode basée sur la décomposition, 3 avec une méthode basée sur la PPC et la même meilleure solution avec les deux types de méthodes pour 24 d'entre elles). Enfin, d'après nos expérimentations, les instances restantes ne peuvent pas être séparées efficacement, même en augmentant la profondeur de l'arbre.

5.5 Conclusion

Dans ce chapitre, nous présentons plusieurs méthodes exactes pour résoudre le pFJSSP avec minimisation du makespan.

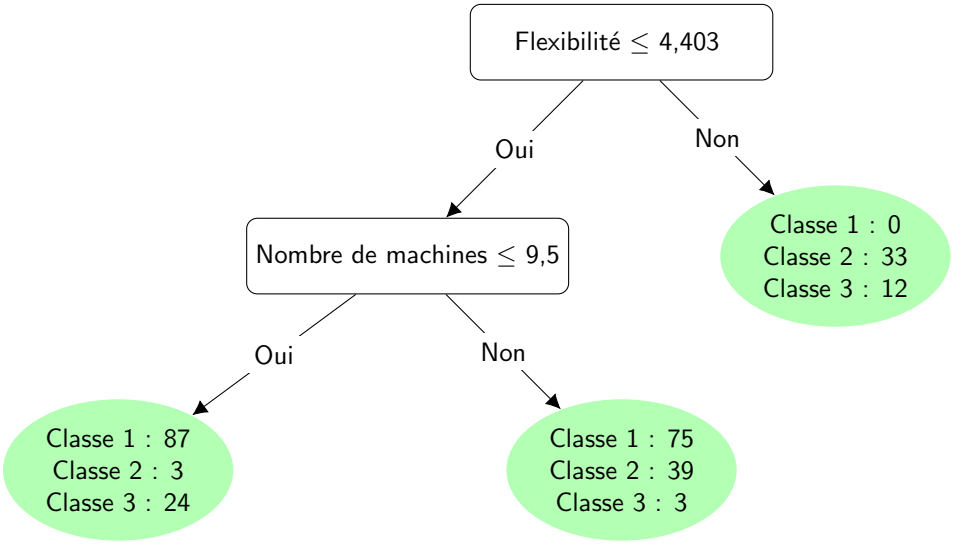


FIGURE 5.7 – Arbre de décision de profondeur 3 classant les instances en fonction du type de méthode permettant d'obtenir la meilleure solution possible en fonction des caractéristiques

Nous modélisons la problème à l'aide de formulations directes basées sur la PLNE et la PPC. Notre principale contribution est d'appliquer le modèle de décomposition de Benders basée sur la logique présenté dans le chapitre 3 à ce problème. Cette décomposition permet de résoudre le sous problème de pJSSP à l'aide d'un algorithme de *branch-and-bound* dédié, issu de la littérature. Avec cette méthode, la décomposition dépasse les formulations directes à la fois en termes d'écart d'optimalité et en termes de qualité des solutions trouvées.

Nous avons également étudié un autre critère, à savoir la minimisation de la somme des dates de fin de tous les travaux. N'ayant pas obtenu de résultats probants pour ce critère, nous n'avons pas inclus cette étude dans ce manuscrit. Cependant, elle est accessible dans le rapport [Juvin 2023d]. Nous y avons reporté, en particulier, les modifications nécessaires pour la relaxation du sous-problème dans le problème maître, et pour la résolution du sous-problème ainsi que les résultats expérimentaux obtenus.

Troisième partie

Méthodes hybrides pour l'ordonnancement robuste

Chapitre 6

Le problème de job-shop robuste

Sommaire

6.1 Introduction

Dans ce chapitre, nous présentons nos contributions dans le cadre de la version robuste du problème de job-shop. En particulier, nous supposons que les durées de traitement des opérations sont soumises à une incertitude, relative à une méconnaissance ou une variabilité des durées de traitement ou à des pannes éventuelles de machine. Plus précisément, l'ensemble d'incertitude est défini à l'aide d'un budget d'incertitude introduit par [Bertsimas 2004]. Nous considérons alors un problème à deux étapes. La première étape consiste à déterminer une séquence d'opérations sur chaque machine. Cette étape est réalisée avant la révélation des aléas sur les durées

de traitement. La seconde étape correspond au choix des dates de traitement des opérations. On considère que cette étape peut être réalisée une fois que l'incertitude est révélée, ce qui permet d'adapter la solution de second niveau à la réalisation du scénario. L'objectif est donc de trouver une séquence de d'opérations et un makespan minimal pour lesquels il existe des dates de début pour chaque scénario dans l'ensemble d'incertitude considéré, respectant les contraintes de précédence et de ressource. Une partie de ces travaux ont fait l'objet de la publication [Juvin 2023e].

6.2 Définition du problème

On note Σ l'ensemble des solutions de premier niveau réalisables et $\sigma = (\sigma_1, \ldots, \sigma_{|\mathcal{M}|})$ un élément de cet ensemble avec σ_m une séquence d'opérations sur la machine $m \in \mathcal{M}$. On note $O_{i,j} \prec_\sigma O_{i',j'}$ une relation de précédence induite par σ, qui signifie que l'opération $O_{i,j}$ termine avant le début de l'opération $O_{i',j'}$ dans la séquence σ.

Soit $\tau = (t^\sigma_{i,j}(\xi) | i \in \mathcal{J}, j \in \{1, \ldots, n_i\})$ un vecteur tel que $t^\sigma_{i,j}(\xi) \in \mathbb{R}$ est un réel désignant la date de début de l'opération $O_{i,j}$, sous la séquence σ, dans le scénario ξ, alors le problème peut être défini comme suit :

$$\min_{\sigma \in \Sigma, C_{\max} \in \mathbb{R}} \max_{\xi \in \mathcal{U}} \min_{\tau \in \mathbb{R}^n} C_{\max} \tag{6.1}$$

t.q.

$$C_{\max} \geq t^\sigma_{i,n_i}(\xi) + p_{i,n_i}(\xi) \quad \forall i \in \mathcal{J} \tag{6.2}$$

$$t^\sigma_{i,j+1}(\xi) \geq t^\sigma_{i,j}(\xi) + p_{i,j}(\xi) \quad \forall i \in \mathcal{J}, \forall j \in \{1, \ldots, n_i - 1\} \tag{6.3}$$

$$t^\sigma_{i',j'}(\xi) \geq t^\sigma_{i,j}(\xi) + p_{i,j}(\xi) \quad \forall (O_{i,j}, O_{i',j'}) \in \sigma^2_m, \ t.q.\ O_{i,j} \prec_\sigma O_{i',j'}, \forall m \in \mathcal{M} \tag{6.4}$$

L'objectif (6.1) est de trouver une séquence d'opération σ minimisant le makespan dans le scénario pire cas, en considérant que les dates de traitement peuvent s'adapter au scénario. Les contraintes (6.2) définissent le makespan. Les contraintes (6.3) sont les contraintes de précédence entre deux opérations successives d'un même travail. Les contraintes (6.4) sont les contraintes disjonctives sur les machines.

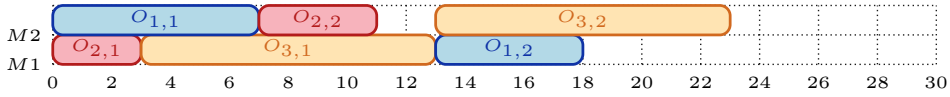
Exemple 11 *Considérons une instance de job-shop robuste avec 3 travaux et 2 machines. Les intervalles $[\bar{p}_{i,j}, \bar{p}_{i,j} + \hat{p}_{i,j}]$ des temps de traitement $p_{i,j}$ des opérations $O_{i,j}$, $\forall i \in \mathcal{J}$, $\forall j \in \{1, \ldots, n_i\}$, sont donnés dans le tableau 6.1.*

Une solution optimale est obtenue avec la séquence σ_1, qui consiste à ordonnancer, dans cet ordre, les opérations $O_{2,1}$, puis $O_{3,1}$ et $O_{1,2}$ sur la machine M_1, et les opérations $O_{1,1}$, $O_{2,2}$ et $O_{3,2}$ sur la machine M_2 (figure 6.1). La figure 6.1a illustre cette solution lorsque tous les temps de traitement prennent leur valeur nominale. Le makespan est alors de 23.

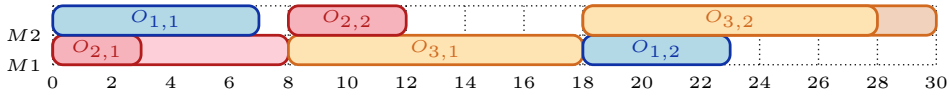
En considérant un budget d'incertitude $\Gamma = 2$, le pire cas pour cette solution est rencontré lorsque les temps de traitement des opérations $O_{2,1}$ et $O_{3,2}$ dévient de leur durée nominale et prennent leur plus grande valeur. La figure 6.1b montre le

		$M1$	$M2$
$J1$	$O_{1,1}$		[7,12]
	$O_{1,2}$	[5,9]	
$J2$	$O_{2,1}$	[3,8]	
	$O_{2,2}$		[4,6]
$J3$	$O_{3,1}$	[10,11]	
	$O_{3,2}$		[10,12]

TABLEAU 6.1 – Exemple numérique d'une instance de job-shop avec durées de traitement incertaines



(a) Temps de traitement nominaux, makespan = 23



(b) Scénario pire cas pour $\Gamma = 2$, makespan = 30

FIGURE 6.1 – Diagrammes de Gantt pour l'exemple 11 et la séquence σ_1

diagramme de Gantt dans ce cas. La valeur de la fonction objectif de cette solution pour un budget d'incertitude $\Gamma = 2$ *atteint* 30.

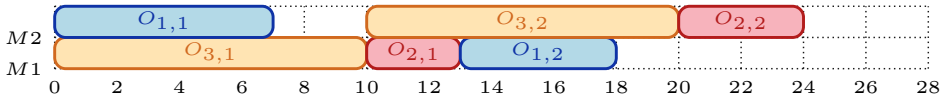
Une autre solution possible consiste à considérer la séquence σ_2 *qui ordonne les opérations* $O_{3,1}$, $O_{2,1}$ *puis* $O_{1,2}$ *sur la machine* M_1 *et les opérations* $O_{1,1}$, $O_{3,2}$ *et* $O_{2,2}$ *sur la machine* M_2 *(figure 6.2). La figure 6.2a représente cette solution lorsque tous les temps de traitement prennent leur valeur nominale. Le makespan est de* 24.

En considérant un budget d'incertitude $\Gamma = 2$, *le pire cas pour cette solution est rencontré lorsque les temps de traitement des opérations* $O_{1,1}$ *et* $O_{3,2}$ *dévient. La figure 6.2b illustre ce cas. Le makespan est maintenant égal à* 28, *cette solution a donc un meilleur makespan dans le pire des cas pour un budget d'incertitude* $\Gamma = 2$.

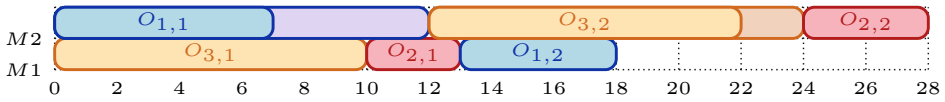
Cet exemple simple montre l'utilité de mettre en oeuvre une solution robuste lorsque les paramètres du problème ne sont pas totalement identifiés et de ne pas se contenter de la meilleure (ou d'une bonne) solution dans le cas déterministe.

6.3 Évaluation d'un scénario pire cas

Dans ce paragraphe, on suppose qu'une solution de premier niveau σ est fixée. Étant donné un budget d'incertitude Γ, l'objectif de l'évaluation du scénario pire



(a) Temps de traitement nominaux, makespan = 24



(b) Scénario pire cas pour $\Gamma = 2$, makespan = 28

FIGURE 6.2 – Diagrammes de Gantt pour l'exemple 11 et la séquence σ_2

cas est d'identifier le scénario, avec au plus Γ opérations dont la durée dévie, qui conduit au plus grand makespan possible :

$$\max_{\xi \in \mathcal{U}} \min_{t_{i,j}} C_{\max} \tag{6.5}$$

Pour une séquence σ et un scénario ξ fixés, il est évident qu'il est optimal de débuter une opération dès que possible (c'est-à-dire à la plus grande date de fin de ses prédécesseurs si elle en a, à 0 sinon). L'objectif est donc de trouver le scénario qui maximise le makespan, tout en contraignant les opérations à commencer au plus tôt.

Nous proposons deux formulations mathématiques de ce problème basées respectivement sur un modèle PLNE et un modèle PPC. Ensuite, nous montrons que ce problème se rapporte à un problème de plus long chemin dans un graphe.

6.3.1 Formulations mathématiques

Un scénario est totalement défini par l'ensemble des opérations dont la durée de traitement dévie, les variables de décision peuvent donc être définies comme :

$$\xi_{i,j} = \begin{cases} 1 & \text{si la durée de l'opération } O_{i,j} \text{ dévie} \\ 0 & \text{sinon.} \end{cases} \qquad \forall i \in \mathcal{J}, \forall j \in \{1, \ldots, n_i\}$$

et $t_{i,j}$, la date de début de l'opération $O_{i,j}, \forall i \in \mathcal{J}, \forall j \in \{1, \ldots, n_i\}$.

Le problème d'évaluation pire cas peut alors être décrit par le modèle mathématique suivant (où $\mathcal{A}_{i,j}$ représente l'ensemble des prédécesseurs immédiats de l'opération $O_{i,j}$) :

$$\max \; C_{\max} \tag{6.6}$$

t.q.

$$\sum_{i\in\mathcal{J}}\sum_{j\in\{1,\dots,n_i\}}\xi_{i,j}\leq\Gamma \tag{6.7}$$

$$t_{i,1}=0 \qquad \forall i\in\mathcal{J}\mid\mathcal{A}_{i,0}=\emptyset \tag{6.8}$$

$$t_{i,j}=\max_{O_{i',j'}\in\mathcal{A}_{i,j}}(t_{i',j'}+\bar{p}_{i',j'}+\xi_{i,'j'}\hat{p}_{i',j'}) \qquad \forall i\in\mathcal{J},\forall j\in\{1,\dots,n_i\} \tag{6.9}$$

$$C_{\max}=\max_{i\in\mathcal{J}}(t_{i,n_i}+\bar{p}_{i,n_i}+\xi_{i,n_i}\hat{p}_{i,n_i}) \tag{6.10}$$

$$\xi_{i,j}\in\{0,1\} \qquad \forall i\in\mathcal{J},\forall j\in\{1,\dots,n_i\} \tag{6.11}$$

$$t_{i,j}\in\mathbb{R} \qquad \forall i\in\mathcal{J},\forall j\in\{1,\dots,n_i\} \tag{6.12}$$

L'objectif est la maximisation du makespan (6.6). La contrainte (6.7) garantit qu'au plus Γ opérations ont leur durée qui dévie. Les contraintes (6.8) et (6.9) contraignent les opérations à commencer à la plus grande date de fin de leurs prédécesseurs ; à 0 pour les opérations sans prédécesseur (contraintes (6.8)). Les contraintes (6.10) définissent le makespan (plus grande date de fin des travaux). Enfin les contraintes (6.11) et (6.12) définissent le domaine de définitions des variables du problème.

6.3.1.1 Programmation linéaire en nombres entiers

Dans ce paragraphe, nous proposons de linéariser le modèle précédent en supprimant les fonctions max des contraintes. Pour cela, nous introduisons les variables binaires suivantes :

$$y_{i,j,i',j'}=\begin{cases}1 & \text{si l'opération } O_{i,j} \text{ débute exactement à la fin de l'opération } O_{i',j'}\\ 0 & \text{sinon}\end{cases}$$

et

$$z_i=\begin{cases}1 & \text{si le travail } i \text{ termine avec la fin de l'ordonnancement}\\ 0 & \text{sinon.}\end{cases}$$

Les contraintes (6.9) sont remplacées par les contraintes :

$$t_{i,j}-(t_{i',j'}+\bar{p}_{i',j'}+\xi_{i,'j'}\hat{p}_{i',j'})\leq H(1-y_{i,j,i',j'}) \qquad \forall i\in\mathcal{J},\forall j\in\{1,\dots,n_i\},\forall O_{i',j'}\in\mathcal{A}_{i,j} \tag{6.13}$$

et

$$\sum_{(i',j')\in\mathcal{A}_{i,j}}y_{i,j,i',j'}\geq 1 \qquad \forall i\in\mathcal{J},\forall j\in\{1,\dots,n_i\}\mid\mathcal{A}_{i,j}\neq\emptyset \tag{6.14}$$

Pour deux opérations fixées $O_{i,j}$ et $O_{i',j'}$ telles que $O_{i',j'}\in\mathcal{A}_{i,j}$:

— si $y_{i,j,i',j'}=0$, alors le membre de droite de l'inéquation (6.13) est égal à H et la contrainte est inactive.

— si $y_{i,j,i',j'}=1$, alors la contrainte (6.13) force l'opération $O_{i,j}$ à débuter au plus tard à la fin de l'opération $O_{i',j'}$.

Les contraintes (6.14) assurent que chaque opération $O_{i,j}$ (avec au moins un prédécesseur $\mathcal{A}_{i,j} \neq \emptyset$) débute quand l'un de ses prédécesseurs termine.

Sur le même principe, les contraintes (6.10) sont remplacées par les contraintes suivantes :

$$C_{\max} - (t_{i,n_i} + \bar{p}_{i,n_i} + \xi_{i,n_i}\hat{p}_{i,n_i}) \leq H(1 - z_i) \qquad \forall i \in \mathcal{J} \tag{6.15}$$

et

$$\sum_{i \in \mathcal{J}} z_i \geq 1 \tag{6.16}$$

6.3.1.2 Programmation par contraintes

Le modèle de PPC que nous proposons ici est défini à l'aide de variables d'intervalle optionnelles, présentes dans la solution seulement si la durée de l'opération à laquelle elles sont associées dévie dans le scénario pire cas. Les variables de décision du problème sont les suivantes :

— $task_{i,j}$: variable d'intervalle associée à l'opération $O_{i,j}$ de durée $\bar{p}_{i,j,m}$;
— $dev_{i,j}$: variable d'intervalle optionnelle de durée $\hat{p}_{i,j,m}$, présente si l'opération $O_{i,j}$ dévie.

Ainsi, dans une solution, la période de traitement d'une opération $O_{i,j}$ sera représentée par l'union des intervalles $task_{i,j}$ et $dev_{i,j}$.

$$\max C_{\max} \tag{6.17}$$

t.q.

$$task_{i,1}.debut = 0 \qquad \forall i \in \mathcal{J} \mid \mathcal{A}_{i,1} = \emptyset \tag{6.18}$$

$$task_{i,j}.debut = \max_{O_{i',j'} \in \mathcal{A}_{i,j}} (\{task_{i',j'}.fin\} \cup \{dev_{i',j'}.fin\}) \qquad \forall i \in \mathcal{J}, \forall j \in \{1, \ldots, n_i\} \tag{6.19}$$

$$StartAtEnd(dev_{i,j}, task_{i,j}) \qquad \forall i \in \mathcal{J}, \forall j \in \{1, \ldots, n_i\} \tag{6.20}$$

$$\sum_{i \in \mathcal{J}} \sum_{j \in \{1,\ldots,n_i} \} PresenceOf(dev_{i,j}) = \Gamma \tag{6.21}$$

$$C_{\max} = \max_{i \in \mathcal{J}} (\{task_{i,n_i}.fin\} \cup \{dev_{i,n_i}.fin\}) \tag{6.22}$$

L'objectif est la maximisation du makespan (6.17). La contrainte (6.21) garantit que la durée d'au plus Γ opérations dévie de sa valeur nominale. Les contraintes (6.18) et (6.19) contraignent les opérations à débuter au plus tôt, compte tenu des séquences sur les machines et des contraintes de précédence. Les contraintes (6.20) garantissent que, si une variable d'intervalle $dev_{i,j}$ est présente, alors elle débute à la fin de l'intervalle $task_{i,j}$. Ainsi, le traitement de l'opération $O_{i,j}$ s'étend sur une

durée égale à sa durée maximale et sans interruption. Enfin, les contraintes (6.22) expriment le makespan comme la plus grande date de fin des travaux.

6.3.2 Problème de plus long chemin dans un graphe

Le problème d'évaluation pire cas peut être vu comme l'évaluation du plus long chemin dans un graphe. Une solution équivalente à celle proposée ici est présentée dans [Bruni 2017] dans le cadre du problème d'ordonnancement de projet sous contraintes de ressources robuste.

Une solution de premier niveau peut être représentée par un graphe $G = (\mathcal{N}, \mathcal{A})$ dans lequel les nœuds $\mathcal{N}$ représentent les opérations du problème ainsi que deux tâches fictives *debut* et fin. Les arcs $\mathcal{A}$ représentent les relations de précédence (dans le cadre du problème de job-shop, il s'agit à la fois des précédences sur les travaux définies par le problème et celles induites par les séquences fixées par la solution de premier niveau).

Exemple 12 *Considérons à nouveau l'instance présentée dans l'exemple 11 avec le même ordonnancement que celui représenté sur la figure 6.2. La figure 6.3 illustre le graphe conjonctif G représentant cette solution. Les opérations $O_{3,1}$, puis $O_{2,1}$ et $O_{1,2}$ sont exécutes sur la machine M_1 (en bleu), et les opérations $O_{1,1}$, $O_{3,2}$ et $O_{2,2}$ sur la machine M_2 (en rouge), dans cet ordre. Les arcs noirs représentent des contraintes de précédence.*

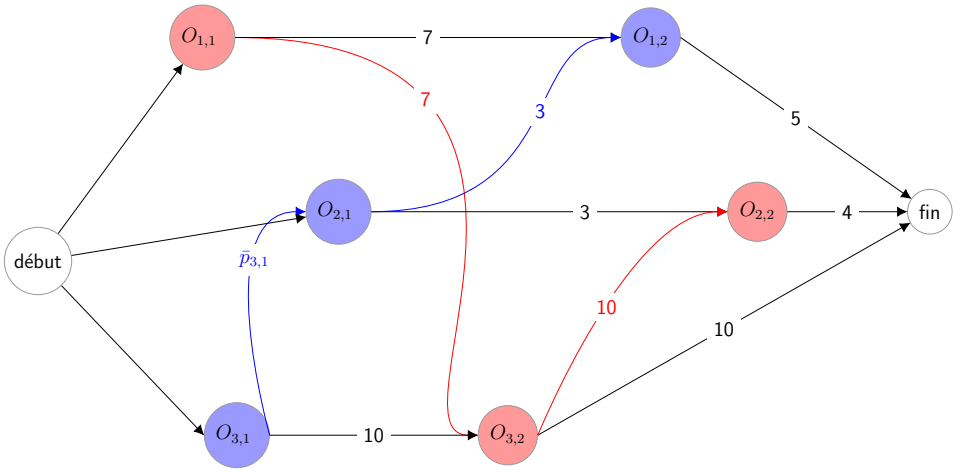


FIGURE 6.3 – Exemple 11 : graphe conjonctif G

À partir d'un graphe G et étant donné un budget d'incertitude Γ, il est possible de créer le graphe $G' = (\mathcal{N}', \mathcal{A}')$. Dans ce but, pour chaque nœud u dans $\mathcal{N}$, on crée $\Gamma + 1$ nœuds $\{u^\gamma\}$ dans $\mathcal{N}'$ avec $\gamma \in \{0, \ldots, \Gamma\}$. Le nœud u^γ représente l' opération u après au plus γ déviations. Pour chaque arc $(u, v) \in \mathcal{A}$ du nœud u au nœud v, on crée $2 \times \Gamma + 1$ arcs, à savoir : $\Gamma + 1$ arcs, de u^γ vers v^γ de valuation $cout_{u^\gamma, v^\gamma} = \bar{p}_u$,

correspondant à la durée nominale de l'opération notée u, $\forall \gamma \in \{0, \ldots, \Gamma\}$, et Γ arcs, du nœud u^{γ} au nœud $v^{\gamma+1}$ de valuation $\bar{p}_u + \hat{p}_u$, correspondant à la durée nominale de l'opération notée u, $\forall \gamma \in \{0, \ldots, \Gamma - 1\}$. Ainsi, tout chemin du nœud source *debut* à un nœud quelconque $u^{\gamma} \in \mathcal{N}' \setminus \{debut, fin\}$ passera par au plus γ arcs de valuation déviée et trouver un plus long chemin dans le graphe G' est équivalent à déterminer le scénario pire cas.

Exemple 13 *À partir du graphe G représenté par la figure 6.3 et en considérant un budget d'incertitude $\Gamma = 2$ et les données de l'exemple 11, nous construisons le graphe G' de la figure 6.4.*

Notons que si dans G l'ensemble des chemins du nœud debut au nœud représentant une opération $O_{i,j}$ passent par au plus λ nœuds, seules λ déviations peuvent se produire avant elle. Ainsi, les nœuds des opérations sans prédécesseur, $O_{1,1}$ et $O_{3,1}$, ne sont pas dupliqués, et les nœuds des opérations avec au plus un seul prédécesseur, par chemin, depuis le nœud debut, $O_{2,1}$ et $O_{3,2}$, ne sont dupliqués qu'en deux exemplaires (aucune déviation ou une seule déviation).

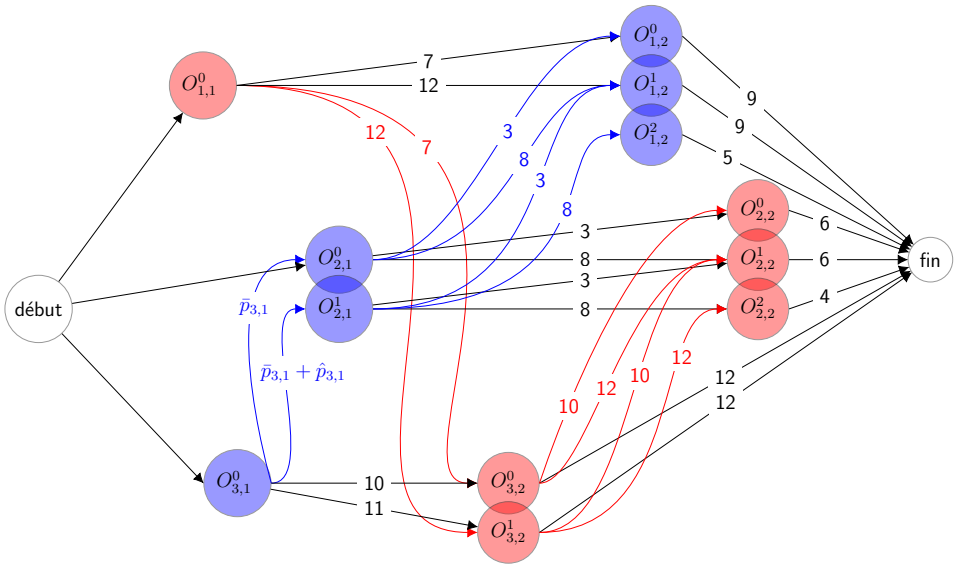


FIGURE 6.4 – Exemple 11 : graphe augmenté G' avec un niveau par nombre de déviations possibles, pour $\Gamma = 2$

L'un des plus longs chemins de ce graphe G' (d'autres chemins de même longueur existent), du nœud "debut" au nœud "fin", est le chemin passant par les nœuds $O^0_{1,1}$, $O^1_{3,2}$ et $O^2_{2,2}$, d'une longueur de 28. De ce chemin, on peut déduire un scénario parmi les plus défavorables, pour cet ordonnancement et à ce budget d'incertitude $\Gamma = 2$. En effet, ce chemin est composée de deux arcs de longueur déviée, ce qui signifie que les opérations associées à ces arcs ($O_{1,1}$ et $O_{3,2}$) sont celles dont le temps de traitement dévie dans un des scénarios pire cas. La représentation de cette solution sous forme de diagramme de Gantt correspond exactement à la figure 6.2b.

Le graphe G' ainsi construit possède $|\mathcal{N}'| \leq (\Gamma+1)n+2 \leq (n+1)n+2$ nœuds et $|\mathcal{A}'| \leq 4|\mathcal{N}'|$ arcs.

6.3.2.1 Algorithme de programmation dynamique

Il est ainsi possible de déterminer le scénario le plus défavorable, pour tout budget d'incertitude $\Gamma \leq n$ et pour un ordonnancement donné en calculant le chemin le plus long dans un graphe orienté et acyclique avec au plus n^2+n+2 nœuds. Or, il existe un algorithme de programmation dynamique de complexité linéaire en nombre de nœuds, pour trouver un plus long chemin dans un graphe orienté acyclique [Katriel 2005]. Le problème d'évaluation d'un scénario pire cas peut donc être résolu à l'optimum avec une complexité de $\mathcal{O}(n^2)$.

6.3.2.2 Programmation linéaire en nombres entiers

Il est aussi possible de reformuler le problème de plus long chemin dans un graphe à l'aide d'un modèle de programmation linéaire en nombres entiers. Pour cela, nous introduisons les variables binaires suivantes :

$$y_{u,v} = \begin{cases} 1 & \text{si le chemin sélectionné passe par l'arc } (u,v)\,; \\ 0 & \text{sinon.} \end{cases}$$

Le modèle est le suivant :

$$\max \sum_{(u,v)\in\mathcal{A}'} y_{u,v} coût_{u,v} \tag{6.23}$$

t.q.

$$\sum_{u\in\mathcal{N}'\setminus\{debut,fin\}} y_{debut,u} = 1 \tag{6.24}$$

$$\sum_{u\in\mathcal{N}'\setminus\{debut,fin\}} y_{u,fin} = 1 \tag{6.25}$$

$$\sum_{(u,v)\in\mathcal{A}'} y_{u,v} - \sum_{(v,u)\in\mathcal{A}'} y_{v,u} = 0 \qquad \forall u \in \mathcal{N}' \setminus \{debut, fin\} \tag{6.26}$$

$$y_{u,v} \in \{0,1\} \qquad \forall (u,v) \in \mathcal{A}' \tag{6.27}$$

L'objectif (6.23) est la maximisation de la longueur du chemin. La contrainte (6.24) garantit que le nœud source *debut* a exactement un successeur sur le chemin. La contrainte (6.25) oblige le nœud de destination fin à être le successeur d'exactement un nœud. Les contraintes (6.26) assurent la conservation du flux sur le chemin. Enfin, les contraintes (6.27) définissent le domaine de définition des variables.

La relaxation linéaire de (6.23- 6.27) est entière, et l'optimum peut aussi être calculé à travers la formulation duale. Ainsi, le problème du plus long chemin dans

le graphe G' peut être obtenu à l'aide du modèle suivant :

$$\min C_{fin} + C_{debut} \tag{6.28}$$

$$C_u - C_v \geq coût_{u,v} \qquad \forall (u,v) \in \mathcal{A}' \tag{6.29}$$

$$C_u \geq 0 \qquad \forall u \in \mathcal{N}' \tag{6.30}$$

où la variable duale C_u représentent la longueur du chemin le plus long entre le nœud *debut* et le nœud $u \in \mathcal{N}'$.
L'interprétation du modèle (6.28-6.30) est la suivante. L'objectif (6.28) est la minimisation de la longueur du plus long chemin entre les nœuds *debut* et fin. Les contraintes (6.29) garantissent que la longueur du plus long chemin C_v vers un nœud donné v est au moins aussi grande que celle de chacun de ses prédécesseurs ($u \in \mathcal{N}' | (u,v) \in \mathcal{A}'$) augmenté de la longueur $coût_{u,v}$ de l'arc (u,v). Enfin, les contraintes (6.30) définissent le domaine des variables.

Soit $O_{i,j}$ l'opération notée u après γ déviations, $C_u = C^{\gamma}_{i,j}$. Les contraintes (6.29) peuvent être remplacées par :

$$C^{\gamma}_{i,j+1} \geq C^{\gamma}_{i,j} + \bar{p}_{i,j+1} \qquad \forall i \in \mathcal{J}, \forall j \in \{1,\dots,n_i - 1\}, \forall \gamma \in \{0,\dots \Gamma\} \tag{6.31}$$

$$C^{\gamma+1}_{i,j+1} \geq C^{\gamma}_{i,j} + \bar{p}_{i,j+1} + \hat{p}_{i,j+1} \qquad \forall i \in \mathcal{J}, \forall j \in \{1,\dots,n_i - 1\}, \forall \gamma \in \{0,\dots \Gamma - 1\} \tag{6.32}$$

$$C^{\gamma}_{i',j'} \geq C^{\gamma}_{i,j} + \bar{p}_{i',j'} \qquad \forall (O_{i,j}, O_{i',j'}) \in \sigma^2_m, \text{ t.q. } O_{i,j} \prec_\sigma O_{i',j'}, \forall m \in \mathcal{M}, \forall \gamma \in \{0,\dots \Gamma\} \tag{6.33}$$

$$C^{\gamma+1}_{i',j'} \geq C^{\gamma}_{i,j} + \bar{p}_{i',j'} + \hat{p}_{i',j'} \qquad \forall (O_{i,j}, O_{i',j'}) \in \sigma^2_m, \text{ t.q. } O_{i,j} \prec_\sigma O_{i',j'}, \forall m \in \mathcal{M}, \forall \gamma \in \{0,\dots \Gamma - 1\} \tag{6.34}$$

6.4 Formulations directes

Dans ce paragraphe, nous présentons deux types d'approches directes pour résoudre le problème de job-shop robuste, par opposition aux méthodes de décomposition présentées dans le paragraphe 6.5. Le premier type d'approche repose sur une formulation étendue du problème, dans laquelle les variables et les contraintes du second niveau sont dupliquées pour chaque scénario considéré, on note alors que le nombre de scénarios augmente de manière exponentielle avec la taille du problème. Le deuxième type d'approche est basé sur une formulation compacte du problème, dont le nombre de variables et de contraintes croît de manière polynomiale par rapport à la taille du problème.

6.4.1 Modèles étendus

Nous présentons ici deux formulations étendues du problème de JSSP robuste, l'une basée sur la PLNE, l'autre sur la PPC.

Dans ces formulations, l'ensemble des scénarios considérés est noté $\mathcal{S}$. Afin de trouver une solution robuste, il convient de tenir compte de tous les scénarios de l'ensemble d'incertitude $\mathcal{S} = \mathcal{U}$. Il est cependant possible de considérer seulement un sous-ensemble plus restreint $\mathcal{S} \subset \mathcal{U}$. Dans ce cas, les formulations obtenues sont des relaxations du problème robuste et permettent d'obtenir une borne inférieure sur la valeur optimale du problème robuste.

6.4.1.1 Programmation linéaire en nombres entiers

Le modèle suivant est basée sur une formulation disjonctive. Il est inspiré du modèle présenté par Manne [Manne 1960] pour le problème de job-shop déterministe. On note $\mathcal{I}_m$ l'ensemble des opérations traitées par la machine $m \in \mathcal{M}$. Les variables de décision de premier niveau concernent la position relative de deux opérations traitées par une même machine :

$$y_{i,j,i',j'} = \begin{cases} 1 & \text{si } O_{i,j} \text{ exécutée avant } O_{i',j'}; \\ 0 & \text{sinon} \end{cases} \qquad \forall m \in \mathcal{M}, \forall (O_{i,j}, O_{i,j'}) \in \mathcal{I}_m^2$$

Les dates de début $t_{i,j}(\xi) \in \mathbb{R}_+$ de l'opération $O_{i,j}$ dans le scénario ξ sont les variables de second niveau. La formulation du modèle est la suivante :

$$\min C_{\max} \tag{6.35}$$

t.q.

$$C_{\max} \geq t_{i,n_i}(\xi) + p_{i,n_i}(\xi) \qquad \forall i \in \mathcal{J}, \forall \xi \in \mathcal{S} \tag{6.36}$$

$$t_{i,j+1}(\xi) \geq t_{i,j}(\xi) + p_{i,j}(\xi) \qquad \forall i \in \mathcal{J}, \forall j \in \{1, \ldots, n_i - 1\}, \forall \xi \in \mathcal{S} \tag{6.37}$$

$$t_{i,j}(\xi) \geq t_{i',j'}(\xi) + p_{i',j'}(\xi) - y_{i,j,i',j'} H \qquad \forall m \in \mathcal{M}, \forall (O_{i,j}, O_{i,j'}) \in \mathcal{I}_m^2, \forall \xi \in \mathcal{S} \tag{6.38}$$

$$t_{i',j'}(\xi) \geq t_{i,j}(\xi) + p_{i,j}(\xi) - (1 - y_{i,j,i',j'}) H \qquad \forall m \in \mathcal{M}, \forall (O_{i,j}, O_{i,j'}) \in \mathcal{I}_m^2, \forall \xi \in \mathcal{S} \tag{6.39}$$

$$y_{i,j,i',j'} \in \{0, 1\} \qquad \forall (i, i') \in \mathcal{J}^2, \forall j \in \{1, \ldots, n_i\}, \forall j' \in \{1, \ldots, n_i'\} \tag{6.40}$$

$$t_{i,j}(\xi) \geq 0 \qquad \forall i \in \mathcal{J}, \forall j \in \{1, \ldots, n_i\}, \forall \xi \in \mathcal{S} \tag{6.41}$$

avec H une borne supérieure du makespan pire cas.

L'objectif (6.35) est la minimisation du makespan $C_{\max}$. Les contraintes (6.36) permettent de calculer le plus grand makespan parmi tous les scénarios réalisables et

donc de déterminer la solution optimale dans le pire des cas. Les contraintes (6.37) garantissent les relations de précédence entre deux opérations consécutives d'un même travail. Les contraintes (6.38) et (6.39) sont les contraintes disjonctives et évitent le chevauchement des opérations ordonnancées sur la même machine. On notera que les contraintes (6.37), (6.38) et (6.39) sont dupliquées pour chaque scénario réalisable. Enfin, les contraintes (6.40) et (6.41) définissent le domaine des variables.

6.4.1.2 Programmation par contraintes

Nous présentons également un modèle de PPC. Il s'inspire aussi d'un modèle conçu pour le problème du job-shop déterministe proposé par IBM dans les exemples de CPO [IBM 2022] et présenté dans le paragraphe 4.3.2.

Il implique les variables de décision suivantes :

— $task_{i,j,\xi}$: variable d'intervalle entre le début et la fin de l'opération $O_{i,j}$ dans le scénario ξ ;

— $seqs_{m,\xi}$: variable de séquence des tâches s'effectuant sur la machine m dans le scénario ξ.

La formulation du modèle est la suivante :

$$\min C_{\max} \tag{6.42}$$

t.q.

$$C_{\max} \geq task_{i,n_i,\xi}.fin \qquad \forall i \in \mathcal{J}, \forall \xi \in \mathcal{S} \tag{6.43}$$

$$EndBeforeStart(task_{i,j,\xi}, task_{i,j+1,\xi}) \qquad \forall i \in \mathcal{J}, \forall j \in \{1, \ldots, n_i - 1\}, \forall \xi \in \mathcal{S} \tag{6.44}$$

$$seqs_{m,\xi}.Add(task_{i,j+1,\xi}) \qquad \forall O_{i,j} \in \mathcal{I}_m, \forall \xi \in \mathcal{S} \tag{6.45}$$

$$NoOverlap(seqs_{m,\xi}) \qquad \forall m \in \mathcal{M}, \forall \xi \in \mathcal{S} \tag{6.46}$$

$$SameSequence(seqs_{m,1}, seqs_{m,\xi}) \qquad \forall m \in \mathcal{M}, \forall \xi \in \mathcal{S} \setminus \{1\} \tag{6.47}$$

Les contraintes (6.43) permettent de déterminer le makespan, qui est égal à la fin de la dernière opération dans le pire des cas. Les contraintes (6.44) assurent les relations de priorité entre deux opérations d'un même travail, dans chaque scénario ξ dans $\mathcal{S}$. Les contraintes (6.45) définissent les variables de séquence pour chaque machine et chaque scénario. Les contraintes (6.46) garantissent que, dans chaque scénario $\xi \in \mathcal{S}$, chaque machine effectue au plus une opération à la fois. Les contraintes (6.47) garantissent, pour une machine donnée $m \in \mathcal{M}$, que la séquence σ_m est la même pour chaque scénario. Le premier scénario $\xi = 1$ est utilisé comme référence, et la contrainte est dupliquée pour chaque scénario et chaque machine.

Dans les formulations étendues présentées dans ce paragraphe, on considère l'ensemble des scénarios $\mathcal{S} = \mathcal{U}$. Il s'agit de l'ensemble des combinaisons possibles des réalisations des paramètres incertains. Le nombre de scénarios croît donc de ma-

nière exponentielle avec le nombre d'opérations, ce qui rend rapidement ces modèles insolubles. Il est possible de construire un modèle équivalent avec un nombre polynomial de contraintes et de variables. Ce modèle, connu sous le nom de modèle compact, est présenté dans le paragraphe suivant.

6.4.2 Modèle compact

Dans le paragraphe 6.3.2, nous avons observé que le problème d'évaluation du scénario pire cas peut être formulé comme un problème de minimisation, représenté par les équations (6.28)–(6.30). En combinant ces équations avec les contraintes du problème d'ordonnancement du job-shop, nous obtenons une formulation compacte (introduite dans le paragraphe 2.3.2) pour le problème de job-shop robuste avec un budget d'incertitude.

Le modèle compact, comme le modèle étendu, est inspiré du modèle disjonctif proposé dans [Manne 1960] pour le problème de job-shop déterministe. Cependant ici, les variables de second niveau ne sont pas dupliquées pour chaque scénario possible. Au lieu de cela, nous introduisons les variables $C^{\gamma}_{i,j}$ qui représentent la date de fin de l'opération $O_{i,j}$ dans le pire des cas, en tenant compte d'au plus γ déviations. Ainsi, dans ce modèle, il y a $\Gamma + 1 \leq n + 1$ variables de ce type par opération $O_{i,j}$. Le nombre de contraintes et de variables croît donc de façon polynomiale avec le nombre total d'opérations n.

$$\min C_{\max} \tag{6.48}$$

t.q.

$$C_{\max} \geq C^{\Gamma}_{i,n_i} \qquad \forall i \in \mathcal{J} \tag{6.49}$$

$$C^{\gamma}_{i,j+1} \geq C^{\gamma}_{i,j} + \bar{p}_{i,j+1} \qquad \forall i \in \mathcal{J}, \forall j \in \{1,\dots,n_i-1\}, \forall \gamma \in \{0,\dots\Gamma\} \tag{6.50}$$

$$C^{\gamma+1}_{i,j+1} \geq C^{\gamma}_{i,j} + \bar{p}_{i,j+1} + \hat{p}_{i,j+1} \qquad \forall i \in \mathcal{J}, \forall j \in \{1,\dots,n_i-1\}, \forall \gamma \in \{0,\dots\Gamma-1\} \tag{6.51}$$

$$C^{\gamma}_{i,j} \geq C^{\gamma}_{i',j'} + \bar{p}_{i,j} - y_{i,j,i',j'}H \qquad \forall m \in \mathcal{M}, \forall (O_{i,j}, O_{i,j'}) \in \mathcal{I}^2_m, \forall \gamma \in \{0,\dots\Gamma\} \tag{6.52}$$

$$C^{\gamma+1}_{i,j} \geq C^{\gamma}_{i',j'} + \bar{p}_{i,j} + \hat{p}_{i,j} - y_{i,j,i',j'}H \qquad \forall m \in \mathcal{M}, \forall (O_{i,j}, O_{i,j'}) \in \mathcal{I}^2_m, \forall \gamma \in \{0,\dots\Gamma-1\} \tag{6.53}$$

$$C^{\gamma}_{i',j'} \geq C^{\gamma}_{i,j} + \bar{p}_{i,j} - (1 - y_{i,j,i',j'})H \qquad \forall m \in \mathcal{M}, \forall (O_{i,j}, O_{i,j'}) \in \mathcal{I}^2_m, \forall \gamma \in \{0,\dots\Gamma\} \tag{6.54}$$

$$C^{\gamma+1}_{i',j'} \geq C^{\gamma}_{i,j} + \bar{p}_{i,j} + \hat{p}_{i,j} - (1 - y_{i,j,i',j'})H \qquad \forall m \in \mathcal{M}, \forall (O_{i,j}, O_{i,j'}) \in \mathcal{I}^2_m, \forall \gamma \in \{0,\dots\Gamma-1\} \tag{6.55}$$

$$C^{0}_{i,1} \geq \bar{p}_{i,1} \qquad \forall i \in \mathcal{J} \tag{6.56}$$

$$C^{\gamma}_{i,1} \geq \bar{p}_{i,1} + \hat{p}_{i,1} \qquad \forall i \in \mathcal{J},\ \forall \gamma \in \{1,\dots\Gamma\} \tag{6.57}$$

L'objectif (6.48) est la minimisation du makespan dans le pire des cas. Les contraintes (6.50) et (6.51) garantissent la relation de précédence entre deux opérations consécutives d'un même travail. Les contraintes (6.52)–(6.55) évitent le chevauchement des opérations ordonnancées sur la même machine. Les contraintes (6.56) et (6.57) garantissent que la date de fin de la première opération de chaque travail est au moins égale à son propre temps de traitement.

6.5 Méthodes de décomposition

Dans ce paragraphe, nous nous intéressons aux méthodes de décomposition pour le problème de job-shop robuste avec budget d'incertitude. En particulier, nous présentons une méthode de décomposition de Benders basée sur la logique ainsi qu'une méthode de génération de colonnes et de contraintes.

Ces deux approches itératives visent à décomposer le problème en un problème maître, permettant de déterminer la séquence d'opérations sur chaque machine ainsi que le makespan pire cas associé, et un sous-problème permettant de déterminer si cette solution est admissible pour l'ensemble des scénarios considérés. Elles suivent le même schéma, décrit par l'algorithme 3.

À chaque itération h, une version relaxée du problème, le problème maître, est résolue pour obtenir une solution de premier niveau $(\bar{\sigma}^*_h, \omega^*_h)$. Celle-ci peut être obtenue à l'aide de l'une des formulations étendues présentées dans le paragraphe 6.4.1 en considérant seulement un sous-ensemble de scénarios $\mathcal{S} \subset \mathcal{U}$. En tant que re-

Algorithme 3 Algorithme de décomposition

Initialisation :
Problème maître ← relaxation du problème robuste global ;
$BI \leftarrow 0$, $BS \leftarrow +\infty$, $h = 1$;
répéter
  $(\bar{\sigma}_h^*, \omega_h^*) \leftarrow$ séquence optimale et makespan associé du problème maître ;
  $BI \leftarrow \max(BI, \omega_h^*)$;
  $(\xi_h^*, \psi_h^*) \leftarrow$ scénario pire cas et makespan associé pour la séquence $\bar{\sigma}_h^*$;
  $BS = \min(BS, \psi_h^*)$;
  si $BI < BS$ **alors**
    Ajouter informations au problème maître ;
    $h \leftarrow h + 1$;
jusqu'à $(\omega_h^* = \psi_h^*)$;
retour $(\bar{\sigma}_h^*, \omega_h^*)$;

laxation, le problème maître fournit une borne inférieure (BI) sur le makespan du problème global. Ensuite, étant donné la solution du problème maître et l'ensemble d'incertitude $\mathcal{U}$, un sous-problème adverse identifie, un scénario pire cas ξ_h^* et le makespan associé ψ_h^*, pour la séquence $\bar{\sigma}_h^*$ fixée par le problème maître, et fournit une borne supérieure (BS) pour le problème.

Les informations relatives à ce scénario sont alors ajoutées au problème maître, ce qui permet d'exclure cette solution de premier niveau lors des prochaines itérations (si la séquence est optimale, seul le makespan est mis à jour dans cette solution de premier niveau). La façon dont sont ajoutées ces informations dépend de la méthode choisie, les détails sont donnés dans le paragraphe 6.5.1 pour l'algorithme de décomposition de Benders basée sur la logique et dans le paragraphe 6.5.2 pour l'algorithme de génération de colonnes et de contraintes. Cette procédure est répétée jusqu'à ce que la solution optimale soit trouvée.

6.5.1 Décomposition de Benders

Pour rappel, la méthode de décomposition de Benders repose sur la génération de coupes d'optimalité ou de faisabilité. Dans notre cas, il est impossible qu'une séquence d'opérations sur les machines obtenue par la résolution du problème maître conduise à une solution infaisable pour l'un des scénarios considérés dans le sous-problème. Ainsi, seules des coupes d'optimalité sont générées dans cette méthode.

La première coupe proposée est basée sur le constat que pour améliorer la valeur du makespan d'une solution, il est nécessaire de modifier au moins une décision relative à la séquence d'opérations. Elle est formulée comme suit :

$$C_{max} \geq \psi_h^*(1 - \sum_{(i,j,i',j') \in \mathcal{D}} (1 - y_{i,j,i',j'})) \tag{6.58}$$

avec $\mathcal{D} = \{(i, j, i', j') \mid O_{i,j}$ avant $O_{i',j'}$ dans $\bar{\sigma}_h^*\}$.

Elle indique que le makespan pire cas, pour l'ensemble d'incertitude considéré, ne peut être inférieur à ψ_h^* que si au moins une des variables qui définissent la position relative entre deux opérations d'une même machine est modifiée. Plus précisément, si l'ensemble de ces variables conserve les mêmes valeurs qu'à l'itération h, alors la somme est nulle et la partie de droite de l'inégalité prend la valeur ψ_h^*. À l'inverse, si au moins une variable $y_{i,j,i',j'}$ prend une valeur différente, alors la partie de droite de l'inégalité est négative ou nulle et la coupe est inactive. Cette coupe implique donc l'ensemble des disjonctions et ne permet d'exclure qu'une seule solution du problème maître. Cependant, il est possible de créer une coupe plus générale en considérant uniquement les opérations présentes dans le chemin critique, c'est-à-dire le plus long chemin obtenu par la résolution du sous-problème.

En considérant l'ensemble $\mathcal{C}$ des paires d'opérations successives, traitées par la même machine et appartenant au chemin critique de la solution du sous-problème, la formulation de cette coupe est la suivante :

$$C_{max} \geq \psi_h^*(1 - \sum_{(i,j,i',j') \in \mathcal{C}} (1 - y_{i,j,i',j'})). \tag{6.59}$$

6.5.2 Génération de colonnes et de contraintes

Pour l'algorithme de génération de colonnes et de contraintes, l'ajout d'informations sur le scénario pire cas consiste à générer les variables de décision de deuxième niveau correspondantes ainsi que les contraintes associées.

6.5.2.1 Programmation linéaire en nombres entiers

Dans le cadre de la PLNE, la génération de colonnes et de contraintes revient à ajouter au problème maître et pour le scénario ξ_h^*, le bloc de variables suivant :

$$t_{i,j}(\xi_h^*) \quad \forall i \in \mathcal{J}, \forall j \in \{1, \ldots, n_i\}$$

ainsi que les contraintes associées, à savoir :

$$C_{\max} \geq t_{i,n_i}(\xi_h^*) + p_{i,n_i}(\xi_h^*) \qquad \forall i \in \mathcal{J} \tag{6.60}$$

$$t_{i,j+1}(\xi_h^*) \geq p_{i,j}(\xi_h^*) + t_{i,j}(\xi_h^*) \qquad \forall i \in \mathcal{J}, \forall j \in \{1, \ldots, n_i - 1\} \tag{6.61}$$

$$t_{i,j}(\xi_h^*) \geq t_{i',j'}(\xi_h^*) + p_{i',j'}(\xi_h^*) - y_{i,j,i',j'}H \qquad \forall m \in \mathcal{M}, \forall (O_{i,j}, O_{i,j'}) \in \mathcal{I}_m^2 \tag{6.62}$$

$$t_{i',j'}(\xi_h^*) \geq t_{i,j}(\xi_h^*) + p_{i,j}(\xi_h^*) - (1 - y_{i,j,i',j'})H \qquad \forall m \in \mathcal{M}, \forall (O_{i,j}, O_{i,j'}) \in \mathcal{I}_m^2 \tag{6.63}$$

6.5.2.2 Programmation par contraintes

Dans le problème maître formulé à l'aide de la PPC, cela revient à ajouter les blocs de variables :

$$task_{i,j,\xi_h^*} \quad \forall i \in \mathcal{J}, \forall j \in \{1, \ldots, n_i\}$$

et

$$seqs_{m,\xi_h^*} \quad \forall m \in \mathcal{M}$$

ainsi que les contraintes associées, à savoir :

$$C_{\max} \geq task_{i,|\mathcal{M}|,\xi_h^*}.fin \quad \forall i \in \mathcal{J} \tag{6.64}$$

$$EndBeforeStart(task_{i,j,\xi_h^*}, task_{i,j,\xi_h^*}) \quad \forall i \in \mathcal{J}, \forall j \in \{1, \ldots, n_i - 1\} \tag{6.65}$$

$$NoOverlap(seqs_{m,\xi_h^*}) \quad \forall m \in \mathcal{M} \tag{6.66}$$

$$SameSequence(seqs_{m,1}, seqs_{m,\xi_h^*}) \quad \forall m \in \mathcal{M} \tag{6.67}$$

6.6 Problème flexible

L'ensemble des méthodes exposées dans ce chapitre peuvent être appliquées à la version flexible du problème de job-shop robuste. Pour rappel, le problème de job-shop flexible est une extension du problème de job-shop dans lequel plusieurs machines sont potentiellement capables d'exécuter une même opération. On note alors $\mathcal{M}_{i,j} \in \mathcal{M}$ l'ensemble des machines en mesure de traiter une opération $O_{i,j}$ et $p_{i,j,m}$ le temps de traitement nécessaire pour traiter cette opération sur la machine $m \in \mathcal{M}$.

Le problème de job-shop flexible combine donc à la fois le problème d'affectation de ressources et le problème d'ordonnancement des opérations dans le temps. Dans le problème robuste considéré dans ce paragraphe, les décisions d'affectation et de séquencement constituent les décisions de premier niveau et, comme pour le problème de job-shop classique, les dates de traitement des opérations sont les décisions de second niveau. Le problème d'évaluation d'un scénario pire cas est donc exactement le même que pour le problème de job-shop non flexible.

6.6.1 Modèle étendu

Nous présentons ici les adaptations apportées aux formulations étendues du job-shop robuste présentées dans le paragraphe 6.4.1, pour prendre en compte la flexibilité des machines.

6.6.1.1 Programmation linéaire en nombres entiers

Dans le modèle PLNE, on introduit les variables, de premier niveau, d'affectation des opérations aux machines, définies comme suit :

$$x_{i,j,m} = \begin{cases} 1 & \text{si l'opération } O_{i,j} \text{ est exécutée sur la machine } m\,; \\ 0 & \text{sinon.} \end{cases}$$

De plus, les contraintes d'affectation (6.68) assurent que chaque opération est associée à exactement une machine.

$$\sum_{m \in \mathcal{M}_{i,j}} x_{i,j,m} = 1 \quad \forall i \in \mathcal{J}, \forall j \in \{1, \ldots, n_i\} \tag{6.68}$$

Enfin, les contraintes disjonctives (6.38) et (6.39) sont remplacées par les deux blocs de contraintes suivants :

$$t_{i',j'}(\xi) + p_{i',j',m}(\xi) - (2 - x_{i,j,m} - x_{i',j',m} + y_{i,j,i',j'})H \leq t_{i,j}(\xi), \\ \forall m \in \mathcal{M}, \forall (O_{i,j}, O_{i,j'}) \in \mathcal{I}_m^2, \forall \xi \in \mathcal{U} \tag{6.69}$$

$$t_{i,j}(\xi) + p_{i,j,m}(\xi) - (3 - x_{i,j,m} - x_{i',j',m} - y_{i,j,i',j'})H \leq t_{i',j'}(\xi), \\ \forall m \in \mathcal{M}, \forall (O_{i,j}, O_{i,j'}) \in \mathcal{I}_m^2, \forall \xi \in \mathcal{U} \tag{6.70}$$

inspirés du modèle disjonctif de [Shen 2018] pour le problème de job-shop flexible déterministe.

6.6.1.2 Programmation par contraintes

Dans le modèle de PPC, nous introduisons les variables d'intervalle optionnelles $mode_{i,j,m,\xi}$, qui représentent la période de traitement de l'opération $O_{i,j}$ sur la machine $m \in \mathcal{M}_{i,j}$ dans le scénario ξ. Ces variables sont optionnelles car leur présence dépend de l'affectation des opérations et, dans une solution donnée, une seule de ces variables est active pour chaque opération et scénario. Les affectations de ces variables sont réalisées grâce aux contraintes suivantes :

$$Alternative(task_{i,j,\xi}, mode_{i,j,m,\xi} : \forall m \in \mathcal{M}_{i,j}) \quad \forall i \in \mathcal{J}, \forall j \in \{1, \ldots, n_i\}, \forall \xi \in \mathcal{U} \tag{6.71}$$

qui assurent que chaque opération est associée à exactement une machine. Enfin, les contraintes :

$$PresenceOf(mode_{i,j,m,0}) \Rightarrow PresenceOf(mode_{i,j,m,\xi}) \\ \forall i \in \mathcal{J}, \forall j \in \{1, \ldots, n_i\}, \forall m \in \mathcal{M}_{i,j}, \forall \xi \in \mathcal{U} \setminus \{0\} \tag{6.72}$$

assurent que l'affectation des opérations aux machines est la même pour chaque scénario. Pour ce faire, le premier scénario $\xi = 0$ est utilisé comme référence.

6.6.2 Modèle compact

Pour le modèle compact, on introduit aussi les variables d'affectation des opérations aux machines $x_{i,j,m}$, définies dans le paragraphe 6.6.1.1, ainsi que les contraintes d'affectation (6.68).

De plus, les contraintes disjonctives sont adaptées pour prendre en compte ces variables d'affectation, en remplaçant les contraintes (6.52)–(6.55) par les contraintes suivantes :

$$C^{\gamma}_{i,j} \geq C^{\gamma}_{i',j'} + \bar{p}_{i',j'} - (2 - x_{i,j,m} - x_{i',j',m} + y_{i,j,i',j'})H \quad \forall m \in \mathcal{M}, \forall (O_{i,j}, O_{i,j'}) \in \mathcal{I}^2_m, \gamma \in \{0, \dots \Gamma\} \quad (6.73)$$

$$C^{\gamma+1}_{i,j} \geq C^{\gamma}_{i',j'} + \bar{p}_{i',j'} + \hat{p}_{i',j'} - (2 - x_{i,j,m} - x_{i',j',m} + y_{i,j,i',j'})H \quad \forall m \in \mathcal{M}, \forall (O_{i,j}, O_{i,j'}) \in \mathcal{I}^2_m, \gamma \in \{0, \dots \Gamma - 1\} \quad (6.74)$$

$$C^{\gamma}_{i',j'} \geq C^{\gamma}_{i,j} + \hat{p}_{i,j} - (3 - x_{i,j,m} - x_{i',j',m} - y_{i,j,i',j'})H \quad \forall m \in \mathcal{M}, \forall (O_{i,j}, O_{i,j'}) \in \mathcal{I}^2_m, \gamma \in \{0, \dots \Gamma\} \quad (6.75)$$

$$C^{\gamma+1}_{i',j'} \geq C^{\gamma}_{i,j} + \hat{p}_{i,j} + \hat{p}_{i,j} - (3 - x_{i,j,m} - x_{i',j',m} - y_{i,j,i',j'})H \quad \forall m \in \mathcal{M}, \forall (O_{i,j}, O_{i,j'}) \in \mathcal{I}^2_m, \gamma \in \{0, \dots \Gamma - 1\} \quad (6.76)$$

qui évitent le chevauchement des opérations ordonnancées sur une même machine

6.6.3 Décomposition de Benders

Les informations sont ajoutées au problème maître, formulé comme un PLNE, sous forme de coupes d'optimalité. Une coupe naïve indique que si toutes les affectations et toutes les disjonctions sont les mêmes qu'à une itération h donnée, alors le makespan est au moins égale à celui obtenu lors de l'évaluation pire cas ψ^*_h. Elle est donnée par :

$$C_{max} \geq \psi^*_h(1 - \sum_{(i,j,i',j') \in \mathcal{D}} (1 - y_{i,j,i',j'}) - \sum_{(i,j,m) \in A} (1 - x_{i,j,m})) \quad (6.77)$$

avec $\mathcal{D} = \{(i,j,i',j') \mid O_{i,j}$ avant $O_{i',j'}$ dans $\bar{\sigma}^*_h\}$ et $A = \{(i,j,m) \mid O_{i,j}$ affectée à la machine $m \in \mathcal{M}_{i,j}$ dans $\bar{\sigma}^*_h\}$. Il est possible d'affiner cette coupe en considérant seulement les opérations du chemin critique :

$$C_{max} \geq \psi^*_h(1 - \sum_{(i,j,i',j') \in \mathcal{D} | O_{i,j} \in \mathcal{C}, O_{i',j'} \in \mathcal{C}} (1 - y_{i,j,i',j'}) - \sum_{(i,j,m) \in A | O_{i,j} \in \mathcal{C}} (1 - x_{i,j,m})) \quad (6.78)$$

6.6.4 Génération de colonnes et de contraintes

Pour l'algorithme de génération de colonnes et de contraintes, l'ajout d'information revient à ajouter le scénario pire cas ξ_h^* à $\mathcal{S}$ l'ensemble des scénarios considérés dans le problème maître.

Ainsi pour le modèle PLNE, cela revient à ajouter le bloc de variables

$$t_{i,j}(\xi_h^*) \quad \forall i \in \mathcal{J}, \forall j \in \{1, \dots, n_i\}$$

ainsi que les contraintes (6.60, 6.61, 6.69 et 6.70) associées.

Pour le modèle PPC, les blocs de variables suivants doivent être générés :

$$mode_{i,j,m,\xi_h^*} \quad \forall i \in \mathcal{J}, \forall j \in \{1, \dots, n_i\}, \forall m \in \mathcal{M}_{i,j}$$

$$task_{i,j,\xi_h^*} \quad \forall i \in \mathcal{J}, \forall j \in \{1, \dots, n_i\}$$

et

$$seqs_{m,\xi_h^*} \quad \forall m \in \mathcal{M}$$

ainsi que les contraintes (6.43, 6.44, 6.46, 6.47 et 6.71) associées.

6.7 Expérimentations numériques

Dans ce paragraphe, nous procédons à une évaluation numérique des différentes méthodes présentées dans le chapitre pour résoudre le problème de job-shop robuste (non flexible), en considérant un ensemble d'incertitude discret défini par un budget, dans les conditions expérimentales exposées dans le paragraphe 4.4.

La première méthode, notée *Compact*, consiste à résoudre le modèle compact présenté dans le paragraphe 6.4.2.

La deuxième méthode, notée *Benders*, utilise une décomposition de Benders, introduite dans le paragraphe 6.5.1. Le problème maître est résolu à l'aide du modèle PLNE présenté dans le paragraphe 6.4.1.1 pour le problème de job-shop robuste.

Deux autres méthodes sont basées sur la génération de colonnes et de contraintes, introduites dans le paragraphe 6.5.2. Elles sont notées CCG_{PLNE} et CCG_{PPC}. Dans la méthode CCG_{PLNE}, le problème maître est résolu à l'aide du modèle PLNE présenté dans le paragraphe 6.4.1.1. Dans la méthode CCG_{PPC}, le problème maître est résolu à l'aide du modèle de PPC présenté dans le paragraphe 6.4.1.2.

6.7.1 Résultats, instances de petite et moyenne tailles

Dans un premier temps, nous utilisons un ensemble de 45 instances de petite taille et 45 instances de taille moyenne pour nos tests numériques. Les instances de petite taille ont été générées avec un nombre de travaux variant entre 4 et 6, tandis que pour les instances de taille moyenne, le nombre de travaux varie entre 7 et 9. Le nombre de machines varie quant à lui entre 4 et 6 pour l'ensemble des instances.

Les durées nominales $\bar{p}_{i,j}$ et les déviations maximales $\hat{p}_{i,j}$ de chaque opération $O_{i,j}$ sont des entiers générés aléatoirement à partir d'une distribution uniforme $\texttt{Unif}(1, 20)$. Pour chaque combinaison de paramètres, cinq instances ont été générées, ce qui donne un total de 90 instances. Chaque instance est testée en faisant varier le budget d'incertitude selon quatre ratios : 5 %, 10 %, 15 % et 20 %, ce qui donne un total de 360 expériences par méthode.

Le tableau 6.2 présente les performances des différentes méthodes sur des instances de job-shop de petite et moyenne taille, classées par taille d'instance.

$\|\mathcal{J}\|$	$\|\mathcal{M}\|$	*Compact*		*Benders*		CCG_{PLNE}		CCG_{PPC}	
		#opti.	t (s)	#opti.	t (s)	#opti.	t (s)	#opti.	t (s)
4	4	20	0.19	20	0.43	20	0.27	20	0.14
4	5	20	0.28	20	0.56	20	0.64	20	0.7
4	6	20	0.2	20	0.57	20	0.72	20	0.37
5	4	20	0.42	20	31.55	20	1.85	20	0.66
5	5	20	0.45	19	3.88	20	2.42	20	1.45
5	6	20	0.45	18	3.45	20	9.07	20	18.18
6	4	20	1.09	20	420.08	20	10.45	20	3.76
6	5	20	1.36	20	187.05	20	117.45	20	99.62
6	6	20	2.39	20	461.39	19	318.83	20	314.13
7	4	20	7.01	8	1018.85	19	219.45	20	220.39
7	5	20	8.1	10	430.8	16	229.13	18	126.83
7	6	20	59.44	5	995.98	10	463.2	13	720.04
8	4	18	614.37	0	-	16	199.86	20	32.9
8	5	20	123.93	0	-	8	439.96	14	772.53
8	6	20	314.32	0	-	6	697.24	11	254.55
9	4	6	1051.24	0	-	17	502.85	20	12.23
9	5	9	749.57	0	-	6	766.1	16	131.94
9	6	12	1580.57	0	-	4	187.6	8	80.37

Tableau 6.2 – Performances de méthodes pour les instances de job-shop robuste de petite et moyenne taille, selon la taille des instances

Comme attendu, nous constatons que la difficulté des instances augmente avec leur taille, ce qui se traduit par une diminution du nombre d'instances résolues et une augmentation du temps de calcul. En analysant les performances des méthodes de génération de colonnes et de contraintes, nous constatons que les deux approches, basées sur la PLNE et la PPC obtiennent des résultats similaires pour les instances de petite taille. Cependant, pour les instances de taille moyenne, la méthode CCG_{PPC} se révèle plus efficace en termes de nombre d'instances résolues de manière optimale. Par ailleurs, la méthode *Benders* n'arrive à résoudre que très peu d'instances de taille moyenne. Pour ces instances, la méthode la plus performante est *Compact*. Il semble que la méthode *Compact* soit plus adaptée aux instances de forme carrée, où le nombre de machines est proche du nombre de travaux. En revanche, pour les instances où le nombre de machines est plus petit par rapport au nombre de travaux, la méthode CCG_{PPC} est préférée.

Le tableau 6.3 permet de visualiser la performance des méthodes en fonction du budget d'incertitude.

budget	*Compact*		*Benders*		CCG_{PLNE}		CCG_{PPC}	
	#opti.	t (s)	#opti.	t (s)	#opti.	t (s)	#opti.	t (s)
5%	88	79.2	52	131	88	95.4	90	11.6
10%	82	184	49	122	71	138	83	192
15%	77	139	51	243	66	215	74	122
20%	78	266	48	301	56	247	73	220
total	325	164.5	200	198	281	164.3	320	131.6

Tableau 6.3 – Performances de méthodes pour les instances de job-shop robuste de petite et moyenne taille, selon le budget d'incertitude ; #instances = 90

De manière générale, le nombre d'instances résolues diminue à mesure que le budget d'incertitude pris en compte augmente. Cette observation s'explique par le fait que plus le budget d'incertitude est grand, plus l'ensemble d'incertitude à considérer est étendu. Cependant, il est intéressant de noter que cette tendance est plus marquée pour les modèles CCG_{PPC} et CCG_{PLNE} par rapport aux autres approches pour ces instances.

La figure 6.5 offre une visualisation des profils de performance des différentes méthodes sur les instances de job-shop robuste de taille moyenne. Elle met en évidence la proportion d'instances résolues à l'optimum en fonction du temps (figure 6.5a), ainsi que la proportion d'instances résolues avec un écart d'optimalité inférieur à une valeur donnée (figure 6.5b). Nous observons que les méthodes CCG_{PPC} et *Compact* obtiennent des résultats très comparables en termes de temps de calcul jusqu'à l'optimum. De plus, ces deux méthodes surpassent nettement les autres méthodes pour ce critère. Nous constatons aussi une nette domination de la méthode CCG_{PPC} sur les autres méthodes en termes d'écart d'optimalité. La méthode CCG_{PLNE} présente des performances comparables à celle de la méthode *Compact*.

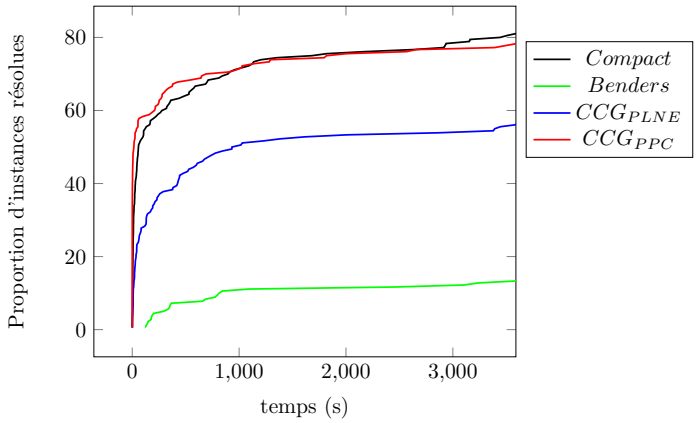
6.7.2 Instances de la littérature

Nous considérons à présent 58 instances classiques de la littérature pour le problème de job-shop déterministe (présentées dans le paragraphe 1.4.2). Nous les adaptons au contexte robuste en générant aléatoirement des valeurs de déviation des durées opératoires.

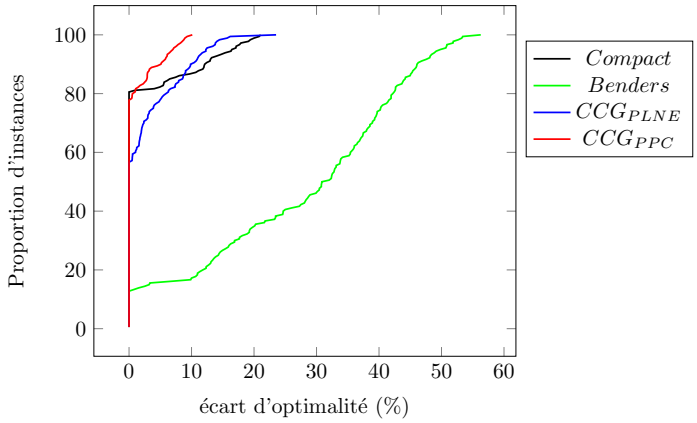
Pour chaque opération $O_{i,j}$, la durée de traitement fournie par l'instance est considérée comme la valeur nominale $\bar{p}_{i,j}$. Ensuite, nous générons aléatoirement une valeur de déviation $\hat{p}_{i,j}$ dans un intervalle de 25% à 100% de la durée nominale maximale $\bar{p}_{max} = \max\{\bar{p}_{i,j} | i \in \mathcal{J}, O_{i,j} \in \{1, \ldots, n_i\}\}$.

Nous testons tous les modèles en faisant varier le budget d'incertitude selon quatre ratios : 5 %, 10 %, 15 % et 20 %, soit un total de 232 expériences par méthode.

Pour ces instances, nous observons qu'il est difficile d'obtenir des solutions optimales. L'ensemble des méthodes ne parviennent à trouver que très peu de solutions



(a) Proportion d'instances résolues à l'optimum en fonction du temps



(b) Proportion d'instances résolues avec un écart d'optimalité inférieur à une valeur donnée

FIGURE 6.5 – Profils de performance sur les instances de job-shop robuste de taille moyenne

optimales parmi les 232 instances considérées. Par conséquent, afin de mettre en évidence les performances relatives des différentes méthodes, les résultats du tableau 6.4 sont présentés en termes de nombre de meilleures solutions trouvées par rapport aux autres méthodes, ainsi que l'écart d'optimalité des solutions obtenues.

$\|\mathcal{J}\|$	$\|\mathcal{M}\|$	#	*Compact*		*Benders*		CCG_{PLNE}		CCG_{PPC}	
			#Best	Gap (%)	#Best	Gap (%)	#Best	Gap (%)	#Best	Gap (%)
6	6	4	4	0	4	0	3	2	4	0
10	5	20	18	15.28	10	48.2	7	11.13	15	7.67
10	10	72	29	21.87	30	52.39	9	27.89	11	24.42
15	5	20	12	44.15	19	66.45	6	57.35	19	21
15	10	20	1	44	16	62.9	1	58.15	2	33.8
15	15	20	0	40.7	15	61.95	3	59.5	2	42.4
20	5	24	2	61.92	16	75.67	4	70.5	22	11.5
20	10	20	0	50.9	4	71.15	1	67.65	15	31.85
20	15	12	0	-	3	66.25	0	65.83	9	47.25
30	10	20	0	-	0	80.25	0	79.35	20	19.05

Tableau 6.4 – Nombre de meilleures solutions réalisables trouvées et écart d'optimalité moyen par méthode et taille d'instance ; instances de job-shop de la littérature

Pour les instances de petite taille (6×6), toutes les méthodes ont réussi à trouver la solution optimale (sauf pour une pour la méthode CCG_{PLNE}). Pour les instances jusqu'à 15 machines la méthode *Benders* a généralement obtenu le plus grand nombre de meilleures solutions, mais avec des écarts d'optimalité relativement élevés. Enfin, pour les plus grandes instances, la méthode CCG_{PPC} obtient le plus grand nombre de meilleures solutions.

La figure 6.6 met en évidence la proportion d'instances résolues avec un écart d'optimalité inférieur à une valeur donnée. On constate alors que pour cet indicateur, la méthode CCG_{PPC} est nettement plus performante que les autres. Il est également intéressant de noter que pour les instances les plus complexes, les méthodes CCG_{PLNE} et *Benders* obtiennent exactement le même écart d'optimalité. Cela peut s'expliquer par le fait que ces instances sont extrêmement difficiles et que le premier problème maître de ces algorithmes de décomposition n'a pas suffisamment de temps pour être résolu dans le temps imparti. Par conséquent, il n'y a qu'une seule itération effectuée, et les formulations pour cette première itération sont les mêmes pour les deux méthodes.

6.8 Conclusion

Dans ce chapitre, nous avons présenté plusieurs formulations pour le problème du job-shop robuste en considérant un ensemble d'incertitude défini par un budget. Parmi les formulations directes, la formulation compacte présente de bons résultats, mais ils sont cependant surpassés par les méthodes de décomposition et plus particulièrement par celle qui présente un problème maître en modèle PPC. Une extension à la version flexible de ce problème d'ordonnancement a également été

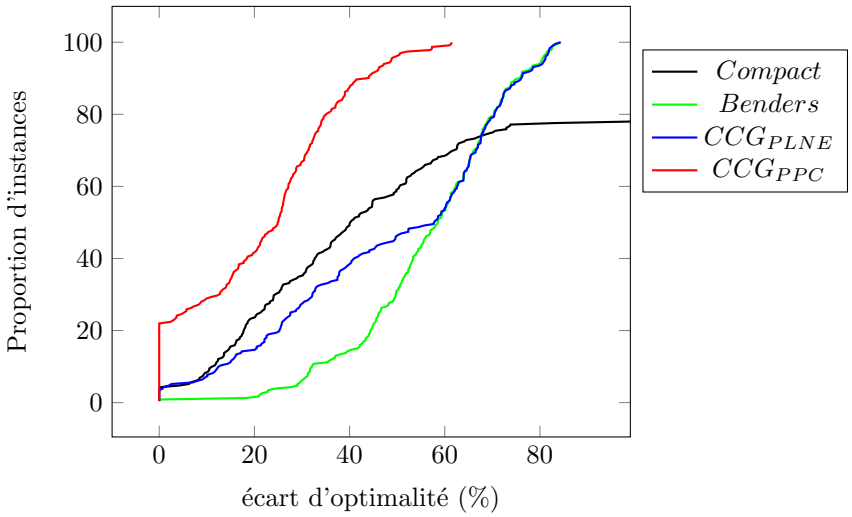


FIGURE 6.6 – Profil d'écart d'optimalité pour les instances de la littérature

proposée. L'ensemble des méthodes présentées sont appliquées à un autre problème d'ordonnancement dans le chapitre suivant, le problème de flow-shop de permutation.

Chapitre 7

Le problème de flow-shop de permutation robuste

Sommaire

7.1 Introduction

Dans ce chapitre, nous présentons nos contributions dans le cadre de la version robuste du problème de flow-shop de permutation présenté dans le paragraphe 1.2.1.2. Nous considérons des durées d'opérations incertaines dont la modélisation utilise un budget d'incertitude tel qu'introduit dans [Bertsimas 2004]. Nous abordons le problème selon une approche à deux niveaux, où la séquence des travaux sur les machines constitue le premier niveau de décision et doit être décidée avant la révélation de l'incertitude. Les décisions de second niveau correspondent aux dates de traitement des opérations, qui peuvent être adaptées en fonction des réalisations des durées d'opérations. L'objectif du problème consiste à trouver une séquence de travaux minimisant le makespan dans le pire cas sur l'ensemble d'incertitude considéré.

7.2 Définition du problème

On note Σ l'ensemble des solutions de premier niveau réalisables et σ un élément de cet ensemble, où σ est une séquence de travaux. On note $i \prec_\sigma i'$ une relation de précédence induite par σ, qui signifie que le travail i précède le travail i' sur l'ensemble des machines $m \in \mathcal{M}$ dans la séquence σ.

Soit $\tau = (t^\sigma_{i,m}(\xi) | i \in \mathcal{J}, m \in \mathcal{M})$ un vecteur tel que $t^\sigma_{i,m}(\xi) \in \mathbb{R}$ est un réel désignant la date de début du travail i, sur la machine m, sous la séquence σ, dans le scénario ξ. Le problème peut être alors défini comme suit :

$$\min_{\sigma \in \Sigma,\, C_{\max} \in \mathbb{R}} \max_{\xi \in \mathcal{U}^\Gamma} \min_{\tau \in \mathbb{R}^n} C_{\max} \tag{7.1}$$

t.q.

$$C_{\max} \geq t^\sigma_{i,|\mathcal{M}|}(\xi) + p_{i,|\mathcal{M}|}(\xi) \qquad \forall i \in \mathcal{J} \tag{7.2}$$

$$t^\sigma_{i,m+1}(\xi) \geq t^\sigma_{i,m}(\xi) + p_{i,m}(\xi) \qquad \forall i \in \mathcal{J},\, \forall m \in \mathcal{M} \setminus \{M_{|\mathcal{M}|}\} \tag{7.3}$$

$$t^\sigma_{i',m}(\xi) \geq t^\sigma_{i,m}(\xi) + p_{i,m}(\xi) \qquad \forall (i,i') \in \mathcal{J} \times \mathcal{J},\ t.q.\ i \prec_\sigma i', \forall m \in \mathcal{M} \tag{7.4}$$

L'objectif (7.1) est de trouver une séquence de travaux σ minimisant le makespan dans le scénario pire cas, en considérant que les dates de traitement peuvent s'adapter au scénario. Les contraintes (7.2) définissent le makespan pire cas. Les contraintes (7.3) sont les contraintes de précédence entre deux opérations successives d'un même travail. Les contraintes (7.4) sont les contraintes disjonctives sur les machines.

7.3 Formulations directes

Dans ce paragraphe, nous abordons deux approches directes pour résoudre le problème de flow-shop de permutation robuste. La première approche est une formulation étendue dans laquelle le nombre de variables et de contraintes est relatif au nombre de scénarios considérés. La deuxième approche est une formulation compacte dans laquelle la taille du problème dépend seulement du nombre maximal de déviations considéré.

7.3.1 Modèles étendus

Nous présentons ici deux formulations étendues du problème de flow-shop de permutation robuste, l'une basée sur la PLNE, l'autre sur la PPC.

7.3.1.1 Programmation linéaire en nombres entiers

Il existe plusieurs formulations PLNE pour le problème de flow-shop de permutation déterministe dans la littérature. Ces formulations peuvent être classées

en deux catégories distinctes. La première repose sur un modèle disjonctif introduit dans [Manne 1960] pour le problème de job-shop et utilise des variables qui décrivent la position relative de deux opérations traitées par une même machine ([Tseng 2001]). Dans la seconde catégorie, les contraintes disjonctives sont modélisées à l'aide de variables de rang qui sont associées à la position explicite de chaque opération dans la séquence ([Wagner 1959, Wilson 1989, Stafford 1988]). Selon une étude expérimentale menée dans [Tseng 2004], les formulations basées sur les variables de rang, notamment le modèle de Wagner et celui de Wilson, se sont révélées plus efficaces pour ce problème. Sur la base de ces deux modèles, une contrepartie robuste a été développée récemment pour le problème de flow-shop de permutation à deux machines [Levorato 2022]. Les résultats expérimentaux ont montré que le modèle de Wagner était particulièrement performant pour le problème robuste. Ce modèle, initialement développé pour le problème de flow-shop déterministe à trois machines, a également été étendu pour un nombre quelconque de machines dans [Stafford 1988].

Sur la base de ces travaux, nous proposons un modèle pour le problème robuste avec budget d'incertitude. Ce modèle utilise les variables de décision suivantes :

— $z_{i,l}$: variable binaire associée au rang d'un travail dans la séquence, avec :

$$z_{i,l} = \begin{cases} 1 & \text{si le travail } i \in \mathcal{J} \text{ est traité au rang } l\,; \\ 0 & \text{sinon} \end{cases}$$

— $x_{l,m}(\xi)$: temps d'inactivité sur la machine m, avant le début de traitement du $l^{ième}$ travail dans le scénario ξ ;

— $y_{l,m}(\xi)$: temps d'inactivité du $l^{ième}$ travail, après son traitement sur la machine m dans le scénario ξ.

Il est défini comme suit :

$$\min C_{\max} \tag{7.5}$$

$$\sum_{i=1}^{|\mathcal{J}|} z_{i,l} = 1 \qquad \forall l \in \{1, \ldots, |\mathcal{J}|\} \tag{7.6}$$

$$\sum_{l=1}^{|\mathcal{J}|} z_{i,l} = 1 \qquad \forall i \in \mathcal{J} \tag{7.7}$$

$$\begin{aligned} &\sum_{i=1}^{|\mathcal{J}|}(p_{i,m}(\xi)z_{i,l+1}) + y_{l+1,m}(\xi) + x_{l+1,m}(\xi) = \\ &\sum_{i=1}^{|\mathcal{J}|}(p_{i,m+1}(\xi)z_{i,l}) + y_{l,m}(\xi) + x_{l+1,m+1}(\xi) \qquad \forall l \in \{1, \ldots, |\mathcal{J}| - 1\}, \\ &\qquad\qquad \forall m \in \mathcal{M} \setminus \{M_{|\mathcal{M}|}\},\ \forall \xi \in \mathcal{S} \end{aligned} \tag{7.8}$$

$$\sum_{m'=1}^{m-1} \sum_{i=1}^{|\mathcal{J}|}(p_{i,m'}(\xi)z_{i,1}) = x_{1,m}(\xi) \qquad \forall m \in \mathcal{M},\ \forall \xi \in \mathcal{S} \tag{7.9}$$

$$C_{\max} \geq \sum_{i=1}^{|\mathcal{J}|} p_{i,|\mathcal{M}|}(\xi) + \sum_{l=1}^{|\mathcal{J}|} x_{l,|\mathcal{M}|}(\xi) \qquad \forall \xi \in \mathcal{S} \tag{7.10}$$

$$z_{i,l} \in \{0,1\} \qquad \forall i \in \mathcal{J},\ \forall l \in \mathcal{J} \tag{7.11}$$

$$x_{l,m}(\xi) \geq 0,\ x_{l,1}(\xi) = 0, y_{l,m}(\xi) \geq 0,\ y_{1,m}(\xi) = 0 \qquad \forall l \in \mathcal{J},\ \forall m \in \mathcal{M} \tag{7.12}$$

Les contraintes (7.6) et (7.7) garantissent que chaque travail est affecté à un rang et un seul dans la séquence et que chaque rang est occupé par exactement un travail. Les contraintes (7.8) et (7.9) garantissent le respect des précédences à la fois pour les travaux et les machines, elles sont nommées *job-adjacency-machine-linkage (JAML) constraints* dans la littérature [Tseng 2001]. Les contraintes (7.10) calculent le makespan pire cas sur l'ensemble des scénarios considérés. Enfin, les contraintes (7.11) et (7.12) définissent le domaine des variables.

7.3.1.2 Programmation par contraintes

Le modèle de PPC proposé ici est similaire à celui utilisé pour le problème de job-shop robuste présenté dans le chapitre 6, à une différence près : la séquence de travaux sur chaque machine doit être identique. Pour tenir compte de cette contrainte supplémentaire, nous ajoutons la contrainte suivante au modèle PPC (6.42)–(6.47) :

$$SameSequence(seqs_{1,\xi}, seqs_{m,\xi}) \qquad \forall m \in \mathcal{M} \setminus \{M_1\},\ \forall \xi \in \mathcal{S} \tag{7.13}$$

La première machine M_1 est utilisée comme référence, et la contrainte est dupliquée pour chaque scénario et chaque machine.

7.3.2 Modèle compact

Le modèle compact présenté dans ce paragraphe est basé sur celui proposé dans [Wilson 1989] pour la version déterministe du problème. Tout comme pour le problème de job-shop (paragraphe 6.4.2), nous combinons ce modèle avec le problème d'évaluation du scénario pire cas, formulé comme un problème de minimisation. Cela nous permet d'obtenir une formulation compacte, pour le problème de flow-shop de permutation robuste avec budget d'incertitude.

Ce modèle utilise les variables de rang $z_{i,l}, \forall i \in \mathcal{J}, \forall l \in \{1, \ldots, |\mathcal{J}|\}$ définies dans le paragraphe 7.3.1.1. De plus, on introduit les variables $C^{\gamma}_{l,m}$ qui représentent la date de fin de traitement pire cas du $l^{ième}$ travail sur la machine m, en considérant au plus γ déviations. Le modèle est défini comme suit :

$$\min C^{\Gamma}_{|\mathcal{J}|,|\mathcal{M}|} \tag{7.14}$$

t.q.

$$\sum_{i\in\mathcal{J}}^{|\mathcal{J}|} z_{i,l} = 1 \qquad \forall l \in [0, |\mathcal{J}|] \tag{7.15}$$

$$\sum_{l=1}^{|\mathcal{J}|} z_{i,l} = 1 \qquad \forall i \in \mathcal{J} \tag{7.16}$$

$$C^{\gamma}_{l,m} \geq C^{\gamma}_{l,m-1} + \sum_{i\in\mathcal{J}} (\bar{p}_{i,m} z_{i,l}) \qquad \forall l \in \{1 \dots |\mathcal{J}|\}, \forall m \in \mathcal{M} \setminus \{M_1\},$$
$$\forall \gamma \in \{0, \dots \Gamma\} \tag{7.17}$$

$$C^{\gamma}_{l,m} \geq C^{\gamma-1}_{l,m-1} + \sum_{i\in\mathcal{J}} ((\bar{p}_{i,m} + \hat{p}_{i,m}) z_{i,l}) \qquad \forall l \in \{1 \dots |\mathcal{J}|\}, \forall m \in \mathcal{M} \setminus \{M_1\},$$
$$\forall \gamma \in \{0, \dots \Gamma - 1\} \tag{7.18}$$

$$C^{\gamma}_{l,m} \geq C^{\gamma}_{l-1,m} + \sum_{i\in\mathcal{J}} (\bar{p}_{i,m} z_{i,l}) \qquad \forall l \in \{2 \dots |\mathcal{J}|\}, \forall m \in \mathcal{M},$$
$$\forall \gamma \in \{0, \dots \Gamma\} \tag{7.19}$$

$$C^{\gamma}_{l,m} \geq C^{\gamma-1}_{l-1,m} + \sum_{i\in\mathcal{J}} ((\bar{p}_{i,m} + \hat{p}_{i,m}) z_{i,l}) \qquad \forall l \in \{2 \dots |\mathcal{J}|\}, \forall m \in \mathcal{M},$$
$$\forall \gamma \in \{0, \dots \Gamma - 1\} \tag{7.20}$$

$$C^{0}_{1,1} \geq \sum_{i\in\mathcal{J}} (\bar{p}_{i,1} z_{i,1}) \tag{7.21}$$

$$C^{\gamma}_{1,1} \geq \sum_{i\in\mathcal{J}} ((\bar{p}_{i,1} + \hat{p}_{i,1}) z_{i,1}) \qquad \forall \gamma \in \{1, \dots \Gamma\} \tag{7.22}$$

L'objectif (7.14) est la minimisation de la date de fin du dernier travail $|\mathcal{J}|$ sur la dernière machine $|\mathcal{M}|$. Les contraintes (7.15) et (7.16) garantissent que chaque travail est affectée à un rang et un seul dans la séquence et que chaque rang est occupé par exactement un travail. Les contraintes (7.17) et (7.18) garantissent la relation de précédence entre deux opérations consécutives d'un même travail. Les contraintes (7.19) et (7.20) assurent la relation de précédence entre deux opérations consécutives sur une même machine. Les contraintes (7.21) et (7.22) définissent la date de fin du premier travail sur la première machine.

7.4 Méthodes de décomposition

Pour les méthodes de décomposition, le schéma de décomposition est similaire à celui présenté pour le problème de job-shop dans le paragraphe 6.5. Il comprend un problème maître et un sous-problème adverse. Le problème maître est chargé de déterminer la séquence des travaux sur les machines, et il peut être résolu à l'aide de l'une des formulations présentées dans le paragraphe 7.3.1 en considérant uniquement un sous-ensemble de scénarios $\mathcal{S} \subset \mathcal{U}^{\Gamma}$. Le sous-problème adverse évalue le scénario pire cas pour une séquence de travaux donnée, et il peut être résolu à l'aide des méthodes décrites dans le paragraphe 6.3. Les informations relatives au scénario

pire cas sont ensuite intégrées dans le problème maître selon la méthode choisie. Les détails de cette intégration sont décrits dans le paragraphe 7.4.1 pour l'algorithme de décomposition de Benders basé sur la logique, et dans le paragraphe 7.4.2 pour l'algorithme de génération de colonnes et de contraintes.

7.4.1 Décomposition de Benders basée sur la logique

Pour la décomposition de Benders basée sur la logique, les informations relatives au scénario pire cas sont intégrées sous la forme de coupes logiques. Soient σ_k^* la séquence de travaux fixée par le problème maître à l'itération k et ψ_k^* le makespan pour cette séquence dans le scénario pire cas trouvé par la résolution du sous-problème. Une coupe de Benders triviale indique que le makespan pire cas est au moins égal à ψ_k^*, lorsque toutes les affectations de rang restent les mêmes que dans la séquence σ_k^*. En d'autres termes, toute solution permettant d'obtenir un meilleur makespan implique un changement dans l'affectation de rang d'au moins l'un des travaux. Elle est décrite comme suit :

$$C_{\max} \geq \psi_k^*(1 - \sum_{l=1}^{|\mathcal{J}|}(1 - z_{[l],l})) \tag{7.23}$$

où $[l]$ désigne le $l^{ième}$ travail dans la séquence $\bar{\sigma}_k^*$.

À chaque itération h, une telle coupe est générée et ajoutée à la formulation PLNE du problème maître.

7.4.2 Génération de contraintes et de colonnes

Dans l'algorithme de génération de colonnes et de contraintes, ajouter des informations consiste simplement à ajouter à l'ensemble des scénarios considérés le scénario pire cas ξ_k^* trouvé lors de la résolution du sous-problème. Plus précisément, il s'agit de générer les variables et les contraintes relatives à ce scénario dans le problème maître.

Dans le cas d'un problème maître formulé à l'aide de la PLNE, cela consiste à générer les variables de second niveau $x_{l,m}(\xi_k^*)$ et $y_{l,m}(\xi_k^*)$, pour chaque travail $i \in \mathcal{J}$ et chaque machine $m \in \mathcal{M}$, ainsi que les blocs de contraintes (7.8), (7.9) et (7.10) associés.

Si le problème maître est formulé à l'aide de la PPC, alors les variables générées sont $task_{i,m,\xi_k^*}$ pour chaque travail $i \in \mathcal{J}$, et chaque machine $m \in \mathcal{M}$, et $seqs_{m,\xi_k^*}$ pour chaque machine $m \in \mathcal{M}$. De plus, les blocs de contraintes (6.43), (6.44), (6.46), (6.47) et (7.13) sont dupliqués pour le scénario ξ_k^*.

7.5 Cas du problème à deux machines

Dans ce paragraphe, on s'intéresse à la complexité du problème de flow-shop de permutation robuste à deux machines avec budget d'incertitude. Ces travaux ont

fait l'objet de la publication [Juvin 2023c].

Le problème déterministe peut être résolu à l'optimum en temps polynomial à l'aide de la règle de Johnson [Johnson 1954], qui construit une solution optimale où le travail i précède le travail i' si $\min(p_{i,1}, p_{i',2}) < \min(p_{i',1}, p_{i,2})$.

Ici, on étudie l'effet de l'ensemble d'incertitude considéré sur l'applicabilité de la règle de Johnson. Plus particulièrement, nous analysons l'effet de deux paramètres, à savoir :

1. les types de budget d'incertitude considérés, budget global ou budget dépendant des machines, définis dans le paragraphe 2.4.2.1 ;
2. le type de déviation :
   - préservant totalement l'ordre des durées de traitement, c.-à-d. : $\forall (i,i') \in \mathcal{J}^2,\ \forall (m,m') \in \mathcal{M}^2,\ \bar{p}_{i,m} < \bar{p}_{i',m'} \Leftrightarrow \bar{p}_{i,m} + \hat{p}_{i,m} < \bar{p}_{i',m'} + \hat{p}_{i',m'}$, ou
   - ne préservant pas l'ordre des durées de traitement, c.-à-d. : $\exists (i,i') \in \mathcal{J}^2,\ \exists (m,m') \in \mathcal{M}^2,$ tel que $\bar{p}_{i,m} < \bar{p}_{i',m'}$ et $\bar{p}_{i,m} + \hat{p}_{i,m} > \bar{p}_{i',m'} + \hat{p}_{i',m'}$.

7.5.1 Budget global et ordre des temps de traitement préservé

Dans ce paragraphe, nous démontrons que, sous certaines conditions, le problème de flow-shop de permutation robuste à deux machines avec un budget d'incertitude peut être résolu de manière optimale en temps polynomial.

Proposition 2 *Si les déviations préservent totalement l'ordre des durées de traitement, alors un ordonnancement respectant la règle de Johnson est optimal pour tout budget d'incertitude global* Γ.

Preuve 3 *Supposons qu'il existe une séquence optimale* σ *telle que les quatre travaux* $\sigma[l-1],\ \sigma[l] = i,\ \sigma[l+1] = i'$ *et* $\sigma[l+2]$ *sont ordonnancés dans cet ordre et telle qu'au moins l'une de ces conditions est vérifiée :*

(i) $\bar{p}_{i,1} > \bar{p}_{i,2}$ *et* $\bar{p}_{i',1} < \bar{p}_{i',2}$

(ii) $\bar{p}_{i,1} < \bar{p}_{i,2}$, $\bar{p}_{i',1} < \bar{p}_{i',2}$ *et* $\bar{p}_{i,1} > \bar{p}_{i',1}$

(iii) $\bar{p}_{i,1} > \bar{p}_{i,2}$, $\bar{p}_{i',1} > \bar{p}_{i',2}$ *et* $\bar{p}_{i,2} < \bar{p}_{i',2}$

Par conséquent, la séquence σ *telle que* i *est avant* i' *ne respecte pas la règle de Johnson. (*$\min(p_{i',1}, p_{i,2}) < \min(p_{i,1}, p_{i',2}) \implies i'$ *avant* i*).*

Soit σ' *la séquence obtenue à partir de* σ *en échangeant la position des travaux* i *et* i'*. Il suffit de montrer que dans chacune des trois conditions (i), (ii) et (iii), le makespan sous* σ' *n'est pas plus grand que sous* σ*.*

Sur la machine M_1*, si l'on considère au plus* γ *déviations, la date de début de traitement au plus tard du travail* $\sigma[l+2] = \sigma'[l+2]$ *(correspondant à la date de fin de traitement du* $l+1^{ième}$ *travail sur la machine* M_1*) est la même sous* σ *et* σ'*. En effet, il s'agit dans les deux cas de la date de fin de traitement du travail* $\sigma[l-1] = \sigma'[l-1]$ *à laquelle on ajoute les durées de traitement des travaux* i *et* i'*, le tout sur la machine* M_1*. On a donc* $C^{\gamma}_{l+1,1}(\sigma) = C^{\gamma}_{l+1,1}(\sigma') = \max(a_1, a_2, a_3)$ *avec :*

$a_1 = C^{\gamma}_{l-1,1}(\sigma) + \bar{p}_{i,1} + \bar{p}_{i',1}$

$a_2 = C^{\gamma-1}_{l-1,1}(\sigma) + \bar{p}_{i,1} + \bar{p}_{i',1} + \max(\hat{p}_{i,1}, \hat{p}_{i',1})$ *si* $\gamma \geq 1$*, 0 sinon*

$a_3 = C^{\gamma-2}_{l-2,1}(\sigma) + \bar{p}_{i,1} + \hat{p}_{i,1} + \bar{p}_{i',1} + \hat{p}_{i',1}$ *si* $\gamma \geq 2$, *0 sinon*

Ainsi, le reste de l'ordonnancement sur le machine M_1 *n'est pas affecté par l'échange de position.*

Étudions à présent l'effet de cet échange sur la machine M_2*. On s'intéresse en particulier à la date de disponibilité de la machine* M_2 *pour le traitement du travail* $\sigma[l+2]$*, à savoir la date de fin de traitement du* $l+1^{ième}$ *travail sur la machine* M_2*,* $C^{\gamma}_{l+1,2}(\sigma)$ *sous la séquence originale* σ *et* $C^{\gamma}_{l+1,2}(\sigma')$ *sous* σ'*.*

Considérant au plus γ *déviations, la date de fin de traitement du* $l+1^{ième}$ *travail sur la machine* M_2 *sous la séquence* σ*,* $C^{\gamma}_{l+1,2}(\sigma)$ *dépend*

— *des contraintes de précédence sur le travail* $l+1$ *et sur la machine* M_2*,*

— *des opérations sur lesquelles les* γ *déviations ont le plus d'impact.*

Ainsi, $C^{\gamma}_{l+1,2}(\sigma) = \max(b_1, b_2, b_3, b_4, b_5, b_6, b_7, b_8, b_9, b_{10}, b_{11})$ *avec*

$b_1 = C^{\gamma}_{l-1,2} + \bar{p}_{i,2} + \bar{p}_{i',2}$

$b_2 = C^{\gamma-1}_{l-1,2} + \bar{p}_{i,2} + \bar{p}_{i',2} + \max(\hat{p}_{i,2}, \hat{p}_{i',2})$ *si* $\gamma \geq 1$, 0 *sinon*

$b_3 = C^{\gamma-2}_{l-1,2} + \bar{p}_{i,2} + \hat{p}_{i,2} + \bar{p}_{i',2} + \hat{p}_{i',2}$ *si* $\gamma \geq 2$, 0 *sinon*

$b_4 = C^{\gamma}_{l-1,1} + \bar{p}_{i,1} + \bar{p}_{i,2} + \bar{p}_{i',2}$

$b_5 = C^{\gamma-1}_{l-1,1} + \bar{p}_{i,1} + \bar{p}_{i,2} + \bar{p}_{i',2} + \max(\hat{p}_{i,1}, \hat{p}_{i,2}, \hat{p}_{i',2})$ *si* $\gamma \geq 1$, 0 *sinon*

$b_6 = C^{\gamma-2}_{l-1,1} + \bar{p}_{i,1} + \bar{p}_{i,2} + \bar{p}_{i',2} + \max(\hat{p}_{i,1} + \hat{p}_{i,2}, \hat{p}_{i,1} + \hat{p}_{i',2}, \hat{p}_{i,2} + \hat{p}_{i',2})$ *si* $\gamma \geq 2$, 0 *sinon*

$b_7 = C^{\gamma-3}_{l-1,1} + \bar{p}_{i,1} + \hat{p}_{i,1} + \bar{p}_{i,2} + \hat{p}_{i,2} + \bar{p}_{i',2} + \hat{p}_{i',2}$ *si* $\gamma \geq 3$, 0 *sinon*

$b_8 = C^{\gamma}_{l-1,1} + \bar{p}_{i,1} + \bar{p}_{i',1} + \bar{p}_{i',2}$

$b_9 = C^{\gamma-1}_{l-1,1} + \bar{p}_{i,1} + \bar{p}_{i',1} + \bar{p}_{i',2} + \max(\hat{p}_{i,1}, \hat{p}_{i',1}, \hat{p}_{i',2})$ *si* $\gamma \geq 1$, 0 *sinon*

$b_{10} = C^{\gamma-2}_{l-1,1} + \bar{p}_{i,1} + \bar{p}_{i',1} + \bar{p}_{i',2} + \max(\hat{p}_{i,1} + \hat{p}_{i',1}, \hat{p}_{i,1} + \hat{p}_{i',2}, \hat{p}_{i',1} + \hat{p}_{i',2})$ *si* $\gamma \geq 2$, 0 *sinon*

$b_{11} = C^{\gamma-3}_{l-1,1} + \bar{p}_{i,1} + \hat{p}_{i,1} + \bar{p}_{i',1} + \hat{p}_{i',1} + \bar{p}_{i',2} + \hat{p}_{i',2}$ *si* $\gamma \geq 3$, 0 *sinon*

De la même manière, la date de fin de traitement du $l+1^{ième}$ *travail sur la machine* M_2 *sous la séquence* σ'*,* $C^{\gamma}_{l+1,2}(\sigma')$ *peut s'exprimer comme* $C^{\gamma}_{l+1,2}(\sigma') = \max(c_1, c_2, c_3, c_4, c_5, c_6, c_7, c_8, c_9, c_{10}, c_{11})$*, avec :*

$c_1 = C^{\gamma}_{l-1,2} + \bar{p}_{i',2} + \bar{p}_{i,2}$

$c_2 = C^{\gamma-1}_{l-1,2} + \bar{p}_{i',2} + \bar{p}_{i,2} + \max(\hat{p}_{i',2}, \hat{p}_{i,2})$ *si* $\gamma \geq 1$, 0 *sinon*

$c_3 = C^{\gamma-2}_{l-1,2} + \bar{p}_{i',2} + \hat{p}_{i',2} + \bar{p}_{i,2} + \hat{p}_{i,2}$ *si* $\gamma \geq 2$, 0 *sinon*

$c_4 = C^{\gamma}_{l-1,1} + \bar{p}_{i',1} + \bar{p}_{i',2} + \bar{p}_{i,2}$

$c_5 = C^{\gamma-1}_{l-1,1} + \bar{p}_{i',1} + \bar{p}_{i',2} + \bar{p}_{i,2} + \max(\hat{p}_{i',1}, \hat{p}_{i',2}, \hat{p}_{i,2})$ *si* $\gamma \geq 1$, 0 *sinon*

$c_6 = C^{\gamma-2}_{l-1,1} + \bar{p}_{i',1} + \bar{p}_{i',2} + \bar{p}_{i,2} + \max(\hat{p}_{i',1} + \hat{p}_{i',2}, \hat{p}_{i',1} + \hat{p}_{i,2}, \hat{p}_{i',2} + \hat{p}_{i,2})$ *si* $\gamma \geq 2$, 0 *sinon*

$c_7 = C^{\gamma-3}_{l-1,1} + \bar{p}_{i',1} + \hat{p}_{i',1} + \bar{p}_{i',2} + \hat{p}_{i',2} + \bar{p}_{i,2} + \hat{p}_{i,2}$ *si* $\gamma \geq 3$, 0 *sinon*

$c_8 = C^{\gamma}_{l-1,1} + \bar{p}_{i',1} + \bar{p}_{i,1} + \bar{p}_{i,2}$

$c_9 = C^{\gamma-1}_{l-1,1} + \bar{p}_{i',1} + \bar{p}_{i,1} + \bar{p}_{i,2} + \max(\hat{p}_{i',1}, \hat{p}_{i,1}, \hat{p}_{i,2})$ *si* $\gamma \geq 1$, 0 *sinon*

$c_{10} = C^{\gamma-2}_{l-1,1} + \bar{p}_{i',1} + \bar{p}_{i,1} + \bar{p}_{i,2} + \max(\hat{p}_{i',1} + \hat{p}_{i,1}, \hat{p}_{i',1} + \hat{p}_{i,2}, \hat{p}_{i,1} + \hat{p}_{i,2})$ *si* $\gamma \geq 2$, 0 *sinon*

$c_{11} = C^{\gamma-3}_{l-1,1} + \bar{p}_{i',1} + \hat{p}_{i',1} + \bar{p}_{i,1} + \hat{p}_{i,1} + \bar{p}_{i,2} + \hat{p}_{i,2}$ *si* $\gamma \geq 3$, 0 *sinon.*

Il est clair que $b_1 = c_1$*,* $b_2 = c_2$ *et* $b_3 = c_3$*.*

Si la condition (i) est vérifiée, alors $\bar{p}_{i,1} > \bar{p}_{i,2}$*, ce qui implique* $c_4 \leq b_8$*,* $c_5 \leq b_9$*,* $c_6 \leq b_{10}$ *et* $c_7 \leq b_{11}$*. De plus,* $\bar{p}_{i',1} < \bar{p}_{i',2}$*, ce qui entraîne* $c_8 \leq b_4$*,* $c_9 \leq b_5$*,* $c_{10} \leq b_6$ *et* $c_{11} \leq b_7$*.*
Sous la condition (ii), $\bar{p}_{i',1} < \bar{p}_{i,1}$*, ainsi* $c_4 \leq b_4$*,* $c_5 \leq b_5$*,* $c_6 \leq b_6$ *et* $c_7 \leq b_7$*. D'autre part,* $\bar{p}_{i',1} < \bar{p}_{i',2}$ *implique* $c_8 \leq b_4$*,* $c_9 \leq b_5$*,* $c_{10} \leq b_6$ *et* $c_{11} \leq b_7$*.*
Enfin, la condition (iii) nous donne $\bar{p}_{i,1} > \bar{p}_{i,2}$ *entraînant* $c_4 \leq b_8$*,* $c_5 \leq b_9$*,* $c_6 \leq b_{10}$ *et* $c_7 \leq b_{11}$*, ainsi que* $\bar{p}_{i',2} > \bar{p}_{i,2}$ *et donc* $c_8 \leq b_8$*,* $c_9 \leq b_9$*,* $c_{10} \leq b_{10}$ *et* $c_{11} \leq b_{11}$*.*

Ainsi, sous chacune des trois conditions (i), (ii) et (iii), le makespan de l'ordonnancement considérant la séquence σ' *n'est pas plus grand que celui considérant* σ*, quel que soit* γ*.* □

Notons que cette preuve ne concerne que la position relative des deux travaux i et i'. Bien qu'elle soit basée sur le cas général où ces travaux sont entourés par d'autres travaux dans la séquence, nous pouvons appliquer le même raisonnement :

- **si les travaux** i **et** i' **sont les deux premiers travaux de la séquence**. Dans ce cas, la date de disponibilité des machines pour débuter le traitement des ces travaux est zéro ($C^{\gamma}_{l-1,m}(\sigma) = C^{\gamma}_{l-1,m}(\sigma') = 0 \quad \forall \gamma \leq \Gamma, \forall m \in \{M_1, M_2\}$, et le travail $\sigma[l-1]$ peut être considérée comme une tâche virtuelle de durée nulle débutant au temps zéro) ;
- **si les travaux** i **et** i' **sont les deux derniers de la séquence**. Dans ce cas, $C^{\gamma}_{l+1,2}(\sigma)$ pour la séquence σ et $C^{\gamma}_{l+1,2}(\sigma')$ pour σ' représentent le makespan des solutions respectives.

Par conséquent, il est possible de trouver la séquence optimale pour le problème de flow-shop robuste à deux machines, avec un budget d'incertitude global, et un ordre des durées de traitement conservé par les déviations, en temps polynomial, en utilisant la règle de Johnson.

7.5.2 Budget dépendant des machines

La règle de Johnson ne permet pas toujours d'obtenir une séquence optimale lorsqu'on considère un budget d'incertitude dépendant des machines, comme nous le montre l'exemple 14.

Exemple 14 *Considérons un problème de flow-shop à deux machines et trois travaux avec un budget d'incertitude dépendant des machines* $\Gamma = (1, 2)$*. L'intervalle* $[\bar{p}_{i,m}, \bar{p}_{i,m} + \hat{p}_{i,m}]$ *de durée de traitement de chaque opération* $O_{i,m}$*,* $i \in \mathcal{J}, m \in \mathcal{M}$ *est donné par le tableau Table 7.1.*

Si on applique la règle de Johnson à cette instance, on obtient alors la séquence $\sigma = \{J_1, J_2, J_3\}$*. Étant donné la séquence* σ *et considérant le budget d'incertitude* $\Gamma = (1, 2)$*, le scénario pire cas est que le temps de traitement du travail* J_2 *sur la machine* M_1*, ainsi que des travaux* J_2 *et* J_3 *sur la machine* M_2 *dévient et prennent leurs plus grandes valeurs. La figure 7.1 représente le diagramme de Gantt dans ce cas. Le makespan de cette solution atteint alors 32.*

	M_1	M_2
J_1	[6,9]	[8,12]
J_2	[10,15]	[4,6]
J_3	[4,6]	[3,5]

TABLEAU 7.1 – Exemple d'une instance de flow-shop à deux machines avec ordre des temps de traitement préservé

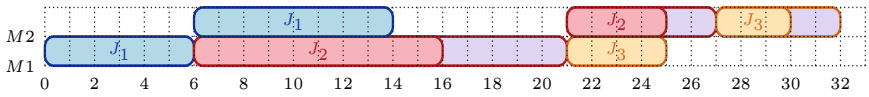


FIGURE 7.1 – Exemple 14 avec la séquence $\{J_1, J_2, J_3\}$; scénario pire cas sous le budget d'incertitude $\Gamma = (1, 2)$

Une autre solution consiste à considérer la séquence $\sigma' = \{J_3, J_1, J_2\}$. *Le scénario pire cas pour cette solution est que les durées de traitement du travail* J_2 *sur la machine* M_1 *et des travaux* J_1 *et* J_2 *sur la machine* M_2 *prennent leurs valeurs maximales. La figure 7.2 décrit le diagramme de Gantt dans cette situation. Cette solution mène à un makespan pire cas de 31. Cette séquence est donc plus robuste face aux incertitudes comprises dans le budget considéré.*

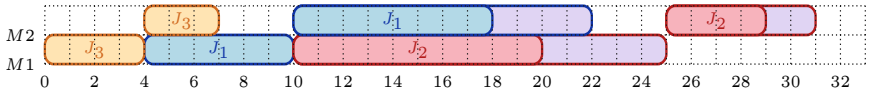


FIGURE 7.2 – Exemple 14 avec la séquence $\{J_3, J_1, J_2\}$; scénario pire cas sous le budget d'incertitude $\Gamma = (1, 2)$

Ainsi, bien que dans cet exemple l'ordre des durées de traitement soit préservé, la règle de Johnson ne permet pas de trouver la séquence optimale pour un budget d'incertitude dépendant des machines $\Gamma = (1, 2)$.

7.5.3 Ordre des temps de traitement non préservé

La règle de Johnson ne permet pas toujours d'obtenir une séquence optimale quand l'ordre des durées de traitement n'est pas le même, avec et sans déviations.

Exemple 15 *Considérons un problème de flow-shop à deux machines avec trois travaux et un budget d'incertitude global* $\Gamma = 2$. *L'intervalle* $[\bar{p}_{i,m}, \bar{p}_{i,m} + \hat{p}_{i,m}]$ *de durée de traitement de chaque opération* $O_{i,m}$, $i \in \mathcal{J}, m \in \mathcal{M}$ *est donné par le tableau 7.2. L'ordre des durées de traitement n'est pas préservé, par exemple,* $\bar{p}_{1,1} = 1 < \bar{p}_{2,1} = 2$ *alors que* $\bar{p}_{1,1} + \hat{p}_{1,1} = 5 > \bar{p}_{2,1} + \hat{p}_{2,1} = 3$.

	M_1	M_2
J_1	[1,5]	[2,3]
J_2	[2,3]	[1,5]
J_3	[2,20]	[4,5]

TABLEAU 7.2 – Exemple d'une instance de flow-shop à deux machines avec ordre des temps de traitement non préservé

Si on applique la règle de Johnson à cette instance, on obtient alors la séquence $\sigma = \{J_1, J_3, J_2\}$. *Étant donné la séquence* σ *et considérant le budget d'incertitude* $\Gamma = 2$, *le scénario pire cas est que le temps de traitement du travail* J_3 *sur la machine* M_1 *ainsi que le travail* J_2 *sur la machine* M_2 *dévient et prennent leurs plus grandes valeurs. La figure 7.3 représente le diagramme de Gantt dans ce cas. Le makespan de cette solution atteint alors 30.*

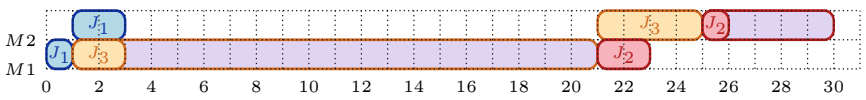


FIGURE 7.3 – Exemple 15 avec la séquence $\{J_1, J_3, J_2\}$; scénario pire cas sous le budget d'incertitude $\Gamma = 2$

Une autre solution consiste à considérer la séquence $\sigma' = \{J_2, J_3, J_1\}$. *Le scénario pire cas, pour cette solution est que les durées de traitement du travail* J_3 *sur le machine* M_1 *et du travail* J_1 *sur le machine* M_2 *prennent leurs valeurs maximales. La figure 7.4 décrit le diagramme de Gantt dans cette situation. Cette solution mène à un makespan pire cas de 29. Cette solution est donc plus robuste face aux incertitudes comprises dans le budget considéré.*

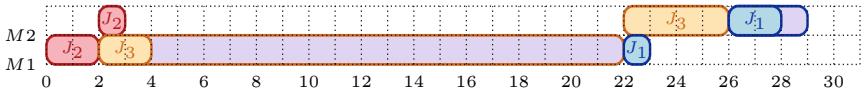


FIGURE 7.4 – Exemple 15 avec la séquence $\{J_2, J_3, J_1\}$; scénario pire cas sous le budget d'incertitude $\Gamma = 2$

Ainsi, bien que dans cet exemple le budget d'incertitude considéré soit global, la règle de Johnson ne permet pas de trouver la séquence optimale pour cette instance dont l'ordre des durées de traitement n'est pas préservé.

En résumé, nous avons montré que lorsque l'on considère un budget d'incertitude global et que les temps de traitement suivent le même ordre, avec ou sans déviation, le problème de flow-shop robuste à deux machines peut être résolu de

manière optimale en utilisant la règle de Johnson. Cependant, dans le cas général où le budget d'incertitude dépend des machines et/ou lorsque l'ordre des temps de traitement n'est pas préservé, la complexité du problème reste une question ouverte. Néanmoins, les méthodes de résolution présentées dans les paragraphes 7.3 et 7.4 peuvent être appliquées pour le problème à deux machines.

7.6 Expérimentations numériques

Dans ce paragraphe, nous étudions les performances des différentes méthodes présentées dans le chapitre pour résoudre le problème de flow-shop de permutation robuste, dans les conditions expérimentales exposées dans le paragraphe 4.4.

7.6.1 Résultats pour le cas général

7.6.1.1 Instance de petite et moyenne taille

Dans un premier temps, nous étudions les performances des différentes méthodes sur des instances de petite et moyenne taille. Les instances de petite taille ont été générées avec un nombre de travaux variant entre 4 et 6, tandis que pour les instances de taille moyenne, le nombre de travaux varie entre 7 et 9. Le nombre de machines varie quant à lui entre 4 et 6 pour l'ensemble des instances. Les durées nominales $\bar{p}_{i,m}$ et les déviations maximales $\hat{p}_{i,m}$ de chaque opération $O_{i,m}$ sont des entiers générés aléatoirement à partir d'une distribution uniforme $\texttt{Unif}(1, 20)$. Pour chaque combinaison de paramètres, cinq instances ont été générées, ce qui donne un total de 90 instances. Nous testons tous les modèles en faisant varier le budget d'incertitude selon quatre ratios : 5 %, 10 %, 15 % et 20 %, soit un total de 360 expériences par méthode.

Le tableau 7.3 présente le nombre de solutions optimales trouvées sur un total de 20 expériences, ainsi que le temps d'exécution moyen pour chaque méthode et pour chaque taille.

On remarque que l'ensemble de ces instances ont été résolues de manière optimale par la méthode *Compact*. La méthode *Benders* permet de résoudre près de la totalité des petites instances, mais ne résout qu'un nombre très restreint, 7, d'instances de taille moyenne. Les méthodes de génération de colonnes et de contraintes parviennent à résoudre la majorité des instances. On note tout de même que la méthode CCG_{PLNE} obtient de meilleurs résultats, en résolvant un plus grand nombre d'instances que la méthode CCG_{PPC}. On remarque finalement que, pour l'ensemble des méthodes, la difficulté des instances augmente avec leur taille. Cela se traduit par une augmentation des temps de résolution pour l'ensemble des méthodes et une diminution du nombre d'instances résolues à l'optimum pour toutes les méthodes sauf *Compact*.

Le tableau 7.4 présente les résultats pour différentes valeurs de budget d'incertitude. On remarque que pour l'ensemble des méthodes, plus le budget est élevé, plus il devient difficile de trouver des solutions optimales. Pour la méthode *Compact*,

$\|\mathcal{J}\|$	$\|\mathcal{M}\|$	*Compact*		*Benders*		CCG_{PLNE}		CCG_{PPC}	
		#opti.	t (s)	#opti.	t (s)	#opti.	t (s)	#opti.	t (s)
4	4	20	0.08	20	0.25	20	0.09	20	0.04
4	5	20	0.13	20	0.4	20	0.22	20	0.05
4	6	20	0.12	20	0.5	20	0.18	20	0.22
5	4	20	0.15	20	3.78	20	0.29	20	0.86
5	5	20	0.15	20	5.12	20	0.43	20	1.62
5	6	20	0.27	20	5.18	20	1.05	20	19.51
6	4	20	0.22	19	109.47	20	0.66	20	7.19
6	5	20	0.28	20	128.61	20	1.08	20	14.33
6	6	20	0.42	20	133.56	20	3.84	20	112.82
7	4	20	0.42	4	1563.3	20	4.34	20	61.18
7	5	20	0.56	3	389.3	20	7.36	20	281.54
7	6	20	1.12	0	-	20	37.59	13	190.1
8	4	20	1.25	0	-	20	18.97	19	301.07
8	5	20	1.83	0	-	19	35.91	15	153.39
8	6	20	3.99	0	-	19	430.08	11	825.68
9	4	20	7.73	0	-	19	278.26	17	116.97
9	5	20	19.29	0	-	19	190.33	15	370.26
9	6	20	20.85	0	-	10	341.76	7	286.4

Tableau 7.3 – Nombre de solutions optimales et temps de résolution moyen pour chaque méthode étudiée, selon la taille des instances, pour les instances de flow-shop de permutation de petite et moyenne taille

on observe une augmentation du temps d'exécution à mesure que le budget d'incertitude augmente. Cela peut s'expliquer par le fait que le nombre de variables considérées dans ce modèle dépend directement du nombre de déviations considéré. Quant aux autres méthodes, *Benders*, CCG_{PLNE} et CCG_{PPC}, on constate que le nombre d'instances résolues à l'optimum diminue à mesure que le budget d'incertitude augmente. Cela peut s'expliquer par le fait que pour les budgets considérés, un budget d'incertitude plus élevé entraîne une augmentation du nombre de scénarios possibles.

7.6.1.2 Instances générées à partir de la littérature

On considère à présent des instances plus grandes qui sont dérivées des instances classiques de flow-shop de permutation déterministe présenté dans le paragraphe 1.4.2. Plus précisément, nous considérons les instances ayant un nombre de travaux appartenant à l'ensemble $\{20, 50, 100\}$ et un nombre de machines appartenant à l'ensemble $\{5, 10, 20\}$. Chaque ensemble est composé de 10 éléments et nous prenons en compte toutes les combinaisons possibles de ces valeurs, ce qui nous donne un total de 90 instances. Ces instances sont adaptées pour tenir compte du contexte robuste en introduisant aléatoirement des valeurs de déviation pour les durées des opérations. Pour chaque opération $O_{i,m}$, la durée de traitement fournie

Budget	*Compact*		*Benders*		CCG_{PLNE}		CCG_{PPC}	
	#opti.	t (s)	#opti.	t (s)	#opti.	t (s)	#opti.	t (s)
5%	90	0.409	51	88.9	90	0.908	90	5.62
10%	90	2.12	46	128	89	27.5	85	93.1
15%	90	5.1	45	53	86	92.6	75	203
20%	90	5.45	44	51.2	81	151	67	231
total	360	3.3	186	80.9 .8	346	65.6	317	123.4

TABLEAU 7.4 – Nombre de solutions optimales et temps de résolution moyen pour chaque méthode étudiée, selon la valeur de budget d'incertitude, pour les instances de flow-shop de permutation de petite et moyenne taille ; #instances = 90

par l'instance est considérée comme la valeur nominale $\bar{p}_{i,m}$. Ensuite, nous générons aléatoirement une valeur de déviation $\hat{p}_{i,m}$ dans un intervalle de 25% à 100% de la durée nominale maximale $\bar{p}_{max} = \max\{\bar{p}_{i,m} | i \in \mathcal{J}, m \in \mathcal{M}$.

Le tableau 7.5 présente le nombre de meilleures solutions réalisables trouvées et l'écart d'optimalité moyen pour chaque méthode et chaque taille d'instance.

$\|\mathcal{J}\|$	$\|\mathcal{M}\|$	*Compact*		*Benders*		CCG_{PLNE}		CCG_{PPC}	
		#Best	Gap (%)	#Best	Gap (%)	#Best	Gap (%)	#Best	Gap (%)
20	5	13	6.49	19	42.65	8	12.2	15	8.74
20	10	20	10.4	6	50.92	5	49.38	12	35.9
20	20	0	37.49	7	56.22	12	56.13	25	57.63
50	5	0	23.03	31	43.67	11	10.15	12	7.88
50	10	0	-	11	53.33	15	53.33	22	45.17
50	20	0	-	11	58.13	9	58.22	32	57.5
100	5	0	-	25	44.83	8	26.8	10	8.93
100	10	0	-	10	52.83	12	52.83	25	52.4
100	20	0	-	0	59.58	0	59.6	40	56.65

TABLEAU 7.5 – Nombre de meilleures solutions réalisables trouvées et écart d'optimalité moyen par méthode et taille d'instance, pour les instances de flow-shop de permutation de la littérature

On remarque que les performances de la méthode *Compact* se détériorent rapidement à mesure que la taille des instances augmente. Bien qu'elle soit performante pour trouver et prouver l'optimum pour les instances de plus petite taille étudiées dans le paragraphe 7.6.1.1, cette méthode ne parvient pas à trouver de solution réalisable à partir de 50 travaux et 10 machines. Il n'y a pas de dominance claire entre les méthodes *Benders*, CCG_{PLNE} et CCG_{PPC}. La méthode *Benders* présente une variation limitée de l'écart d'optimalité en fonction de la taille des instances, mais celui-ci est élevé. Les méthodes basés sur la génération de colonnes et de contraintes CCG_{PLNE} et CCG_{PPC} sont plus sensibles à la variation du nombre de machines dans les instances. En effet, pour ces méthodes, on observe des écarts d'optimalité modérés pour les instances à 5 machines mais des écarts plus élevés pour les instances considérant un plus grand nombre de machines. Pour les plus grandes

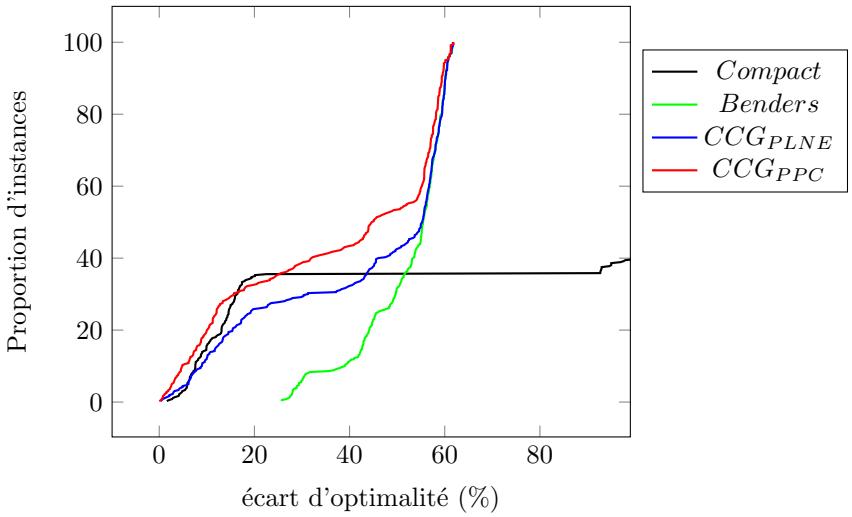


FIGURE 7.5 – Profil d'écart d'optimalité pour les instances de la littérature

instances, c'est la méthode $CCCG_{PPC}$ qui obtient les meilleures solutions.

La figure 7.5 met en évidence la proportion d'instances résolues avec un écart d'optimalité inférieur à une valeur donnée. Les observations concernant la méthode *Compact* vont dans le même sens que celles faites lors de l'étude du tableau précédent. Les performances de cette méthode se dégradent rapidement et, pour une grande majorité des instances, il n'est pas possible d'obtenir une solution réalisable dans les conditions expérimentales fixées. On remarque aussi une dominance des méthodes de génération de colonnes et de contraintes, en particulier de la méthode CCG_{PPC} pour environ 50% des instances, pour le reste des instances, les performances en termes d'écart d'optimalité de l'ensemble des méthodes de décomposition sont similaires.

7.6.2 Résultats pour le cas du problème à deux machines

Dans ce paragraphe nous nous intéressons aux performances des méthodes de génération de colonnes et de contraintes pour les problèmes de flow-shop de permutation à deux machines. Notre objectif est de comparer ces méthodes lorsque la PLNE est utilisée pour résoudre le problème maître, CCG_{PLNE}, comme dans [Levorato 2022], et lorsque le problème maître est remplacé par un problème de PPC, CCG_{PPC}.

7.6.2.1 Instances de la littérature

Dans un premier temps, nous évaluons les performances des méthodes sur les instances utilisées dans [Levorato 2022], générées dans [Ying 2015]. Elles sont composées de six groupes d'instances de taille différente, où le nombre de travaux $|\mathcal{J}|$ varie

dans l'ensemble $\{10, 20, 50, 100, 150, 200\}$. Les temps de traitement nominaux $\bar{p}_{i,m}$, $i \in \mathcal{J}, m \in \{M_1, M_2\}$ sont générés à partir de la distribution uniforme $\texttt{Unif}(10, 50)$ et les déviations $\hat{p}_{i,m}$ correspondent à un ratio du temps de traitement nominal $\alpha\bar{p}_{i,m}$ avec $\alpha = 10, 20, 30, 40$ et $50\,\%$. Ainsi, l'ordre des durées de traitement est préservé par la déviation.

Dix valeurs de durée nominale ont été générées pour chaque taille $|\mathcal{J}|$, et tous les ratios de déviation α ont été appliqués à chacune d'entre elles, ce qui donne un total de 300 instances. Les budgets d'incertitude, Γ_1 et Γ_2, sont fixés à 20%, 40%, 60%, 80% et 100% de $|\mathcal{J}|$.

Nous synthétisons les résultats dans le tableau 7.6 qui présente les performances pour les instances regroupées selon leur taille.

Nous indiquons le pourcentage d'instances résolues de façon optimale (*Opti. (%)*). Pour les instances résolues de manière optimale, nous affichons le temps d'exécution moyen, en secondes, pour atteindre l'optimalité (*t (s)*). Enfin, nous indiquons le pourcentage moyen d'écart d'optimalité pour les instances résolues de manière non optimale (*Gap (%)*).

$\lvert\mathcal{J}\rvert$	CCG_{PPC}			CCG_{PLNE}		
	Opti. (%)	t (s)	Gap (%)	Opti. (%)	t (s)	Gap (%)
10	100	0.42	–	100	14.7	–
20	100	0.76	–	100	191.59	–
50	100	1.47	–	98.75	194.9	0.23
100	99.58	0.99	0.08	85.5	319.17	0.31
150	98.92	3.12	0.02	74.67	341.65	0.47
200	98.83	8.49	0.1	68.92	390.73	0.4

TABLEAU 7.6 – Performances des méthodes de génération de colonnes et de contraintes sur les instances de flow-shop de permutation robuste à deux machines de la littérature

On remarque que la méthode CCG_{PPC} est plus performante que la méthode CCG_{PLNE}. En effet, même pour les plus grandes instances (200 travaux), la méthode basée sur le modèle de PPC parvient à résoudre presque toutes (98,83%) les instances de manière optimale, alors que la méthode basée sur le modèle MILP résout de moins en moins d'instances au fur et à mesure que leur taille augmente (pour atteindre 68,92% pour 200 travaux). Nous observons également que le temps nécessaire pour atteindre l'optimum est beaucoup plus faible pour la méthode CCG_{PPC}, et ce quelle que soit la taille des instances. Enfin, on constate que pour les deux méthodes, l'écart d'optimalité est faible (moins de 0,5%, quelle que soit la taille des instances).

Il convient de noter que les quelques différences entre les résultats rapportés ici et ceux présentés dans [Levorato 2022] concernant la méthode CCG_{PLNE} sont probablement dues à une différence dans l'implémentation de la méthode et les outils (logiciels et matériels) utilisés pour les tests. Cependant, nous obtenons des résultats très comparables.

Nous examinons maintenant la qualité des solutions obtenues en appliquant la règle de Johnson à ces instances. Le tableau 7.7 présente le pourcentage de meilleures solutions connues trouvées (*Best (%)*) et le pourcentage moyen d'écart d'optimalité (*Gap (%)*), par rapport à la meilleure borne inférieure connue BI^*, pour les instances résolues de manière non optimale ($BS^{méthode} > BI^*$).

$\|\mathcal{J}\|$	Best(%)	Gap (%)
10	92.58	1.13
20	92.67	0.4
50	98.5	0.29
100	97.67	0.04
150	89.67	0.04
200	95.67	0.04

TABLEAU 7.7 – Performances de la règle de Johnson sur les instances de flow-shop de permutation robuste à deux machines de la littérature

Nous constatons que l'algorithme permet de trouver des solutions de haute qualité, très rapidement. En effet, pour presque toutes les instances (94,46%), la règle de Johnson fournit une solution robuste avec la même valeur de la fonction objectif que la meilleure solution connue, et l'écart d'optimalité est très faible.

Au regard de ces résultats, nous pouvons considérer que ces instances sont faciles à résoudre, en raison de leur structure particulière qui préserve l'ordre des temps de traitement. Nous avons donc généré de nouvelles instances supposées être plus difficiles ; les résultats obtenus sont présentés dans le paragraphe suivant.

7.6.2.2 Nouvelles instances

Dans ce paragraphe, nous avons généré de nouvelles instances, également basées sur celles de [Ying 2015]. Les temps de traitement nominaux $\bar{p}_{i,m}, i \in \mathcal{J}, m \in \{M_1, M_2\}$ restent les mêmes que dans les instances originales. Cependant, les déviations $\hat{p}_{i,m}$ sont générées aléatoirement pour éviter la préservation de l'ordre des temps de traitement. Soit $\bar{p}_{max}$ le temps de traitement nominal maximum pour toutes les opérations. Pour chaque opération $O_{i,m}, i \in \mathcal{J}, m \in \{M_1, M_2\}$, nous générons aléatoirement une valeur pour $\hat{p}_{i,m}$ allant de 25% à 80% de la valeur de $\bar{p}_{max}$. Nous générons au total 60 instances que nous utilisons pour nos tests. Une fois encore, les budgets d'incertitude, Γ_1 et Γ_2, sont fixés à 20%, 40%, 60%, 80% et 100% du nombre total de travaux $|\mathcal{J}|$.

Le tableau 7.8 présente les mêmes indicateurs de performance que le tableau 7.6 pour les nouvelles instances générées.

En comparant ces deux tableaux, on constate que les nouvelles instances sont plus difficiles à résoudre que celles de la littérature. En effet, la proportion d'instances résolues de manière optimale est plus faible, le temps moyen nécessaire pour atteindre l'optimum est plus élevé, de même que l'écart moyen pour les instances non résolues, et ce pour les deux méthodes. Cependant, en se concentrant sur les

$\|\mathcal{J}\|$	CCG_{PPC}			CCG_{PLNE}		
	#opti.	t (s)	Gap (%)	#opti.	t (s)	Gap (%)
10	100	12.36	–	100	46.77	–
20	80	167.11	2.04	75	1002.05	2.22
50	60	86.85	2.61	49	902.69	3.07
100	56	310.3	2.49	45	600.67	4.11
150	48	271.32	2.3	32	667.89	4.3
200	54	99.57	2.68	30	827.08	4.36

TABLEAU 7.8 – Performances des méthodes de génération de colonnes et de contraintes sur les nouvelles instances de flow-shop de permutation robuste à deux machines

informations fournies par le tableau 7.8, on remarque que la méthode CCG_{PPC} reste plus performant que la méthode CCG_{PLNE}, pour l'ensemble des indicateurs observés.

Les tableaux 7.9 et 7.10 détaillent le pourcentage d'instances résolues en fonction du budget d'incertitude $\Gamma = (\Gamma_1, \Gamma_2)$ pour les méthodes CCG_{PPC} et CCG_{PLNE}, respectivement.

Γ_2 \ Γ_1	20 %	40 %	60 %	80 %	100 %
20 %	28.33	25	41.67	70	100
40 %	35	21.67	26.67	81.67	100
60 %	36.67	28.33	28.33	83.33	100
80 %	60	71.67	75	88.33	100
100 %	100	100	100	100	–

TABLEAU 7.9 – Pourcentage d'instances résolues à l'optimum en fonction du budget d'incertitude $\Gamma = (\Gamma_1, \Gamma_2)$ pour la méthode CCG_{PPC}

Γ_2 \ Γ_1	20 %	40 %	60 %	80 %	100 %
20 %	26.67	26.67	33.33	58.33	100
40 %	23.33	18.33	23.33	65	100
60 %	23.33	21.67	26.67	65	98.33
80 %	35	35	40	60	100
100 %	76.67	83.33	91.67	98.33	–

TABLEAU 7.10 – Pourcentage d'instances résolues à l'optimum en fonction du budget d'incertitude $\Gamma = (\Gamma_1, \Gamma_2)$ pour la méthode CCG_{PLNE}

En comparant ces deux tableaux entre eux, nous remarquons que, pour toutes les combinaisons de Γ_1 et Γ_2, sauf une ($\Gamma_1 = 40\%$, $\Gamma_2 = 20\%$), la méthode CCG_{PPC} est plus performante que la méthode CCG_{PLNE}. Nous constatons également que

le problème est plus difficile à résoudre pour les budgets d'incertitude moyens (40% ou 60%), pour les deux méthodes. Cela peut s'expliquer par le fait que ces budgets d'incertitude génèrent un plus grand nombre de scénarios. Cependant, le nombre de scénarios n'est pas le seul facteur de difficulté. En effet, comme nous pouvons le voir, les méthodes sont plus efficaces pour résoudre les instances avec un grand budget d'incertitude. Par exemple, les instances avec un budget d'incertitude de 80% sont mieux résolues que celles avec un budget de 20%, alors que le nombre de scénarios est le même.

7.7 Conclusion

Dans ce chapitre, nous avons appliqué les formulations présentées dans le chapitre précédent au problème du flow-shop de permutation robuste. Nous nous sommes particulièrement intéressés à la complexité du problème à deux machines et nous avons montré que sous certaines conditions, le problème peut être résolu de façon optimale avec un algorithme polynomial. Pour les instances considérées dans la littérature, cet algorithme reste très efficace.

Dans le cas général, pour les instances de petite taille, une formulation directe du problème permet d'obtenir des solutions optimales du problème très rapidement. Cependant, pour des instances plus grandes, les méthodes de décomposition permettent de trouver des solutions de meilleure qualité, en particulier la méthode de génération de colonnes et de contraintes avec un problème maître résolu à l'aide de la PPC.

Conclusions et perspectives

Conclusions

La principale contribution de cette thèse est de proposer des méthodes de résolution exactes et hybrides pour des problèmes d'ordonnancement disjonctif.

Dans la partie II, nous nous sommes intéressés au problème de job-shop flexible, dans sa version non préemptive et préemptive, et avons proposé un algorithme de décomposition de Benders basé sur la logique pour le résoudre. Cette approche décompose le problème en un problème maître d'affectation des opérations aux machines et un sous-problème d'ordonnancement. Pour renforcer cette méthode, nous avons exploré, dans le chapitre 3, différentes relaxations du problème et étudié la génération de coupes d'optimalité. De plus, nous avons ajouté une étape supplémentaire, permettant de rapidement évaluer la qualité des affectations fixées par le problème maître, qui s'est avérée efficace. Pour chaque version du problème, non préemptive (chapitre 4) et préemptive (chapitre 5), nous avons présenté différentes méthodes de résolution du sous-problème. Nous avons ensuite comparé cet algorithme de décomposition avec des méthodes de résolution directes qui considèrent simultanément l'affectation des machines et la séquence des opérations. Pour la version non préemptive, la résolution directe à l'aide de la programmation par contraintes s'est avérée être plus efficace que la méthode que nous proposons, dans la grande majorité des cas. Cependant, notre algorithme de décomposition a permis d'améliorer certaines bornes par rapport à l'état de l'art. Au contraire, pour la version préemptive du problème, l'utilisation d'un algorithme de *branch-and-bound* de la littérature pour la résolution du sous-problème combiné à un problème maître qui décide de l'affectation a conduit à de meilleures performances par rapport à la résolution directe du problème.

Dans la partie III, nous nous sommes intéressés aux problèmes d'ordonnancement robuste, dans lesquels les durées de traitement des opérations sont incertaines et modélisées par un budget d'incertitude. Nous avons considéré un processus de décision à deux niveaux, où les séquences d'opérations doivent être décidées a priori, c'est-à-dire avant de connaître la réalisation de l'incertitude, de manière à être réalisables pour l'ensemble des scénarios, mais où les dates de traitement des opérations peuvent être adaptées en fonction des durées réelles observées pour chaque opération. Dans le chapitre 6, nous avons présenté plusieurs méthodes pour le problème du job-shop robuste. Tout d'abord, des formulations directes ont été introduites. Plus précisément, des formulations étendues, en programmation linéaire en nombres entiers et en programmation par contraintes, dans lesquelles certaines variables et contraintes sont dupliquées pour tous les scénarios de l'ensemble d'incertitude considéré. En outre, nous avons proposé une formulation compacte qui permet de réduire le nombre de contraintes à un nombre polynomial par rapport à la taille du problème. Ensuite, nous avons développé deux méthodes de décomposition qui reposent

sur la résolution d'un problème maître relâché et la recherche de contraintes violées à chaque itération. La première méthode utilise la génération de coupes de Benders pour ajouter les contraintes violées, tandis que la deuxième méthode consiste en la génération de variables et de contraintes relatives aux scénarios qui entraînent la violation de ces contraintes. Nous avons également proposé une extension de ces méthodes pour la version flexible du problème de job-shop dans ce même chapitre 6, ainsi que pour le problème de flow-shop de permutation robuste dans le chapitre 7. Dans le cas général, pour les instances de petite taille, une formulation directe du problème permet d'obtenir rapidement des solutions optimales. Cependant, pour des instances plus grandes, les méthodes de décomposition, en particulier la méthode de génération de colonnes et de contraintes avec un problème maître résolu à l'aide de la programmation par contraintes, permettent d'obtenir des solutions de meilleure qualité. Dans le chapitre 7, nous nous sommes aussi penchés sur la complexité du problème de flow-shop robuste à deux machines et nous avons montré que, sous certaines conditions, le problème peut être résolu de façon optimale avec un algorithme polynomial.

Perspectives

Programmation par contraintes pour les problèmes d'ordonnancement préemptif

Dans le chapitre 5, nous avons proposé une modélisation PPC des opérations morcelables à l'aide de sous-opérations de durée unitaire. Or, cela induit l'utilisation de nombreuses variables, ce qui rend les modèles peu performants, comme on le constate dans les résultats présentés dans le paragraphe 5.4. Cependant, pour les problèmes de job-shop étudiés, il n'est pas nécessaire de représenter l'ensemble des interruptions dans le traitement des tâches [Ebadi 2012]. Dans une contribution récemment acceptée [Juvin 2023a], nous proposons une approche de programmation par contraintes sans fragmentation explicite des opérations pour la résolution du problème de job-shop préemptif. Notre étude expérimentale montre qu'elle améliore sensiblement l'état de l'art pour ce problème. Une perspective consiste maintenant à adapter et évaluer cette approche pour le problème de job-shop flexible. De plus, nous envisageons d'intégrer cette méthode de résolution pour le sous-problème dans notre schéma de décomposition de Benders basée sur la logique.

Amélioration des coupes de Benders logiques

De manière générale, l'efficacité des méthodes de décomposition de Benders basées sur la logique repose en grande partie sur la génération de coupes globales, capables d'exclure un grand nombre de solutions infaisables ou non optimales du problème maître. Malheureusement, nous ne sommes pas parvenus à générer de telles coupes de manière satisfaisante. Cependant, pour les problèmes d'ordonnancement robuste présentés dans la partie III, nous faisons une observation prometteuse. Plutôt que de considérer toutes les positions relatives des opérations dans le chemin critique,

nous avons constaté que pour un groupe d'opérations successives traitées par une même machine, seules les positions de la première et de la dernière opération ont une incidence sur la valeur du makespan. Dans la figure 7.6, si l'on considère que les opérations représentées sont des opérations consécutives d'un chemin critique, changer la position relative des opérations représentées en noir ne peut pas entraîner une diminution de la valeur du makespan.

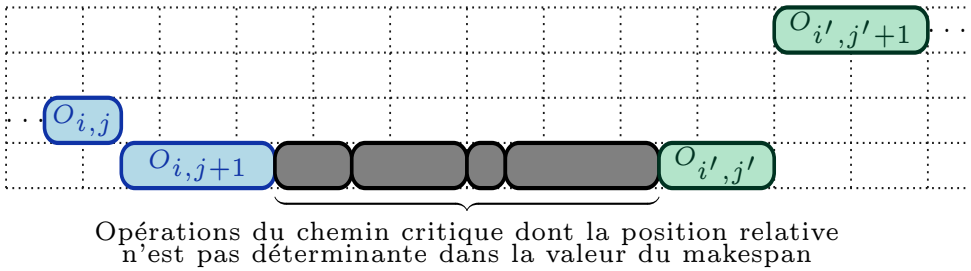


FIGURE 7.6 – Illustration d'un chemin critique avec opérations pour lesquelles un changement de position relative ne permet pas de réduire le makespan

Par conséquent, il serait possible de générer des coupes plus globales en considérant toutes les permutations des autres opérations au sein de ce groupe. Une perspective consiste donc à approfondir cette idée et de mener des études supplémentaires pour évaluer son impact sur la qualité des solutions obtenues et sur les performances de la méthode.

Bibliographie

[Adams 1988] Joseph Adams, Egon Balas et Daniel Zawack. *The shifting bottleneck procedure for job shop scheduling.* Management Science, vol. 34, no. 3, pages 391–401, 1988. (Cité en page 19.)

[Applegate 1991] David Applegate et William Cook. *A computational study of the job-shop scheduling problem.* ORSA Journal on Computing, vol. 3, no. 2, pages 149–156, 1991. (Cité en page 19.)

[Balas 1969] Egon Balas. *Machine sequencing via disjunctive graphs: an implicit enumeration algorithm.* Operations Research, vol. 17, no. 6, pages 941–957, 1969. (Cité en page 13.)

[Balouka 2021] Noemie Balouka et Izack Cohen. *A robust optimization approach for the multi-mode resource-constrained project scheduling problem.* European Journal of Operational Research, vol. 291, no. 2, pages 457–470, 2021. (Cité en page 32.)

[Baptiste 1995] Philippe Baptiste, Claude Le Pape et Wim Nuijten. *Constraint-based optimization and approximation for job-shop scheduling.* In Proceedings of the AAAI-SIGMAN workshop on Intelligent Manufacturing Systems, pages 5–16. Citeseer, 1995. (Cité en page 15.)

[Barnes 1995] J. Wesley Barnes et John B. Chambers. *Solving the job shop scheduling problem with tabu search.* IIE transactions, vol. 27, no. 2, pages 257–263, 1995. (Cité en page 18.)

[Barnes 1996] J. Wesley Barnes et John B. Chambers. *Flexible job shop scheduling by tabu search.* Rapport technique ORP96-09, The University of Texas at Austin, 1996. (Cité en pages 20 et 21.)

[Beasley 1990] John E. Beasley. *OR-Library: distributing test problems by electronic mail.* Journal of the Operational Research Society, vol. 41, no. 11, pages 1069–1072, 1990. (Cité en page 20.)

[Beck 2009] Amir Beck et Aharon Ben-Tal. *Duality in robust optimization: primal worst equals dual best.* Operations Research Letters, vol. 37, no. 1, pages 1–6, 2009. (Cité en page 28.)

[Behnke 2018] Dennis Behnke et Martin Josef Geiger. *Test instances for the flexible job shop scheduling problem with work centers.* Rapport technique RR-12-01-01, Institut für betriebliche Logistik und Organisation, Janvier 2018. ISSN 2192-0826. (Cité en page 20.)

[Ben-Tal 1999] Aharon Ben-Tal et Arkadi Nemirovski. *Robust solutions of uncertain linear programs.* Operations research letters, vol. 25, no. 1, pages 1–13, 1999. (Cité en page 26.)

[Ben-Tal 2000] Aharon Ben-Tal et Arkadi Nemirovski. *Robust solutions of linear programming problems contaminated with uncertain data.* Mathematical programming, vol. 88, pages 411–424, 2000. (Cité en pages 23 et 26.)

[Ben-Tal 2004] Aharon Ben-Tal, Alexander Goryashko, Elana Guslitzer et Arkadi Nemirovski. *Adjustable robust solutions of uncertain linear programs*. Mathematical Programming, vol. 99, no. 2, pages 351–376, 2004. (Cité en page 25.)

[Ben-Tal 2009] Aharon Ben-Tal, Laurent El Ghaoui et Arkadi Nemirovski. Robust optimization, volume 28. Princeton university press, 2009. (Cité en page 24.)

[Benders 1962] J.F Benders. *Partitioning procedures for solving mixed-variables programming problems*. Numerische mathematik, vol. 4, no. 1, pages 238–252, 1962. (Cité en page 15.)

[Bertsimas 2003] Dimitris Bertsimas et Melvyn Sim. *Robust discrete optimization and network flows*. Mathematical programming, vol. 98, no. 1-3, pages 49–71, 2003. (Cité en page 29.)

[Bertsimas 2004] Dimitris Bertsimas et Melvyn Sim. *The price of robustness*. Operations Research, vol. 52, no. 1, pages 35–53, 2004. (Cité en pages 27, 28, 32, 81 et 107.)

[Bertsimas 2016] Dimitris Bertsimas, Iain Dunning et Miles Lubin. *Reformulation versus cutting-planes for robust optimization: A computational study*. Computational Management Science, vol. 13, pages 195–217, 2016. (Cité en page 29.)

[Bold 2021] Matthew Bold et Marc Goerigk. *A compact reformulation of the two-stage robust resource-constrained project scheduling problem*. Computers & Operations Research, vol. 130, page 105232, 2021. (Cité en page 32.)

[Bold 2022] Matthew Bold et Marc Goerigk. *A faster exact method for solving the robust multi-mode resource-constrained project scheduling problem*. Operations Research Letters, vol. 50, no. 5, pages 581–587, 2022. (Cité en page 32.)

[Bougeret 2019] Marin Bougeret, Artur Alves Pessoa et Michaël Poss. *Robust scheduling with budgeted uncertainty*. Discrete Applied Mathematics, vol. 261, pages 93–107, 2019. (Cité en page 31.)

[Bougeret 2021] Marin Bougeret, Klaus Jansen, Michael Poss et Lars Rohwedder. *Approximation results for makespan minimization with budgeted uncertainty*. Theory of Computing Systems, vol. 65, pages 903–915, 2021. (Cité en page 31.)

[Bougeret 2023] Marin Bougeret, Artur Pessoa et Michael Poss. *Single machine robust scheduling with budgeted uncertainty*. Operations Research Letters, vol. 51, no. 2, pages 137–141, 2023. (Cité en page 31.)

[Bowman 1959] Edward H. Bowman. *The schedule-sequencing problem*. Operations Research, vol. 7, no. 5, pages 621–624, 1959. (Cité en pages 13 et 64.)

[Brandimarte 1993] Paolo Brandimarte. *Routing and scheduling in a flexible job shop by tabu search*. Annals of Operations research, vol. 41, no. 3, pages 157–183, 1993. (Cité en pages 20 et 21.)

[Brucker 1990] Peter Brucker et Rainer Schlie. *Job-shop scheduling with multi-purpose machines*. Computing, vol. 45, no. 4, pages 369–375, 1990. (Cité en page 51.)

[Brucker 1999] Peter Brucker, Svetlana A. Kravchenko et Yuri N. Sotskov. *Preemptive job-shop scheduling problems with a fixed number of jobs.* Mathematical Methods of Operations Research, vol. 49, no. 1, pages 41–76, 1999. (Cité en page 63.)

[Bruni 2017] Maria Elena Bruni, Luigi Di Puglia Pugliese, Patrizia Beraldi et Francesca Guerriero. *An adjustable robust optimization model for the resource-constrained project scheduling problem with uncertain activity durations.* Omega, vol. 71, pages 66–84, 2017. (Cité en pages 32 et 87.)

[Carlier 1989] Jacques Carlier et Éric Pinson. *An algorithm for solving the job-shop problem.* Management science, vol. 35, no. 2, pages 164–176, 1989. (Cité en pages 13, 42 et 47.)

[Carlier 1994] Jacques Carlier et Eric Pinson. *Adjustment of heads and tails for the job-shop problem.* European Journal of Operational Research, vol. 78, no. 2, pages 146–161, 1994. (Cité en page 15.)

[Carlier 1996] Jacques Carlier et Ismaïl Rebaï. *Two branch and bound algorithms for the permutation flow shop problem.* European Journal of Operational Research, vol. 90, no. 2, pages 238–251, 1996. (Cité en page 13.)

[Chaudhry 2016] Imran Ali Chaudhry et Abid Ali Khan. *A research survey: review of flexible job shop scheduling techniques.* International Transactions in Operational Research, vol. 23, no. 3, pages 551–591, 2016. (Cité en page 51.)

[Chen 1995] Chuen-Lung Chen, Venkateswara S Vempati et Nasser Aljaber. *An application of genetic algorithms for flow shop problems.* European Journal of Operational Research, vol. 80, no. 2, pages 389–396, 1995. (Cité en page 18.)

[Creemers 2019] Stefan Creemers. *The preemptive stochastic resource-constrained project scheduling problem.* European Journal of Operational Research, vol. 277, no. 1, pages 238–247, 2019. (Cité en page 63.)

[Ćwik 2015] Michał Ćwik et Jerzy Józefczyk. *Evolutionary algorithm for minmax regret flow-shop problem.* Management and Production Engineering Review, vol. 6, 2015. (Cité en page 31.)

[Ćwik 2018] Michał Ćwik et Jerzy Józefczyk. *Heuristic algorithms for the min-max regret flow-shop problem with interval processing times.* Central European journal of operations research, vol. 26, pages 215–238, 2018. (Cité en page 31.)

[Dauzère-Pérès 1997] Stéphane Dauzère-Pérès et Jan Paulli. *An integrated approach for modeling and solving the general multiprocessor job-shop scheduling problem using tabu search.* Annals of Operations Research, vol. 70, pages 281–306, 1997. (Cité en pages 20 et 21.)

[Della Croce 1995] Federico Della Croce, Roberto Tadei et Giuseppe Volta. *A genetic algorithm for the job shop problem.* Computers & Operations Research, vol. 22, no. 1, pages 15–24, 1995. (Cité en page 18.)

[Ebadi 2012] Abbas Ebadi et Ghasem Moslehi. *Mathematical models for preemptive shop scheduling problems.* Computers & Operations Research, vol. 39, no. 7, pages 1605–1614, 2012. (Cité en page 128.)

[Ebadi 2013] Abbas Ebadi et Ghasem Moslehi. *An optimal method for the preemptive job shop scheduling problem.* Computers & Operations Research, vol. 40, no. 5, pages 1314–1327, 2013. (Cité en pages xiii, 68, 69, 70 et 71.)

[Erschler 1976] Jacques Erschler, François Roubellat et Jean-Paul Vernhes. *Finding some essential characteristics of the feasible solutions for a scheduling problem.* Operations Research, vol. 24, no. 4, pages 774–783, 1976. (Cité en page 15.)

[Etiler 2004] O. Etiler, Bilal Toklu, M. Atak et J. Wilson. *A genetic algorithm for flow shop scheduling problems.* Journal of the Operational Research Society, vol. 55, pages 830–835, 2004. (Cité en page 18.)

[Fattahi 2007] Parviz Fattahi, Mohammad Saidi Mehrabad et Fariborz Jolai. *Mathematical modeling and heuristic approaches to flexible job shop scheduling problems.* Journal of Intelligent Manufacturing, vol. 18, no. 3, pages 331–342, 2007. (Cité en pages 20 et 21.)

[Fisher 1963] Henry Fisher et Gerald Luther Thompson. *Probabilistic learning combinations of local job-shop scheduling rules.* In J.F. Muth et G.L. Thompson, editeurs, Industrial scheduling, pages 225–251. Prentice-Hall, Englewood Cliffs, NJ, 1963. (Cité en pages 19 et 20.)

[Giffler 1960] Bernard Giffler et Gerald Luther Thompson. *Algorithms for solving production-scheduling problems.* Operations Research, vol. 8, no. 4, pages 487–503, 1960. (Cité en page 13.)

[Glover 1989] Fred Glover. *Tabu search—part I.* ORSA Journal on Computing, vol. 1, no. 3, pages 190–206, 1989. (Cité en page 18.)

[Graham 1979] Ronald Lewis Graham, Eugene Leighton Lawler, Jan Karel Lenstra et AHG Rinnooy Kan. *Optimization and approximation in deterministic sequencing and scheduling: a survey.* In Annals of discrete mathematics, volume 5, pages 287–326. Elsevier, 1979. (Cité en page 7.)

[Hamaz 2018] Idir Hamaz, Laurent Houssin et Sonia Cafieri. *The Cyclic Job Shop Problem with uncertain processing times.* In 16th International Conference on Project Management and Scheduling (PMS 2018), pages 119–122, Rome, Italy, 2018. (Cité en page 31.)

[Hamaz 2023] Idir Hamaz, Laurent Houssin et Sonia Cafieri. *The robust cyclic job shop problem.* European Journal of Operational Research, 2023. (Cité en page 31.)

[Hooker 1995] John N. Hooker, Hong Yan, V. Saraswat et P. Van Hentenryck. *Verifying logic circuits by Benders decomposition.* In Principles and Practice of Constraint Programming: The Newport Papers, pages 267–288. MIT Press, Cambridge, MA, 1995. (Cité en page 16.)

[Hooker 2000] John N. Hooker. Logic based methods for optimization: Combining optimization and constraint satisfaction. John Wiley and Sons, New York, 2000. (Cité en page 16.)

[Hooker 2003] John N. Hooker et Greger Ottosson. *Logic-based Benders decomposition*. Mathematical Programming, vol. 96, no. 1, pages 33–60, 2003. (Cité en page 16.)

[Hooker 2007] John N. Hooker. *Planning and scheduling by logic-based Benders decomposition*. Operations Research, vol. 55, no. 3, pages 588–602, 2007. (Cité en page 16.)

[Hooker 2019] John N. Hooker. *Logic-based Benders decomposition for large-scale optimization*. Large Scale Optimization in Supply Chains and Smart Manufacturing: Theory and Applications, pages 1–26, 2019. (Cité en pages 16 et 17.)

[Huang 2016] Song Huang, Na Tian, Yan Wang et Zhicheng Ji. *Multi-objective flexible job-shop scheduling problem using modified discrete particle swarm optimization*. SpringerPlus, vol. 5, no. 1, pages 1–22, 2016. (Cité en page 19.)

[Hurink 1994] Johann Hurink, Bernd Jurisch et Monika Thole. *Tabu search for the job-shop scheduling problem with multi-purpose machines*. OR Spectrum, vol. 15, no. 4, pages 205–215, 1994. (Cité en pages 18, 20, 21 et 51.)

[IBM 2022] IBM. *Scheduling examples*. `https://www.ibm.com/docs/en/icos/22.1.1?topic=programming-scheduling-examples`, 2022. [En ligne]. (Cité en pages 53, 55 et 92.)

[Jain 1999] Anant Singh Jain et Sheik Meeran. *Deterministic job-shop scheduling: Past, present and future*. European journal of operational research, vol. 113, no. 2, pages 390–434, 1999. (Cité en page 19.)

[Jamili 2016] Amin Jamili. *Robust job shop scheduling problem: Mathematical models, exact and heuristic algorithms*. Expert systems with Applications, vol. 55, pages 341–350, 2016. (Cité en page 31.)

[Jansen 2005] Klaus Jansen, Monaldo Mastrolilli et Roberto Solis-Oba. *Approximation algorithms for flexible job shop problems*. International Journal of Foundations of Computer Science, vol. 16, no. 2, pages 361–379, 2005. (Cité en page 64.)

[Johnson 1954] Selmer Martin Johnson. *Optimal two-and three-stage production schedules with setup times included*. Naval Research Logistics Quarterly, vol. 1, no. 1, pages 61–68, 1954. (Cité en pages 17, 26 et 113.)

[Juvin 2022a] Carla Juvin, Laurent Houssin et Pierre Lopez. *Décomposition de Benders basée sur la logique pour le problème de job-shop flexible préemptif*. In 23ème congrès annuel de la Société Française de Recherche Opérationnelle et d'Aide à la Décision, 2022. (Cité en page 2.)

[Juvin 2022b] Carla Juvin, Laurent Houssin et Pierre Lopez. *Logic-based Benders decomposition for preemptive flexible job-shop scheduling*. In 18th Internatio-

nal Conference on Project Management and Scheduling (PMS 2022), 2022. (Cité en pages 2 et 38.)

[Juvin 2023a] Carla Juvin, Emmanuel Hebrard, Laurent Houssin et Pierre Lopez. *An Efficient Constraint Programming Approach to Preemptive Job Shop Scheduling*. In Roland H. C. Yap, editeur, 29th International Conference on Principles and Practice of Constraint Programming (CP 2023), volume 280 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 19 :1–19 :16, Dagstuhl, Germany, 2023. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. (Cité en page 128.)

[Juvin 2023b] Carla Juvin, Laurent Houssin et Pierre Lopez. *Approche hybride multi-étape pour la résolution du problème de job-shop flexible robuste avec budget d'incertitude*. In 24e congrès de la Société française de recherche opérationnelle et d'aide à la décision (ROADEF 2023), 2023. (Cité en page 2.)

[Juvin 2023c] Carla Juvin, Laurent Houssin et Pierre Lopez. *Constraint programming for the robust two-machine flow-shop scheduling problem with budgeted uncertainty*. In 20th International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research (CPAIOR), 2023. (Cité en pages 2 et 113.)

[Juvin 2023d] Carla Juvin, Laurent Houssin et Pierre Lopez. *Decomposition methods for the preemptive Flexible Job-Shop Scheduling Problem*. `https://hal.science/hal-04243944`, 2023. Rapport technique. (Cité en page 78.)

[Juvin 2023e] Carla Juvin, Laurent Houssin et Pierre Lopez. *Hybrid methods to solve the two-stage robust flexible job-shop scheduling problem with budgeted uncertainty*. In 12th International Conference on Operations Research and Enterprise Systems (ICORES), pages 135–142, 2023. (Cité en pages 2 et 82.)

[Juvin 2023f] Carla Juvin, Laurent Houssin et Pierre Lopez. *Logic-based Benders decomposition for the preemptive flexible job-shop scheduling problem*. Computers & Operations Research, page 106156, 2023. (Cité en pages 2 et 38.)

[Kacem 2002] Imed Kacem, Slim Hammadi et Pierre Borne. *Pareto-optimality approach for flexible job-shop scheduling problems: Hybridization of evolutionary algorithms and fuzzy logic*. Mathematics and Computers in Simulation, vol. 60, no. 3-5, pages 245–276, 2002. (Cité en pages 20 et 21.)

[Karlsson 2022] Emil Karlsson et Elina Rönnberg. *Logic-based Benders decomposition with a partial assignment acceleration technique for avionics scheduling*. Computers & Operations Research, vol. 146, page 105916, 2022. (Cité en page 16.)

[Kasperski 2012] Adam Kasperski, Adam Kurpisz et Paweł Zieliński. *Approximating a two-machine flow shop scheduling under discrete scenario uncertainty*. European Journal of Operational Research, vol. 217, no. 1, pages 36–43, 2012. (Cité en pages 26 et 31.)

[Katriel 2005] Irit Katriel, Laurent Michel et Pascal Van Hentenryck. *Maintaining longest paths incrementally.* Constraints, vol. 10, pages 159–183, 2005. (Cité en page 89.)

[Kirkpatrick 1983] Scott Kirkpatrick, C. Daniel Gelatt Jr et Mario P. Vecchi. *Optimization by simulated annealing.* science, vol. 220, no. 4598, pages 671–680, 1983. (Cité en page 18.)

[Kolonko 1999] Michael Kolonko. *Some new results on simulated annealing applied to the job shop scheduling problem.* European journal of operational research, vol. 113, no. 1, pages 123–136, 1999. (Cité en page 18.)

[Kondili 1988] E. Kondili et R.W.H. Sargent. A general algorithm for scheduling batch operations. Department of Chemical Engineering, Imperial College, 1988. (Cité en page 13.)

[Kouvelis 2000] Panos Kouvelis, Richard L. Daniels et George Vairaktarakis. *Robust scheduling of a two-machine flow shop with uncertain processing times.* IIE Transactions, vol. 32, no. 5, pages 421–432, 2000. (Cité en page 31.)

[Kravchenko 1995] Svetlana A. Kravchenko et Yuri N. Sotskov. *Complexity of the two machine job-shop scheduling problem with a fixed number of jobs.* Central European Journal for Operations Research and Economics, 1995. (Cité en page 63.)

[Ku 2016] Wen-Yang Ku et J Christopher Beck. *Mixed integer programming models for job shop scheduling: A computational analysis.* Computers & Operations Research, vol. 73, pages 165–173, 2016. (Cité en pages 13, 15, 54 et 56.)

[Laborie 2018] Philippe Laborie, Jérôme Rogerie, Paul Shaw et Petr Vilím. *IBM ILOG CP optimizer for scheduling: 20+ years of scheduling with constraints at IBM/ILOG.* Constraints, vol. 23, pages 210–250, 2018. (Cité en page 14.)

[Lawrence 1984a] Stephen Lawrence. *Resource constrained project scheduling: An experimental investigation of heuristic scheduling techniques (Supplement).* Graduate School of Industrial Administration, Carnegie-Mellon University, 1984. (Cité en page 19.)

[Lawrence 1984b] Stephen Lawrence. *Resource constrained project scheduling: An experimental investigation of heuristic scheduling techniques (Supplement).* Rapport technique, GSIA, Carnegie-Mellon University, 1984. (Cité en page 20.)

[Levorato 2022] Mario Levorato, Rosa Figueiredo et Yuri Frota. *Exact solutions for the two-machine robust flow shop with budgeted uncertainty.* European Journal of Operational Research, vol. 300, no. 1, pages 46–57, 2022. (Cité en pages 31, 109, 121 et 122.)

[Levorato 2023] Mario Levorato, David Sotelo, Rosa Figueiredo et Yuri Frota. *Robust permutation flow shop total weighted completion time problem: Solution and application to the oil and gas industry.* Computers & Operations Research, vol. 151, page 106117, 2023. (Cité en page 31.)

[Li 2022] Yantong Li, Jean-François Côté, Leandro Callegari-Coelho et Peng Wu. *Novel formulations and logic-based Benders decomposition for the integrated parallel machine scheduling and location problem.* INFORMS Journal on Computing, vol. 34, no. 2, pages 1048–1069, 2022. (Cité en page 16.)

[Lian 2008] Zhigang Lian, Xingsheng Gu et Bin Jiao. *A novel particle swarm optimization algorithm for permutation flow-shop scheduling to minimize makespan.* Chaos, Solitons & Fractals, vol. 35, no. 5, pages 851–861, 2008. (Cité en page 19.)

[Liao 1992] Ching-Jong Liao et Chii-Tsuen You. *An improved formulation for the job-shop scheduling problem.* Journal of the Operational Research Society, vol. 43, no. 11, pages 1047–1054, 1992. (Cité en page 13.)

[Lin 2010] Tsung-Lieh Lin, Shi-Jinn Horng, Tzong-Wann Kao, Yuan-Hsin Chen, Ray-Shine Run, Rong-Jian Chen, Jui-Lin Lai et I-Hong Kuo. *An efficient job-shop scheduling algorithm based on particle swarm optimization.* Expert Systems with Applications, vol. 37, no. 3, pages 2629–2636, 2010. (Cité en page 19.)

[Manne 1960] Alan S. Manne. *On the job-shop scheduling problem.* Operations Research, vol. 8, no. 2, pages 219–223, 1960. (Cité en pages 13, 54, 91, 93 et 109.)

[Mastrolilli 1998] Monaldo Mastrolilli. *Flexible Job Shop Problem.* `http://people.idsia.ch/~monaldo/fjsp.html`, 1998. Accessed: 2020-11-26. (Cité en page 20.)

[McMahon 1967] G.B. McMahon et P.G. Burton. *Flow-shop scheduling with the branch-and-bound method.* Operations Research, vol. 15, no. 3, pages 473–481, 1967. (Cité en page 13.)

[Meng 2020] Leilei Meng, Chaoyong Zhang, Yaping Ren, Biao Zhang et Chang Lv. *Mixed-integer linear programming and constraint programming formulations for solving distributed flexible job shop scheduling problem.* Computers & Industrial Engineering, vol. 142, page 106347, 2020. (Cité en pages 13, 15, 51, 54 et 66.)

[Naderi 2021] Bahman Naderi et Vahid Roshanaei. *Critical-path-search logic-based Benders decomposition approaches for flexible job shop scheduling.* INFORMS Journal on Optimization, 2021. (Cité en pages 37 et 46.)

[Naderi 2023] Bahman Naderi, Rubén Ruiz et Vahid Roshanaei. *Mixed-Integer Programming vs. Constraint Programming for Shop Scheduling Problems : New Results and Outlook.* INFORMS Journal on Computing, 2023. (Cité en page 15.)

[Najid 2002] N.M. Najid, Stéphane Dauzère-Pérès et Ali Zaidat. *A modified simulated annealing method for flexible job shop scheduling problem.* In IEEE international conference on systems, man and cybernetics, volume 5, pages 6–pp. IEEE, 2002. (Cité en pages 18 et 51.)

[Nowicki 1996] Eugeniusz Nowicki et Czesław Smutnicki. *A fast tabu search algorithm for the permutation flow-shop problem.* European Journal of Operational Research, vol. 91, no. 1, pages 160–175, 1996. (Cité en page 18.)

[Nowicki 2005] Eugeniusz Nowicki et Czesław Smutnicki. *An advanced tabu search algorithm for the job shop problem.* Journal of Scheduling, vol. 8, pages 145–159, 2005. (Cité en page 18.)

[Nuijten 1998] Wim Nuijten et Claude Le Pape. *Constraint-based job shop scheduling with ILOG SCHEDULER.* Journal of Heuristics, vol. 3, pages 271–286, 1998. (Cité en page 15.)

[Osman 1989] Ibrahim H. Osman et Chris N. Potts. *Simulated annealing for permutation flow-shop scheduling.* Omega, vol. 17, no. 6, pages 551–557, 1989. (Cité en page 18.)

[Ourari 2015] Samia Ourari, L. Berrandjia, R. Boulakhras, A. Boukciat et H. Hentous. *Robust approach for centralized job shop scheduling: Sequential flexibility.* IFAC-PapersOnLine, vol. 48, no. 3, pages 1960–1965, 2015. (Cité en page 31.)

[Özgüven 2010] Cemal Özgüven, Lale Özbakır et Yasemin Yavuz. *Mathematical models for job-shop scheduling problems with routing and process plan flexibility.* Applied Mathematical Modelling, vol. 34, no. 6, pages 1539–1548, 2010. (Cité en page 51.)

[Pezzella 2008] Ferdinando Pezzella, Gianluca Morganti et Giampiero Ciaschetti. *A genetic algorithm for the flexible job-shop scheduling problem.* Computers & Operations Research, vol. 35, no. 10, pages 3202–3212, 2008. (Cité en page 51.)

[Pinedo 2012] Michael L. Pinedo. Scheduling, volume 29. Springer, 2012. (Cité en page 5.)

[Polo-Mejía 2020] Oliver Polo-Mejía, Christian Artigues, Pierre Lopez et Virginie Basini. *Mixed-integer/linear and constraint programming approaches for activity scheduling in a nuclear research facility.* International Journal of Production Research, vol. 58, no. 23, pages 7149–7166, 2020. (Cité en pages 66 et 67.)

[Rahmaniani 2017] Ragheb Rahmaniani, Teodor Gabriel Crainic, Michel Gendreau et Walter Rei. *The Benders decomposition algorithm: A literature review.* European Journal of Operational Research, vol. 259, no. 3, pages 801–817, 2017. (Cité en page 16.)

[Rajendran 2004] Chandrasekharan Rajendran et Hans Ziegler. *Ant-colony algorithms for permutation flowshop scheduling to minimize makespan/total flowtime of jobs.* European Journal of Operational Research, vol. 155, no. 2, pages 426–438, 2004. (Cité en page 18.)

[Reeves 1993] Colin R. Reeves. *Improving the efficiency of tabu search for machine sequencing problems.* Journal of the operational research society, vol. 44, no. 4, pages 375–382, 1993. (Cité en page 18.)

[Reeves 1995] Colin R. Reeves. *A genetic algorithm for flowshop sequencing.* Computers & operations research, vol. 22, no. 1, pages 5–13, 1995. (Cité en page 18.)

[Rossi 2006] Francesca Rossi, Peter Van Beek et Toby Walsh. Handbook of constraint programming. Elsevier, 2006. (Cité en page 14.)

[Rossi 2007] Andrea Rossi et Gino Dini. *Flexible job-shop scheduling with routing flexibility and separable setup times using ant colony optimisation method.* Robotics and Computer-Integrated Manufacturing, vol. 23, no. 5, pages 503–516, 2007. (Cité en page 18.)

[Roy 1964] Bernard Roy et B. Sussmann. *Les problèmes d'ordonnancement avec contraintes disjonctives.* Note DS, SEMA, no. 9 bis, 1964. (Cité en page 9.)

[Sels 2012] Veronique Sels, Nele Gheysen et Mario Vanhoucke. *A comparison of priority rules for the job shop scheduling problem under different flow time- and tardiness-related objective functions.* International Journal of Production Research, vol. 50, no. 15, pages 4255–4270, 2012. (Cité en page 17.)

[Shen 2018] Liji Shen, Stéphane Dauzère-Pérès et Janis S. Neufeld. *Solving the flexible job shop scheduling problem with sequence-dependent setup times.* European Journal of Operational Research, vol. 265, no. 2, pages 503–516, 2018. (Cité en pages 18, 51, 52 et 98.)

[Song 2022] Guopeng Song et Roel Leus. *Parallel machine scheduling under uncertainty : Models and exact algorithms.* INFORMS Journal on Computing, vol. 34, no. 6, pages 3059–3079, 2022. (Cité en page 33.)

[Soyster 1973] Allen L. Soyster. *Convex programming with set-inclusive constraints and applications to inexact linear programming.* Operations Research, vol. 21, no. 5, pages 1154–1157, 1973. (Cité en pages 26 et 27.)

[Stafford 1988] Edward F. Stafford. *On the development of a mixed-integer linear programming model for the flowshop sequencing problem.* Journal of the Operational Research Society, vol. 39, no. 12, pages 1163–1174, 1988. (Cité en page 109.)

[Storer 1992] Robert H. Storer, S. David Wu et Renzo Vaccari. *New search spaces for sequencing problems with application to job shop scheduling.* Management Science, vol. 38, no. 10, pages 1495–1509, 1992. (Cité en page 19.)

[Stützle 1998] Thomas Stützle. *An ant approach to the flow shop problem.* In Proceedings of the 6th European Congress on Intelligent Techniques & Soft Computing (EUFIT'98), volume 3, pages 1560–1564. Verlag Mainz, Wissenschaftsverlag Aachen, 1998. (Cité en page 18.)

[Subramaniam 2000] V. Subramaniam, G.K. Lee, T. Ramesh, G.S. Hong et Y.S. Wong. *Machine selection rules in a dynamic job shop.* The International Journal of Advanced Manufacturing Technology, vol. 16, pages 902–908, 2000. (Cité en page 30.)

[Sun 2019] Defeng Sun, Lixin Tang et Roberto Baldacci. *A Benders decomposition-based framework for solving quay crane scheduling problems.* European Journal of Operational Research, vol. 273, no. 2, pages 504–515, 2019. (Cité en page 16.)

[Taillard 1993] Eric Taillard. *Benchmarks for basic scheduling problems.* European Journal of Operational Research, vol. 64, no. 2, pages 278–285, 1993. (Cité en page 19.)

[Taillard 1994] Eric Taillard. *Parallel taboo search techniques for the job shop scheduling problem.* ORSA Journal on Computing, vol. 6, no. 2, pages 108–117, 1994. (Cité en page 18.)

[Tan 2018] Yingcong Tan et Daria Terekhov. *Logic-based Benders decomposition for two-stage flexible flow shop scheduling with unrelated parallel machines.* In Ebrahim Bagheri et Jackie C.K. Cheung, editeurs, Advances in Artificial Intelligence, pages 60–71, Cham, 2018. Springer International Publishing. (Cité en page 16.)

[Tasgetiren 2004] M. Fatih Tasgetiren, Mehmet Sevkli, Yun-Chia Liang et Gunes Gencyilmaz. *Particle swarm optimization algorithm for permutation flowshop sequencing problem.* In Ant Colony Optimization and Swarm Intelligence: 4th International Workshop, ANTS 2004, Brussels, Belgium, September 5-8, 2004. Proceedings 4, pages 382–389. Springer, 2004. (Cité en page 19.)

[Torres 2000] Philippe Torres et Pierre Lopez. *On not-first/not-last conditions in disjunctive scheduling.* European Journal of Operational Research, vol. 127, no. 2, pages 332–343, 2000. (Cité en page 15.)

[Tran 2016] Tony T. Tran, Arthur Araujo et J. Christopher Beck. *Decomposition methods for the parallel machine scheduling problem with setups.* INFORMS Journal on Computing, vol. 28, no. 1, pages 83–95, 2016. (Cité en page 16.)

[Tseng 2001] Fan T. Tseng et Edward F. Stafford Jr. *Two MILP models for the N×M SDST flowshop sequencing problem.* International Journal of Production Research, vol. 39, no. 8, pages 1777–1809, 2001. (Cité en pages 109 et 110.)

[Tseng 2004] Fan T. Tseng, Edward F. Stafford Jr et Jatinder N.D. Gupta. *An empirical analysis of integer programming formulations for the permutation flowshop.* Omega, vol. 32, no. 4, pages 285–293, 2004. (Cité en pages 13 et 109.)

[Van de Vonder 2007] Stijn Van de Vonder, Francisco Ballestin, Erik Demeulemeester et Willy Herroelen. *Heuristic procedures for reactive project scheduling.* Computers & Industrial Engineering, vol. 52, no. 1, pages 11–28, 2007. (Cité en page 30.)

[Van Laarhoven 1992] Peter J.M. Van Laarhoven, Emile H.L. Aarts et Jan Karel Lenstra. *Job shop scheduling by simulated annealing.* Operations Research, vol. 40, no. 1, pages 113–125, 1992. (Cité en page 18.)

[Wagner 1959] Harvey M. Wagner. *An integer linear-programming model for machine scheduling.* Naval Research Logistics Quarterly, vol. 6, no. 2, pages 131–140, 1959. (Cité en pages 13 et 109.)

[Wilson 1989] J.M. Wilson. *Alternative formulations of a flow-shop scheduling problem.* Journal of the Operational Research Society, vol. 40, no. 4, pages 395–399, 1989. (Cité en pages 13, 109 et 110.)

[Wolsey 1998] Laurence A. Wolsey. Integer programming. John Wiley & Sons, 1998. (Cité en page 13.)

[Xianzhou 2011] Cao Xianzhou et Yang Zhenhe. *An improved genetic algorithm for dual-resource constrained flexible job shop scheduling.* In 2011 Fourth International Conference on Intelligent Computation Technology and Automation, volume 1, pages 42–45. IEEE, 2011. (Cité en page 51.)

[Yazdani 2015] M. Yazdani, M. Zandieh, R. Tavakkoli-Moghaddam et F. Jolai. *Two meta-heuristic algorithms for the dual-resource constrained flexible job-shop scheduling problem.* Scientia Iranica, vol. 22, no. 3, pages 1242–1257, 2015. (Cité en pages 18 et 51.)

[Ying 2004] Kuo-Ching Ying et Ching-Jong Liao. *An ant colony system for permutation flow-shop sequencing.* Computers & Operations Research, vol. 31, no. 5, pages 791–801, 2004. (Cité en page 18.)

[Ying 2015] Kuo-Ching Ying. *Scheduling the two-machine flowshop to hedge against processing time uncertainty.* Journal of the Operational Research Society, vol. 66, no. 9, pages 1413–1425, 2015. (Cité en pages 31, 121 et 123.)

[Zegordi 1995] Seyed Hessameddin Zegordi, Kenji Itoh et Takao Enkawa. *Minimizing makespan for flow shop scheduling by combining simulated annealing with sequencing knowledge.* European Journal of Operational Research, vol. 85, no. 3, pages 515–531, 1995. (Cité en page 18.)

[Zeng 2013] Bo Zeng et Long Zhao. *Solving two-stage robust optimization problems using a column-and-constraint generation method.* Operations Research Letters, vol. 41, no. 5, pages 457–461, 2013. (Cité en page 29.)

[Zhang 2016] Jiae Zhang et Jianjun Yang. *Flexible job-shop scheduling with flexible workdays, preemption, overlapping in operations and satisfaction criteria: An industrial application.* International Journal of Production Research, vol. 54, no. 16, pages 4894–4918, 2016. (Cité en page 63.)

[Ziaee 2018] Mohsen Ziaee. *A mixed integer linear programming model for flexible job shop scheduling problem.* International Journal of Mathematical and Computational Sciences, vol. 12, no. 6, pages 95–99, 2018. (Cité en page 51.)

[Zobolas 2009] G.I. Zobolas, Christos D. Tarantilis et George Ioannou. *Minimizing makespan in permutation flow shop scheduling problems using a hybrid metaheuristic algorithm.* Computers & Operations Research, vol. 36, no. 4, pages 1249–1267, 2009. (Cité en page 18.)