



**PERMUTATION DANS LES PROBLÈMES D'ORDONNANCEMENT D'ATELIERS :
AMBIGUITÉS ET EXTENSIONS**

PAR RANDA OUCHENE

**THÈSE PRÉSENTÉE À L'UNIVERSITÉ DU QUÉBEC À CHICOUTIMI DANS LE
CADRE D'UN PROGRAMME EN EXTENSION DE L'UNIVERSITÉ DU QUÉBEC
EN OUTAOUAIS EN VUE DE L'OBTENTION DU GRADE DE PHILOSOPHIÆ
DOCTOR (PH.D.) EN SCIENCES ET TECHNOLOGIES DE L'INFORMATION**

QUÉBEC, CANADA

© RANDA OUCHENE, 2025

RÉSUMÉ

Cette thèse se consacre à l'analyse et à l'extension de la notion de permutation dans divers environnements d'ordonnancement d'ateliers, en mettant l'accent sur les problèmes de flow shop et de job shop, et en se concentrant sur l'optimisation des critères basés sur les temps de fin de traitement.

Dans un premier temps, nous avons identifié l'ambiguïté liée à la définition de la permutation dans un flow shop comportant des opérations manquantes. Face à cette ambiguïté, nous avons proposé deux nouvelles définitions, fondées sur la transitivité (permutation forte et permutation faible). De plus, nous avons adopté la définition de la discipline FIFO proposée dans PINEDO, 2002c et montré que les différentes définitions ne sont plus équivalentes en présence d'opérations manquantes, contrairement au cas classique sans opérations manquantes. Cette étude a conduit à l'établissement d'un cadre formel visant à clarifier ces différentes notions et à mieux appréhender leurs conséquences sur la performance de l'ordonnancement.

Ensuite, nous avons étudié le compromis entre la complexité de résolution et la qualité des solutions dans les flow shops avec opérations manquantes. Nos résultats montrent que l'écart de performance entre les approches à permutation et à non-permutation peut varier considérablement selon la structure de l'instance. Pour distinguer les situations où une approche non-permutation, plus complexe à mettre en œuvre, apporte un réel bénéfice, nous avons proposé plusieurs indicateurs structurels d'instances. Parmi eux, l'indicateur IND_4 s'est avéré particulièrement pertinent, établissant une forte corrélation avec l'écart de performance et soulignant ainsi son utilité pour orienter le choix de la stratégie d'ordonnancement.

Enfin, nous avons étendu notre étude au contexte du job shop, en nous appuyant sur la définition de permutation proposée par PINEDO, 2002c, initialement formulée pour le flow shop et basée sur la discipline FIFO. Dans ce type de problème, où l'objectif est de minimiser le makespan, certaines opérations peuvent bénéficier d'une priorité plus élevée, leur permettant de passer devant d'autres dans la file d'attente des machines. L'application stricte de la règle FIFO peut ainsi engendrer des délais supplémentaires, car elle peut contraindre à bloquer l'exécution d'opérations moins prioritaires, même lorsque des machines sont disponibles, afin que la tâche prioritaire reste en tête de file. Afin de quantifier et d'analyser cet effet, nous avons développé une modélisation en programmation en nombres entiers et étudié l'impact de l'ajout d'une contrainte no-delay et FIFO sur la performance globale du système. Nos expérimentations ont révélé que le respect strict de la discipline FIFO tend à augmenter les délais, tandis que l'imposition de la contrainte no-delay peut, à l'inverse, conduire à des violations de la règle FIFO. L'activation simultanée de ces deux contraintes entraîne une hausse significative du makespan.

ABSTRACT

This thesis focuses on the analysis and extension of the notion of permutation in various shop scheduling environments, emphasizing flow shop and job shop problems, with a particular focus on optimizing criteria based on completion times.

First, we identified the ambiguity related to the definition of permutation in a flow shop with missing operations. To address this ambiguity, we proposed two new definitions based on transitivity (strong permutation and weak permutation). Additionally, we adopted the definition of the FIFO discipline proposed in PINEDO, 2002c and demonstrated that the different definitions are no longer equivalent in the presence of missing operations, unlike in the classical case without missing operations. This study led to the establishment of a formal framework to clarify these different notions and better understand their impact on scheduling performance.

Next, we studied the trade-off between resolution complexity and solution quality in flow shops with missing operations. Our results show that the performance gap between permutation and non-permutation approaches can vary significantly depending on the structure of the instance. To distinguish situations where a non-permutation approach, which is more complex to implement, provides a real benefit, we proposed several structural instance indicators. Among them, the IND_4 indicator proved to be particularly relevant, establishing a strong correlation with the performance gap and highlighting its usefulness in guiding the choice of scheduling strategy.

Finally, we extended our study to the job shop context, relying on the permutation definition proposed by PINEDO, 2002c, initially formulated for the flow shop and based on the FIFO discipline. In this type of problem, where the objective is to minimize the makespan, certain operations may be assigned higher priority, allowing them to be processed ahead of others in machine queues. The strict application of the FIFO rule can introduce additional delays, as it may require blocking the execution of lower-priority operations, even when machines are available, to ensure that the priority task remains at the front of the queue. To quantify and analyze this effect, we developed an integer programming model and studied the impact of adding both no-delay and FIFO constraints on overall system performance. Our experiments revealed that strictly enforcing the FIFO discipline tends to increase delays, while imposing the no-delay constraint can, conversely, lead to violations of the FIFO rule. The simultaneous activation of these two constraints results in a significant increase in the makespan.

TABLE DES MATIÈRES

RÉSUMÉ	ii
ABSTRACT	iii
LISTE DES TABLEAUX	viii
LISTE DES FIGURES	ix
REMERCIEMENTS	xiii
AVANT-PROPOS	xv
INTRODUCTION	1
CHAPITRE I – NOTIONS DE BASE	7
1.1 NOTIONS DE BASE EN ORDONNANCEMENT	7
1.1.1 DÉFINITION DES PROBLÈMES D’ORDONNANCEMENT	7
1.1.2 FORMULATION DES PROBLÈMES D’ORDONNANCEMENT	8
1.2 NOTIONS DE BASE EN COMPLEXITÉ	14
1.2.1 COMPLEXITÉ ALGORITHMIQUE	14
1.2.2 CLASSES DE COMPLEXITÉ	14
1.3 APPROCHES DE RÉOLUTION DES PROBLÈMES D’ORDONNANCEMENT	18
1.3.1 MÉTHODES EXACTES	18
1.3.2 MÉTHODES APPROCHÉES	19
CHAPITRE II – REVUE DE LA LITTÉRATURE	23
2.1 APPLICATIONS INDUSTRIELLES DES PROBLÈMES D’ATELIER : DES CHÂÎNES DE MONTAGE À L’INDUSTRIE 4.0	23
2.2 APPLICATIONS DES PROBLÈMES D’ATELIER DANS LES SERVICES	24
2.3 FLOW SHOP	25
2.3.1 FLOW SHOP DE PERMUTATION	26
2.3.2 FLOW SHOP DE NON-PERMUTATION	29
2.3.3 FLOW SHOP EN PRÉSENCE D’OPÉRATIONS MANQUANTES	30

2.4	JOB SHOP	31
CHAPITRE III – AMBIGUITY OF THE DEFINITION OF PERMUTATION FLOW SHOPS IN THE PRESENCE OF MISSING OPERATIONS		33
ABSTRACT		34
3.1	INTRODUCTION	34
3.2	AMBIGUITY OF THE DEFINITION OF A PERMUTATION	37
3.3	SPECIAL CASES OF NON AMBIGUITY	44
3.4	CONCLUSION	50
CHAPITRE IV – FLOW SHOP AVEC OPÉRATIONS MANQUANTES		54
4.1	NOTION DE PERMUTATION SELON PINEDO	54
4.2	FLOW SHOP UNITAIRE AVEC OPÉRATIONS MANQUANTES	62
4.3	CONCLUSION	76
CHAPITRE V – THE IMPACT OF MISSING OPERATIONS ON THE FLOW SHOP : PERMUTATION VS NON PERMUTATION MODES		78
ABSTRACT		79
5.1	INTRODUCTION	79
5.2	REVIEW OF THE LITERATURE	81
5.3	FLOW SHOP PROBLEM WITH MISSING OPERATIONS	83
5.3.1	MATHEMATICAL PROGRAMMING APPROACH	84
5.4	DESCRIPTION OF THE INDICATORS	87
5.5	COMPUTATIONAL STUDY	95
5.6	RESULTS AND DISCUSSION	97
5.6.1	STATISTICAL ANALYSIS WITH RESPECT TO THE MAKESPAN	98
5.6.2	STATISTICAL ANALYSIS WITH RESPECT TO THE TOTAL COMPLETION TIME	106
5.7	CONCLUSION	112
5.8	APPENDIX : THE NO WAIT CASE	116
CHAPITRE VI – THE IMPACT OF SERVICE CONSTRAINTS ON THE MAKESPAN IN A JOB SHOP PROBLEM		119

ABSTRACT	120
6.1 INTRODUCTION	120
6.2 LITERATURE REVIEW	122
6.3 MIXED INTEGER PROGRAMMING APPROACH	125
6.3.1 CLASSICAL DISJUNCTIVE JOB SHOP MODEL	126
6.3.2 JOB SHOP WITH THE FIFO CONSTRAINT	128
6.3.3 JOB SHOP WITH THE NO-DELAY CONSTRAINT	131
6.3.4 JOB SHOP WITH THE FIFO AND NO-DELAY CONSTRAINTS	135
6.3.5 JOB SHOP SCHEDULING WITH A SECONDARY CRITERION	136
6.3.6 JOB SHOP SCHEDULING WITH CONSTRAINTS AND A SECONDARY CRITERION	142
6.4 COMPUTATIONAL STUDY	143
6.4.1 METHODOLOGY	143
6.4.2 RESULTS	146
6.5 CONCLUSION	151
CONCLUSION	157
APPENDICE A – COMMUNICATION À CIGI-QUALITA-MOSIM 2023	165
A.1 INTRODUCTION	166
A.2 FLOW SHOP AVEC LA CONTRAINTE DE NO-WAIT	170
A.2.1 DESCRIPTION DES INDICATEURS	171
A.2.2 TESTS ET DISCUSSIONS	173
A.3 FLOW SHOP SANS LA CONTRAINTE DE NO-WAIT	176
A.4 CONCLUSION	178
APPENDICE B – COMMUNICATION À ROADEF 2024 (AMIENS)	181
B.1 INTRODUCTION	181
B.2 COMPARAISON DES MODÈLES $J \parallel C_{\max}$ ET $J \mid \text{FIFO} \mid C_{\max}$ (JOB-SHOP À PERMUTATION)	181
B.2.1 MODÈLE MATHÉMATIQUE	182

B.2.2	RÉSULTATS SUR LES BENCHMARKS	182
B.2.3	RÉSULTATS SUR DES INSTANCES ALÉATOIRES	183
B.3	CONCLUSION	183
APPENDICE C – COMMUNICATION À ROADEF 2025 (CHAMPS-SUR-MARNE)		185
C.1	INTRODUCTION	185
C.2	ÉTUDE EXPÉRIMENTALE	186
C.2.1	DESCRIPTION DE L’INDICATEUR	186
C.2.2	MÉTHODOLOGIE	186
C.2.3	RÉSULTATS	187
C.3	CONCLUSION	188
APPENDICE D – COMMUNICATION À CODIT 2025 (SPLIT)		190
ABSTRACT		191
D.1	INTRODUCTION	191
D.2	PINEDO PERMUTATION FOR FLOW SHOP WITH MISSING OPERATIONS	193
D.3	ADAPTING PINEDO’S PERMUTATION CONCEPT TO THE JOB SHOP ENVIRONMENT	197
D.3.1	MATHEMATICAL MODEL	198
D.3.2	COMPUTATIONAL STUDY	201
D.4	CONCLUSION	203

LISTE DES TABLEAUX

TABLEAU 3.1 :	PROCESSING TIMES OF THE JOBS	43
TABLEAU 5.1 :	SUMMARY OF GROUP 1.2 AND GROUP 2 FOR E_1	98
TABLEAU 5.2 :	SPEARMAN CORRELATION RESULTS	100
TABLEAU 5.3 :	ANOVA TEST RESULTS FOR E_1 BASED ON $IND_1, IND_2, IND_3,$ AND IND_4	102
TABLEAU 5.4 :	ANOVA TEST RESULTS FOR E_1 BASED ON THE NUMBER OF MACHINES, JOBS, AND PROCESSING TIME INTERVALS	104
TABLEAU 5.5 :	SUMMARY RESULTS OF GROUP 1.2 AND GROUP 2 FOR E_2 . .	107
TABLEAU 5.6 :	SPEARMAN CORRELATION TEST RESULTS FOR E_2	108
TABLEAU 5.7 :	ANOVA TEST RESULTS FOR E_2 ACROSS INDICATORS	110
TABLEAU 5.8 :	ANOVA TEST RESULTS FOR E_2 BASED ON THE NUMBER OF JOBS, MACHINES, AND PROCESSING TIME INTERVALS	111
TABLEAU 6.1 :	SUMMARIZED TABLE	147
TABLEAU D.1 :	RESULTS ON THE CLASSICAL BENCHMARK INSTANCES . . .	205

LISTE DES FIGURES

FIGURE 1.1 – Exemple de diagramme de Gantt.	8
FIGURE 1.2 – Contrainte de permutation dans un flow shop : exécution des tâches dans le même ordre sur toutes les machines	12
FIGURE 3.1 – Illustrative example for case 1	38
FIGURE 3.2 – Visualizing graph $G = (V, E)$ representing the binary relations between jobs as in Case 1	39
FIGURE 3.3 – Illustrative example for case 2	39
FIGURE 3.4 – Visualizing graph $G = (V, E)$ representing the binary relations between jobs as in Case 2	40
FIGURE 3.5 – Illustrative example for case 3	40
FIGURE 3.6 – Visualizing graph $G = (V, E)$ representing the binary relations between jobs as in case 3.	41
FIGURE 3.7 – Permutation scheme according to Definition 2 but not to Definition 1 . .	43
FIGURE 3.8 – Optimal solution using Definition 2	44
FIGURE 3.9 – Optimal solution using Definition 1	44
FIGURE 3.10 – Cases of non transitivity.. . . .	47
FIGURE 3.11 – Correction of the case of non transitivity	48
FIGURE 3.12 – (J_a, J_h, J_k) is the first non-transitive encountered case.	48
FIGURE 3.13 – This configuration is not possible for a schedule $s \in D_2(x)$	49
FIGURE 4.1 – Ordonnancement respectant la permutation forte mais pas la définition FIFO de Pinedo	56
FIGURE 4.2 – Ordonnancement respectant la FIFO de Pinedo, mais n'étant ni une permutation faible ni forte.	57
FIGURE 4.3 – Inclusion des permutations faibles, fortes et selon Pinedo.	58
FIGURE 4.4 – Inclusion des permutations faibles, fortes et selon Pinedo.	58

FIGURE 4.5 – Structure d'un ordonnancement O respectant l'ordre de L .	65
FIGURE 4.6 – Solution produite par l'heuristique $\mathbf{H}$.	66
FIGURE 4.7 – Solution optimale de l'instance I avec $C_{\max} = 2$.	67
FIGURE 4.8 – Structure de l'ordonnancement O	70
FIGURE 4.9 – Structure de l'ordonnancement O'	71
FIGURE 4.10 – Solution produite par l'heuristique $\mathbf{H}_1$.	73
FIGURE 4.11 – Solution optimale de l'instance I_1 avec $C_{\max} = 2$.	74
FIGURE 4.12 – Solution produite par l'heuristique $\mathbf{H}_2$.	75
FIGURE 4.13 – Solution optimale de l'instance I_2 avec $C_{\max} = 2$.	76
FIGURE 5.1 – Gantt chart illustrating notation.	89
FIGURE 5.2 – Scatter plot of E_1 vs IND_2 .	100
FIGURE 5.3 – Scatter plot of E_1 vs IND_3 .	100
FIGURE 5.4 – Scatter plot of E_1 vs IND_4 .	100
FIGURE 5.5 – Relationship between E_1 and IND_2 , IND_3 , and IND_4	100
FIGURE 5.6 – E_1 vs IND_1	103
FIGURE 5.7 – E_1 vs IND_2	103
FIGURE 5.8 – E_1 vs IND_3	103
FIGURE 5.9 – E_1 vs IND_4	103
FIGURE 5.10 – Distribution of E_1 across different categories of IND_1 , IND_2 , IND_3 , and IND_4 .	103
FIGURE 5.11 – Impact of the number of jobs	104
FIGURE 5.12 – Impact of the number of machines	104
FIGURE 5.13 – Impact of processing time intervals	104
FIGURE 5.14 – Impact of the number of machines, jobs, and processing time intervals on E_1	104

FIGURE 5.15 – Impact of case types on the makespan gap E_1	105
FIGURE 5.16 – Scatter plot of E_1 vs IND_2	108
FIGURE 5.17 – Scatter plot of E_1 vs IND_3	108
FIGURE 5.18 – Scatter plot of E_2 vs IND_4	108
FIGURE 5.19 – Relationship between E_1 and IND_2 , IND_3 , and IND_4	108
FIGURE 5.20 – E_2 vs IND_1	109
FIGURE 5.21 – E_2 vs IND_2	109
FIGURE 5.22 – E_2 vs IND_3	109
FIGURE 5.23 – E_2 vs IND_4	109
FIGURE 5.24 – Distribution of E_2 across different categories of IND_1 , IND_2 , IND_3 , and IND_4	109
FIGURE 5.25 – E_2 vs Number of Jobs	111
FIGURE 5.26 – E_2 vs Number of Machines	111
FIGURE 5.27 – E_2 vs Processing Time Intervals	111
FIGURE 5.28 – Impact of the number of jobs, number of machines, and processing time intervals on E_2	111
FIGURE 5.29 – Impact of case types on the gap E_2	112
FIGURE 5.30 – Representation of the necessary condition on job peers.	118
FIGURE 6.1 – A visual representation of job shop scheduling models with FIFO and No-Delay constraints, aimed at minimizing makespan, delay, and FIFO violations.	125
FIGURE 6.2 – Scheduling with respect to FIFO in a job shop problem	129
FIGURE 6.3 – Scheduling without the FIFO constraint in a job shop problem	129
FIGURE 6.4 – Inactivity time I_{ij} on machine j before the start of job i	132
FIGURE 6.5 – Scheduling with Respect to FIFO in a job shop problem without delay .	135
FIGURE 6.6 – Scheduling with respect to FIFO in a job shop problem with delay . . .	135

FIGURE 6.7 – Configuration for $x_{i_2 i_2 j} = 1$	141
FIGURE 6.8 – Configuration for $x_{i_2 i_2 j} = 0$	141
FIGURE 6.9 – Illustration of the models, constraints, and indicators used across the three methodological steps.	145
FIGURE 6.10 – Comparison of GAP values for FIFO, FIFO with No-delay, and No- delay constraints	147
FIGURE 6.11 – Minimum FIFO violations (while preserving $C_{\max}^{\text{JSP}}$) vs. average viola- tions observed over 10 $J//C_{\max}$ optimizations.	149
FIGURE 6.12 – Minimum delay (while preserving $C_{\max}^{\text{JSP}}$) vs. average delay over 10 $J//C_{\max}$ optimizations.	149
FIGURE 6.13 – Comparison of delays with and without imposing the FIFO constraint. .	150
FIGURE 6.14 – Comparison of FIFO violations with and without the No-delay Constraint 151	
FIGURE A.1 – Exemple d’ordonnancement sans la contrainte de permutation pour un flow shop avec la contrainte de no-wait	170
FIGURE A.2 – Représentation de la condition nécessaire sur les pairs de tâche.	172
FIGURE A.3 – Graphique à barres pour les catégories de l’indicateur et l’écart.	175
FIGURE C.1 – Relation entre IND et E_1 (gauche) et E_2 (droite).	187
FIGURE C.2 – Boxplots de E_1 (gauche) et E_2 (droite) selon les catégories d’IND	188
FIGURE D.1 – Illustration of strong permutation scheduling.	193
FIGURE D.2 – Illustration of weak permutation scheduling	194
FIGURE D.3 – Pinedo’s permutation scheduling for a Flow Shop with missing opera- tions, illustrating a scenario that is neither a weak nor a strong permuta- tion.	194
FIGURE D.4 – Scheduling with respect to FIFO in a job shop problem	198
FIGURE D.5 – Scheduling without the FIFO constraint in a job shop problem	198
FIGURE D.6 – Comparison of average waiting time for JSP and PJSP	203

REMERCIEMENTS

Cette thèse est l'aboutissement de plusieurs années de travail, d'apprentissage, de doutes et de persévérance. Elle n'aurait jamais vu le jour sans le soutien inestimable de nombreuses personnes à qui j'aimerais exprimer ma plus profonde gratitude.

Tout d'abord, un immense merci à mon directeur de thèse, Djamel Rebaïne, pour sa patience, sa rigueur et son dévouement. Son encadrement attentif, sa disponibilité et ses précieux conseils m'ont guidée à chaque étape de cette recherche. Son soutien financier a également été d'une grande aide, et je lui suis infiniment reconnaissante pour la confiance qu'il m'a accordée.

Je tiens aussi à exprimer ma profonde reconnaissance au professeur Pierre Baptiste, pour la richesse de nos échanges et la qualité de son accompagnement. J'ai apprécié chaque discussion, qui a été une source précieuse de réflexion et d'amélioration. Son soutien, tant scientifique que financier, a joué un rôle essentiel dans la réalisation de ce travail.

Je remercie l'Université du Québec à Chicoutimi (UQAC) ainsi que le Centre de recherche GERAD pour m'avoir offert un cadre de travail stimulant et des ressources essentielles à l'aboutissement de cette thèse. Je suis également reconnaissante envers la Fondation de l'UQAC pour la bourse d'excellence qui a facilité mon parcours, ainsi qu'au GERAD pour la bourse de co-supervision qui m'a apporté un soutien précieux.

Je remercie sincèrement les membres du jury, Alix Munier-Kordon et Hamid Mcheick, pour avoir accepté d'évaluer mon travail et pour le temps qu'ils consacreront à cette thèse. Leur expertise et leur regard critique seront une précieuse contribution à l'aboutissement de cette recherche, et je leur suis infiniment reconnaissante pour leur implication dans ce processus.

Durant ces années, j'ai eu la chance de croiser des âmes exceptionnelles qui ont illuminé mon parcours et ont rendu cette aventure plus douce, plus humaine et surtout plus mémorable. À mes amis du Canada, dont je préfère ne pas citer les noms de peur d'oublier quelqu'un, sachez que vous avez été ma force dans les moments de doute, mon refuge dans les périodes d'incertitude et ma joie dans chaque petite victoire. Vos éclats de rire, vos encouragements, vos discussions profondes et vos gestes d'amitié ont été le fil invisible qui m'a portée à travers ce voyage. Sans vous, cette thèse n'aurait pas eu la même saveur, et je mesure chaque jour la chance de vous avoir dans ma vie.

À mes chers parents, Djamilia Dahak et Nacer Ouchene, les piliers de mon existence, aucun mot ne pourra jamais traduire la reconnaissance infinie que je vous porte. Votre amour inconditionnel, vos sacrifices silencieux et votre soutien indéfectible m'ont donné la force d'avancer et de croire en moi. Chaque étape franchie est le fruit de votre dévouement et de votre amour, et cette réussite est autant la vôtre que la mienne.

À mon frère Abdelkader Ouchene, un modèle d'inspiration et un roc sur lequel je me suis toujours appuyée, merci pour ton soutien sans faille, tes conseils et ta présence rassurante à chaque tournant de mon parcours.

Enfin, à mon mari, Anis Noureddine, mon plus grand soutien et mon compagnon dans cette aventure extraordinaire. Nous avons traversé ensemble les épreuves et les défis du doctorat, et ton amour, ta patience et ta présence ont été ma plus grande source de force. Plus qu'un époux, tu as été un véritable partenaire, un confident et un moteur inépuisable de motivation. À travers chaque difficulté, nous avons grandi ensemble, et cette thèse est aussi la tienne. Peu importe où la vie nous mènera, je sais que nous continuerons à avancer main dans la main, portés par nos rêves et notre détermination.

À toutes ces personnes qui ont cru en moi et qui ont rendu ce parcours possible, merci du fond du cœur.

AVANT-PROPOS

Cette thèse s'inscrit dans l'étude des problèmes d'ordonnancement *flow shop* et *job shop*, avec une attention particulière portée à la notion de permutation et à ses implications. Les travaux menés dans le cadre de cette thèse ont donné lieu aux publications suivantes :

ARTICLES DANS DES REVUES SCIENTIFIQUES

- **Ouchene, R., Rebaine, D., & Baptiste, P.** (2023). *Ambiguity of the definition of permutation flow shops with missing operations*. **Computers & Industrial Engineering**, **182**, 109387.
DOI : <https://doi.org/10.1016/j.cie.2023.109387>
- **Ouchene, R., Rebaine, D., & Baptiste, P.** (2025). *The impact of missing operations on the flow shop : permutation vs non-permutation*. **RAIRO - Operations Research**, **59**(3), 1273–1293.
DOI : <https://doi.org/10.1051/ro/2025032>
- **Ouchene, R., Baptiste, P., & Rebaine, D.** (2025). *The impact of service constraints on the makespan in a job shop*. Soumis à *Annals of Operations Research*, mars 2025.

COMMUNICATIONS ET CONFÉRENCES

- **Ouchene, R., Rebaine, D., & Baptiste, P.** (juin 2023). *Impact des opérations manquantes sur le flow shop avec et sans la contrainte de permutation pour le makespan*. **CIGI Qualita MOSIM 2023**, Trois-Rivières, Québec, Canada.
- **Baptiste, P., Ouchene, R., & Rebaine, D.** (mars 2024). *Job shop de permutation !*. **25^e Congrès ROADEF 2024**, Amiens, France.
- **Ouchene, R., Baptiste, P., & Rebaine, D.** (mai 2024). *Interprétation et conséquences de la notion de flow shop de permutation en présence d'opérations manquantes*. **Optimization Days 2024**.
- **Ouchene, R., Baptiste, P., & Rebaine, D.** (février 2025). *Flow shop de permutation vs flow shop de non-permutation*. **26^e Congrès ROADEF 2025**, Paris, France.
- **IEEE CoDIT** (juillet 2025). *FIFO in scheduling problems with missing operations*.

INTRODUCTION

L'ordonnancement s'impose depuis longtemps comme une discipline fondamentale de recherche opérationnelle. Cette discipline recouvre un vaste champ d'applications où il s'agit de déterminer l'organisation et la planification de différentes tâches soumises à des contraintes multiples : disponibilité des ressources, séquençage logique, délais de réalisation, etc. Au fil des décennies, les enjeux de l'ordonnancement se sont étendus bien au-delà du secteur industriel, incluant désormais une diversité de domaines comme la logistique, les transports ou les services.

Parmi les nombreux types de problèmes d'ordonnancement, les problèmes d'ordonnancement d'ateliers occupent une place centrale, notamment dans le domaine industriel. Ces problèmes concernent des environnements où les tâches, composées de plusieurs opérations, doivent être traitées sur plusieurs machines ou postes spécialisés. Chaque tâche j , avec $1 \leq j \leq n$, est composée de m opérations notées $O_{1j}, O_{2j}, \dots, O_{mj}$. Une opération O_{ij} correspond à l'exécution de la tâche j par la machine M_i , pour $1 \leq i \leq m$. Chaque machine, considérée comme une ressource disjonctive, ne peut traiter qu'une seule tâche à la fois. En outre, chaque opération d'une tâche ne peut être exécutée qu'une seule fois.

Historiquement, l'ordonnancement d'ateliers a joué un rôle essentiel dans le développement de la production de masse. L'introduction de la chaîne de montage par Henry Ford, au début du XX siècle, a contribué de manière décisive à la formalisation de modèles spécialisés, tels que le *flow shop* ou le *job shop* (DUDEK, PANWALKAR et SMITH, 1992; NAMIKI et al., 1979a), deux catégories majeures de problèmes d'ordonnancement d'ateliers. Dans un *flow shop*, chaque tâche passe sur l'ensemble des machines selon un enchaînement prédéfini, tandis que dans un *job shop*, l'ordre de passage sur les machines peut varier d'une tâche à l'autre. Initialement fondées sur l'optimisation de la production industrielle, ces problématiques ont

toutefois vu leur champ d'application s'étendre, en particulier sous l'effet de l'explosion démographique, et trouvent désormais des utilisations clés dans les services (LIANG et al., 2015b; MARYNISSEN et DEMEULEMEESTER, 2019b; VOGL et al., 2019b). Contrairement à l'industrie, où les tâches représentent souvent des pièces ou des produits, dans les services, elles impliquent fréquemment des flux de personnes. Par exemple, dans le secteur administratif (comme pour le traitement des demandes de visas), il permet d'optimiser le travail des agents tout en réduisant les délais d'attente. L'importance des modèles d'ordonnancement d'ateliers dans les services a été particulièrement mise en évidence lors de la pandémie de COVID-19, où la gestion des flux de patients, l'organisation des campagnes de vaccination et l'optimisation des ressources critiques ont nécessité des approches méthodiques.

Au sein des problèmes de *flow shop*, la notion de permutation a longtemps occupé une place centrale (GMYS et al., 2020; NAWAZ et al., 1983a; RUIZ et MAROTO, 2005a). Dans le *flow shop* de permutation (PFSP), on impose un ordre fixe des tâches sur toutes les machines, ce qui restreint l'espace de recherche mais permet une modélisation plus simple et facilite souvent la mise en œuvre d'heuristiques ou de méthodes exactes. À l'inverse, dans le *flow shop* de non-permutation (NFSP), l'ordre des tâches peut varier d'une machine à l'autre, augmentant la complexité algorithmique tout en offrant une flexibilité potentiellement plus avantageuse (C. LIAO et al., 2006a; LIN et YING, 2009; PUGAZHENDHI et al., 2003a).

Dans de nombreux environnements industriels et, plus encore, dans les services, il est fréquent que certaines tâches ne requièrent pas un passage sur toutes les machines. De telles tâches se caractérisent par l'absence de certaines opérations (dites « opérations manquantes ») sur certaines machines. Par exemple, dans un service hospitalier, un patient peut avoir besoin de réaliser plusieurs examens médicaux alors qu'un autre, atteint d'une pathologie différente, se dirigera directement vers un autre service. À première vue, l'existence d'opérations manquantes pourrait sembler n'avoir qu'une incidence mineure sur la structure du problème, mais

elle soulève en réalité une question centrale : comment définir la permutation dans un atelier où les séquences de traitement peuvent différer d'une machine à l'autre ?

Pendant longtemps, on a intuitivement considéré, dans un contexte de *flow shop* intégrant des opérations manquantes, que la permutation correspondait simplement à un ordre total d'exécution des tâches dans le système. Cependant, d'autres définitions du concept de permutation sont possibles. L'une d'elles repose sur une règle de non-dépassement : si une tâche i précède une tâche j sur une machine, alors i doit également précéder j sur toutes les machines qu'elles ont en commun. Une autre définition, proposée par PINEDO, 2002c, assimile l'ordonnancement par permutation à une organisation respectant une discipline FIFO (First In, First Out) entre les machines. Dans le cas d'un *flow shop* sans opérations manquantes, ces différentes définitions sont équivalentes. En revanche, lorsque des opérations manquantes sont autorisées, elles conduisent à des interprétations divergentes du concept même de permutation, rendant ce dernier ambigu.

La première partie de cette recherche s'intéresse à l'étude des différents concepts de permutation possibles dans le contexte des opérations manquantes. Nous posons mathématiquement leurs divergences et examinons leurs implications sur l'objectif de minimisation du makespan. Cette analyse théorique établit un cadre formel pour comprendre l'impact de ces définitions dans les systèmes de *flow shop* intégrant des opérations manquantes.

Après avoir établi le cadre théorique des concepts de permutation dans le contexte des *flow shops* intégrant des opérations manquantes, la deuxième partie de cette recherche explore une problématique classique liée aux *flow shops* : l'équilibre entre la complexité de résolution et la performance des solutions. Il s'agit de déterminer si la complexité de résolution supplémentaire du *flow shop* de non-permutation se justifie par des gains sur les fonctions objectif. Dans cette étude, nous considérons en particulier le makespan et le temps de fin total.

Dans le contexte des *flow shops* avec opérations manquantes, cette question est particulièrement pertinente. L'absence de certaines opérations sur certaines machines pourrait offrir un avantage aux ordonnancements de non-permutation. Pour évaluer cet avantage potentiel, nous avons identifié une série d'indicateurs reflétant les propriétés structurelles et mathématiques des instances étudiées. L'analyse de ces indicateurs permet alors de déterminer les configurations d'instances où l'adoption d'un modèle de non-permutation se révèle pertinente, et celles pour lesquelles un ordonnancement de permutation reste un bon compromis.

Dans la troisième partie de cette recherche, nous étendons la notion de permutation, initialement étudiée dans le cadre du *flow shop*, à d'autres environnements d'atelier, notamment le *job shop*. La définition de permutation proposée par Pinedo, fondée sur une discipline FIFO pour le transfert des tâches entre les machines, s'avère en effet applicable au-delà du *flow shop*. Sur le plan pratique, l'adoption d'une règle FIFO est particulièrement pertinente dans le secteur des services, car elle renforce la perception d'équité et améliore l'expérience client. Toutefois, dans un *job shop* où l'on cherche à minimiser le makespan, certaines opérations peuvent se voir attribuer un niveau de priorité plus élevé que d'autres. Il arrive alors que ces opérations « passent devant » dans la file d'attente des machines. L'application stricte de la règle FIFO peut provoquer des délais supplémentaires : dans certains cas, on peut être amené à bloquer l'exécution d'opérations moins prioritaires, alors même qu'une machine est disponible, pour que les tâches prioritaires restent en tête de file. Ces retards potentiels, surtout dans un contexte de service, risquent de nuire à la satisfaction des clients lorsque les ressources attendent sans être mobilisées. Dans cette étude, nous modélisons les contraintes FIFO ainsi que l'absence de retard (no-delay) et nous analysons leur impact respectif. Nous évaluons l'effet de la discipline FIFO sur le makespan et sur les délais, ainsi que l'impact de la contrainte no-delay sur le makespan et sur les dépassements de la règle FIFO. Pour cela, nous proposons une modélisation en programmation linéaire en nombres entiers (MIP)

permettant de mieux comprendre les interactions entre ces contraintes dans un contexte de *job shop* adapté au secteur des services.

Les travaux présentés dans cette thèse apportent plusieurs contributions originales à l'étude de l'ordonnancement d'ateliers. Tout d'abord, ils clarifient les différentes définitions possibles de la permutation dans un contexte de *flow shop* intégrant des opérations manquantes, en mettant en lumière les ambiguïtés générées par la divergence de ces définitions. En second lieu, ils examinent les configurations structurelles, notamment celles liées à la présence d'opérations manquantes, afin de déterminer dans quelles conditions les ordonnancements de non-permutation peuvent offrir un avantage significatif. Cette analyse permet ainsi aux décideurs d'orienter leur choix entre un ordonnancement de permutation et un ordonnancement de non-permutation. Enfin, la troisième contribution élargit l'étude de la permutation à d'autres environnements d'ateliers, tels que le *job shop*. Elle propose une modélisation des contraintes majeures rencontrées dans les services, notamment la règle FIFO et l'absence de retard (no-delay), et analyse leur influence sur le makespan.

La composition de cette thèse est structurée comme suit :

- Chapitre 1 : Les notions de base de l'ordonnancement et la complexité des problèmes d'ordonnancement sont présentées.
- Chapitre 2 : Une revue de la littérature se concentre sur les problématiques spécifiques des ateliers, en particulier les *flow shops* et les *job shops*.
- Chapitre 3 : Le premier article analyse l'ambiguïté des définitions de permutation dans les *flow shops* avec opérations manquantes, en posant les bases théoriques nécessaires pour clarifier ces notions.
- Chapitre 4 : Ce chapitre prolonge l'analyse du chapitre 3 en étudiant les propriétés de la notion de permutation selon PINEDO, 2002c dans un *flow shop* avec opérations

manquantes. Par ailleurs, il examine le cas particulier du *flow shop* unitaire avec opérations manquantes. L'accent est mis sur l'évaluation du pire cas généré par quelques heuristiques simples.

- Chapitre 5 : Le deuxième article étudie l'impact des opérations manquantes sur les ordonnancements de permutation et de non-permutation, en identifiant les conditions où chacun de ces modèles est pertinent.
- Chapitre 6 : Ce chapitre est consacré au troisième article, qui explore l'impact des contraintes spécifiques aux services (FIFO et no-delay) sur le makespan dans un problème de *job shop*.
- Chapitre 7 : Enfin, la conclusion synthétise les principales contributions de ce travail, discute ses limites et propose des perspectives de recherche futures.

CHAPITRE I

NOTIONS DE BASE

Dans ce chapitre, nous présentons les fondements théoriques nécessaires à la compréhension de cette thèse. Nous commençons par introduire les notions de base en ordonnancement, en mettant l'accent sur les définitions, la classification et les notations des problèmes. Ensuite, nous abordons les notions clés de la théorie de la complexité, telles que les différentes classes de problèmes et les méthodes de résolution adaptées à chaque type de problème.

1.1 NOTIONS DE BASE EN ORDONNANCEMENT

Dans cette section, nous introduisons les concepts fondamentaux relatifs à l'ordonnancement, notamment sa définition, sa classification et la notation standard.

1.1.1 DÉFINITION DES PROBLÈMES D'ORDONNANCEMENT

L'ordonnancement (ou *scheduling*) consiste à planifier ou programmer dans le temps l'exécution de tâches, en leur attribuant des ressources et en respectant un ensemble de contraintes, dans le but d'optimiser un ou plusieurs critères FRENCH, [1982a](#).

Il est essentiel de distinguer plusieurs types de problèmes d'ordonnancement en fonction des informations disponibles et de la nature des contraintes.

Ordonnancement déterministe Un problème est dit *déterministe* lorsque tous les paramètres durant la planification (durées de tâches, ressources disponibles, dates de début ou de livraison, etc.) sont *a priori* connus et fixes.

Ordonnancement non-déterministe (incertain) Un problème est dit *non-déterministe* ou *incertain* lorsque certaines données ne sont pas connues à l’avance et peuvent varier au cours du temps. Par exemple :

- Les temps de traitement des tâches peuvent être modélisés comme des variables aléatoires suivant une loi de probabilité.
- Les ressources (machines) peuvent devenir indisponibles de façon imprévue (pannes, maintenance, etc.).

Un ordonnancement est couramment visualisé à l’aide de diagrammes appelés *diagrammes de Gantt*. Ces diagrammes permettent de représenter, sur une échelle temporelle, l’occupation des opérations des tâches sur les machines. Chaque machine est associée à une ligne, et les tâches sont représentées par des barres horizontales indiquant leur début et leur fin. Cette visualisation met également en évidence les périodes de temps mort et les indisponibilités des ressources. La Figure 1.1 illustre cette représentation.

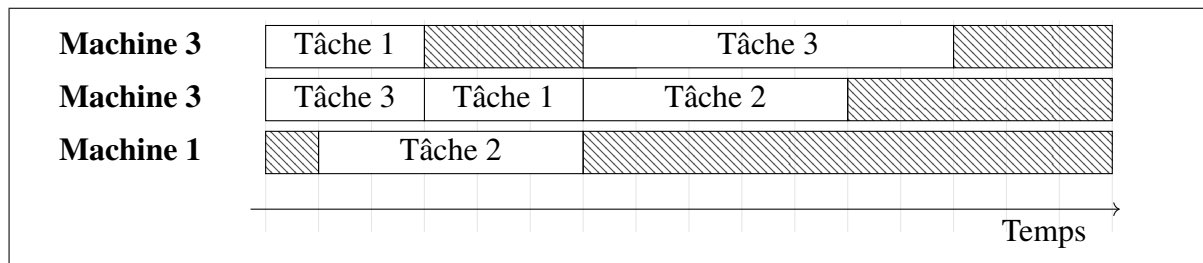


FIGURE 1.1 : Exemple de diagramme de Gantt.

1.1.2 FORMULATION DES PROBLÈMES D'ORDONNANCEMENT

Un problème d’ordonnancement met en jeu quatre paramètres fondamentaux : les tâches, les machines, les contraintes et les objectifs. Dans ce qui suit, nous décrivons chacun de ces paramètres de manière détaillée.

CARACTÉRISATION DES TÂCHES

On considère un ensemble de n tâches, noté $T = \{T_1, \dots, T_n\}$, et un ensemble de m machines, noté $M = \{M_1, \dots, M_m\}$. Chaque tâche T_i peut être décrite par plusieurs données, dont les principales sont :

- **Temps de traitement** p_{ij} : le temps requis par la machine M_i pour exécuter complètement T_j .
- **Date de disponibilité** r_j : la date à partir de laquelle la tâche peut débiter son exécution.
- **Date d'échéance** d_j : la date avant laquelle il est souhaitable que T_j soit achevée.
- **Date limite (absolue)** $\overline{d_j}$: la date avant laquelle T_j doit impérativement être terminée (sinon, la solution est jugée invalide).
- **Poids** w_j : un coefficient exprimant la priorité relative de la tâche (plus w_j est grand, plus la tâche est importante).

Dans certains cas, les tâches sont soumises à des contraintes de précédence, qui imposent un ordre partiel entre elles : une tâche T_j ne peut débiter que si toutes les tâches dont elle dépend directement sont achevées. Ces contraintes modifient la structure du problème en limitant les séquences réalisables et en augmentant la complexité de la résolution.

Les tâches peuvent également être classées en deux catégories selon leur mode d'exécution :

- Les tâches *préemptives* permettent une interruption de leur traitement, avec reprise ultérieure sur la même machine ou sur une autre.
- Les tâches *non-préemptives* exigent une exécution continue : une fois commencées, elles doivent être achevées sans interruption.

Ces distinctions influencent directement les approches de modélisation et les algorithmes de résolution.

CARACTÉRISATION DES MACHINES

Les machines représentent les ressources nécessaires à l'exécution des tâches. Elles jouent un rôle central dans la structure et la classification des problèmes d'ordonnancement. On distingue deux grandes catégories :

1. Machines parallèles

Chaque tâche peut s'exécuter sur une des m machines en parallèle.

- **Machines identiques** (P) : mêmes vitesses de traitement.
- **Machines uniformes** (Q) : vitesses différentes mais liées par un coefficient de proportionnalité.
- **Machines générales** (R) : vitesses arbitraires selon la tâche et la machine.

2. Machines d'atelier (spécialisées)

Chaque tâche T_j est décomposée en un nombre fini d'opérations $(O_{j,1}, O_{j,2}, \dots)$, à exécuter sur différentes machines. Les opérations d'une même tâche ne peuvent pas s'exécuter simultanément, et chaque machine ne peut traiter qu'une seule opération à la fois. Selon le mode de passage entre les machines :

- *Flow Shop* (FSP) : même ordre de passage sur toutes les machines.
Version avec opérations manquantes : FSPMO.
- *Open Shop* (OSP) : ordre libre des opérations pour chaque tâche.
- *Job Shop* (JSP) : ordre spécifique pour chaque tâche.

En outre, il existe des environnements hybrides dits *ateliers flexibles*, combinant machines parallèles et machines d'atelier. Par exemple, dans un flow shop flexible, chaque étape du processus est composée de plusieurs machines parallèles. Les tâches peuvent alors être

affectées à n'importe quelle machine disponible dans une étape donnée, tout en respectant l'ordre séquentiel des étapes.

CARACTÉRISATION DES CONTRAINTES

Les contraintes d'un problème d'ordonnancement encadrent la manière dont les tâches peuvent être planifiées et exécutées. Elles peuvent être principalement de trois natures :

- **Contraintes de ressources** : limitations ou indisponibilités liées aux machines (pannes, maintenance, etc.), types d'ateliers, quantité de ressources disponibles, etc.
- **Contraintes temporelles** : dates de disponibilité, fenêtres de temps, contraintes de précédence, etc.
- **Contraintes technologiques** : découlent directement de la logique de fabrication ou de la nature des opérations à effectuer.

Parmi les contraintes les plus étudiées dans les problèmes de type *flow shop*, figure la contrainte de permutation : le même ordre d'exécution des tâches est imposé sur toutes les machines. Par exemple, si les tâches sont exécutées dans l'ordre Tâche 1, Tâche 2 et Tâche 3 sur la première machine, elles suivront le même ordre sur les suivantes. La Figure 1.2 illustre cette contrainte de permutation dans un diagramme de Gantt avec trois machines et trois tâches.

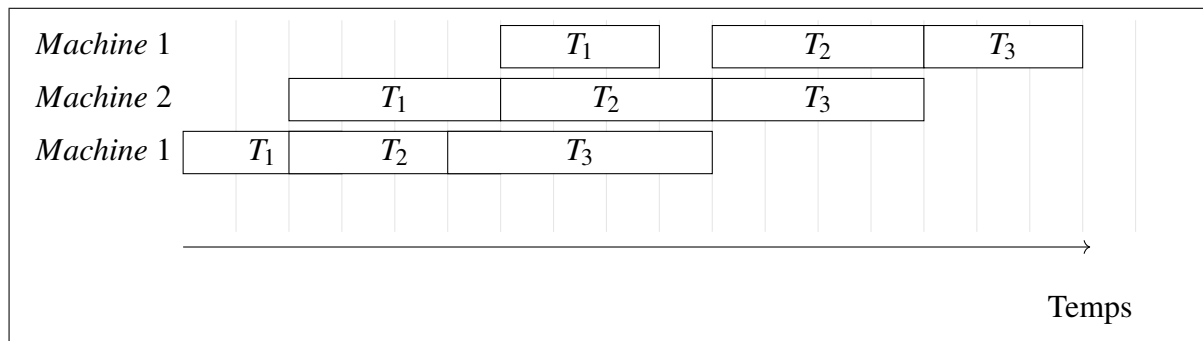


FIGURE 1.2 : Contrainte de permutation dans un flow shop : exécution des tâches dans le même ordre sur toutes les machines

Une autre définition de la permutation, proposée par PINEDO, [2002c](#), assimile un ordonnancement par permutation à une organisation respectant une discipline FIFO (First-In, First-Out) : la première tâche arrivée sur la machine est la première à être traitée.

CARACTÉRISATION DES OBJECTIFS

Pour évaluer la qualité d'un ordonnancement, on utilise des critères à minimiser (ou maximiser), selon les objectifs du problème. Ils se divisent principalement en deux catégories :

1. Critères liés au temps : ces critères mesurent les performances temporelles d'un ordonnancement. Les plus courants incluent :
 - Makespan : La date de fin de la dernière tâche, qui reflète la durée totale de l'ordonnancement.
 - Moyenne (ou somme) des dates d'achèvement des tâches, utilisée pour évaluer le temps moyen nécessaire pour terminer les tâches.
 - Critères liés aux retards : ils incluent la somme des retards ($\sum T_j$), le nombre de tâches en retard, ou encore des pénalités associées aux délais d'exécution dépassés.
2. Critères liés aux ressources ou aux coûts : ces critères visent à optimiser l'utilisation des ressources et à réduire les coûts.

Un critère est dit *régulier* lorsqu'il ne peut être amélioré que par un avancement des dates de fin des tâches, sans remettre en cause leur ordre d'exécution. Plus formellement, si l'on considère deux ordonnancements S et S' , et si, pour chaque tâche, la date d'achèvement dans S' est au plus égale à celle dans S (c'est-à-dire $C_j(S') \leq C_j(S)$ pour toute tâche i), alors la valeur du critère dans S' est nécessairement au moins aussi bonne que dans S .

On distingue par ailleurs trois grandes catégories d'ordonnancements selon leur structure. Un ordonnancement est dit *actif* lorsqu'il n'est pas possible d'avancer une tâche ou une opération sans enfreindre une contrainte. S'il est uniquement impossible d'avancer une tâche sans modifier l'ordre d'exécution ou sans enfreindre les contraintes, on parle alors d'ordonnancement *semi-actif*. Enfin, un ordonnancement est qualifié de *non-delay* (ou sans retard) lorsqu'aucune machine ne reste inutilisée alors qu'une opération est prête à être exécutée. Dans le cas où le critère d'optimisation est régulier, les ordonnancements actifs sont considérés comme dominants.

CLASSIFICATION ET NOTATION

Pour décrire un problème d'ordonnancement de manière concise, la notation $\alpha/\beta/\gamma$ est adoptée (BLAZEWICZ et al., 1983), où :

- α décrit l'environnement machine (par exemple P, Q, R, F, O, J , etc.) et, si nécessaire, le nombre de machines (ex. P_m, F_m , etc.).
- β indique les contraintes spécifiques (tâches préemptives, dates de disponibilité, précédences, etc.).
- γ précise le critère à optimiser ($C_{\max}, \bar{C}$, somme des retards, etc.).

1.2 NOTIONS DE BASE EN COMPLEXITÉ

La théorie de la complexité est une branche de l'informatique théorique qui étudie les ressources (temps, espace, etc.) requises pour résoudre un problème. Elle permet de classer les problèmes selon leur difficulté et de déterminer si leurs solutions peuvent être trouvées de manière efficace (en temps polynomial) ou non. Dans cette section, nous présentons les bases de la théorie de la complexité, y compris les classes de problèmes et un aperçu des différentes méthodes de résolution employées pour aborder ces problèmes.

1.2.1 COMPLEXITÉ ALGORITHMIQUE

La complexité algorithmique mesure les ressources nécessaires à l'exécution d'un algorithme. On distingue principalement l'espace mémoire, dit complexité spatiale, et le temps de calcul, dit complexité temporelle.

Dans ce travail, nous nous intéresserons principalement à la complexité temporelle. Celle-ci évalue le nombre d'opérations (ou le temps de calcul) qu'un algorithme effectue en fonction de la taille de son entrée, notée n . Pour exprimer la complexité temporelle, on utilise généralement les notations asymptotiques (telles que $\mathcal{O}(\cdot)$, $\Omega(\cdot)$, $\Theta(\cdot)$), qui caractérisent le comportement de l'algorithme lorsque n devient grand. Ainsi, on peut comparer l'efficacité de plusieurs algorithmes selon qu'ils s'exécutent en $\mathcal{O}(n)$ (temps linéaire), $\mathcal{O}(p(n))$ (temps polynomial, où p est un polynôme), $\mathcal{O}(a^n)$ (temps exponentiel), etc.

1.2.2 CLASSES DE COMPLEXITÉ

Pour introduire la notion de classes de problèmes, il est tout d'abord nécessaire de distinguer les problèmes de décision des problèmes d'optimisation. Un problème de décision se formule comme une question à laquelle on peut répondre par oui ou par non. Par exemple, déterminer

s'il existe un ordonnancement sur une machine, pour n tâches, dont le makespan est inférieur ou égal à une valeur k , relève d'un problème de décision. Un problème d'optimisation consiste, quant à lui, à trouver une solution qui maximise ou minimise une fonction objectif (f), sous un ensemble de contraintes. Par exemple, rechercher l'ordonnancement d'un ensemble de n tâches sur une machine en vue de minimiser le makespan constitue un problème d'optimisation. Il est d'ailleurs possible d'associer à chaque problème d'optimisation un problème de décision correspondant, en introduisant un seuil k associé à la fonction objectif f . Dès lors, le problème de décision consiste à déterminer s'il existe une solution réalisable S telle que $f(S) \leq k$.

La *théorie de la complexité* propose plusieurs classes permettant de regrouper les problèmes en fonction de leur difficulté de résolution. Parmi ces classes, on distingue notamment :

- **Classe P** : ensemble des problèmes de décision résolus en temps polynomial (par rapport à la taille de l'instance, notée n) par un algorithme déterministe. Autrement dit, il existe une constante k telle qu'un algorithme exécute son calcul en un nombre d'opérations borné par n^k .
- **Classe NP** : ensemble des problèmes de décision pouvant être résolus en temps polynomial par un algorithme non déterministe (c'est-à-dire un algorithme comportant des « instructions de choix »). De manière informelle, un problème appartient à NP s'il existe un moyen de vérifier en temps polynomial qu'une réponse oui (ou solution candidate) est correcte, même si la recherche d'une telle solution peut, dans le pire des cas, exiger un temps exponentiel.

Il est clair que tout problème appartenant à P appartient également à NP , puisqu'une solution trouvée en temps polynomial peut trivialement être vérifiée en temps polynomial.

Parmi l'ensemble des problèmes de NP , on distingue une sous-classe notée **classe des problèmes NP -complets**, considérés comme les plus difficiles de la classe NP . Le concept de NP -complétude s'appuie sur la notion de réduction en temps polynomial.

Définition 1.2.1 (Réduction en temps polynomial). *Soient deux problèmes de décision A et B . On dit que A se réduit en temps polynomial à B (noté $A \leq_p B$) s'il existe une fonction f , calculable en temps polynomial, telle que, pour tout élément x , on ait*

$$x \in A \iff f(x) \in B.$$

Formellement, un problème est défini comme étant NP -complet si et seulement si :

1. Le problème Q appartient à NP .
2. Tout problème Q' de NP se réduit en temps polynomial à Q . Autrement dit, $\forall Q' \in NP$, on a $Q' \leq_p Q$.

Un problème NP -complet est donc un noyau de NP , dans le sens où la résolution polynomiale d'un seul de ces problèmes impliquerait la résolution polynomiale de tous les problèmes de NP . La démarche standard pour établir que Q est NP -complet comporte quatre étapes :

1. Montrer que Q appartient à NP .
2. Choisir un problème NP -complet connu, noté Q' .
3. Construire une réduction f de Q' vers Q .
4. Prouver que f s'effectue en *temps polynomial*.

La démonstration qu'un problème de décision appartient à la classe des NP -complets présente une importance fondamentale. Une fois cette appartenance établie, il est généralement admis qu'il est peu probable qu'un algorithme polynomial puisse être conçu pour résoudre ce problème de manière générale. Une liste exhaustive des problèmes NP -complets est disponible

dans GAREY et al., 1976a. Par ailleurs, un problème d'optimisation Q_{opt} est qualifié de NP -difficile si le problème de décision qui lui est associé, noté Q_{dec} , est démontré NP -complet.

La classe des problèmes NP -complets peut être divisée en deux sous-classes : la NP -complétude au sens fort et la NP -complétude au sens faible. Cette distinction repose sur la notion de transformation pseudo-polynomiale et la manière dont la complexité dépend des valeurs numériques des instances.

Un *algorithme pseudo-polynomial* est un algorithme dont la complexité est polynomiale en la valeur numérique des données de l'instance, mais potentiellement exponentielle en la taille binaire de celle-ci. Autrement dit, si l'on note $|x|_{\text{bin}}$ la longueur (en bits) de l'instance et $|x|_{\text{max}}$ la valeur numérique maximale qu'elle contient, le temps d'exécution de l'algorithme s'exprime comme un polynôme en $|x|_{\text{max}}$ et $|x|_{\text{bin}}$.

Un problème est dit *faiblement NP -complet* s'il admet un algorithme pseudo-polynomial. Dans ce cas, le temps d'exécution dépend à la fois de $|x|_{\text{max}}$ et de $|x|_{\text{bin}}$.

Un problème est *fortement NP -complet* s'il reste NP -complet même lorsque toutes les valeurs numériques de l'instance sont bornées par un polynôme de sa taille binaire. Ainsi, il n'existe pas d'algorithme pseudo-polynomial pour le résoudre, sauf si $P = NP$.

Pour plus de détails sur la NP -complétude et ses sous-classes, le lecteur pourra se référer à GAREY et al., 1976a.

Cette classification a des conséquences directes :

- Les problèmes faiblement NP -complets peuvent être abordés à l'aide d'algorithmes pseudo-polynomiaux si les données numériques sont restreintes.
- Les problèmes fortement NP -complets requièrent généralement des méthodes approchées ou des heuristiques.

1.3 APPROCHES DE RÉOLUTION DES PROBLÈMES D'ORDONNANCEMENT

Les problèmes d'ordonnancement, qui relèvent de l'optimisation combinatoire, peuvent être résolus à l'aide de différentes stratégies. De manière générale, on distingue deux grandes catégories d'approches :

- **Les méthodes exactes**
- **Les méthodes approchées**

1.3.1 MÉTHODES EXACTES

Les méthodes exactes permettent d'obtenir une solution optimale. Leur mise en œuvre exige toutefois souvent un temps de calcul important, ce qui restreint leur utilisation aux problèmes de taille relativement modeste. Parmi les techniques exactes les plus courantes, citons :

1. Programmation dynamique

Introduite par BELLMAN, 1966, elle consiste à décomposer le problème en sous-problèmes de taille plus réduite. Chaque sous-problème est résolu de manière récursive, et les solutions partielles sont ensuite combinées pour obtenir la solution optimale du problème initial.

2. Méthodes arborescentes de séparation et évaluation (Branch and Bound)

Cette approche procède par une exploration structurée de l'ensemble des solutions réalisables :

- *Séparation (Branch)* : on subdivise l'espace des solutions en plusieurs sous-espaces, formant ainsi une arborescence où chaque nœud représente un sous-problème.
- *Évaluation (Bound)* : on calcule une borne (une borne inférieure pour un problème de minimisation) pour chaque sous-problème. Si cette borne atteint la valeur de

la meilleure solution déjà trouvée, on peut stériliser le nœud associé au sous-problème.

3. Programmation par contraintes

Ici, le problème est formulé comme un ensemble de contraintes logiques liant des variables, chacune possédant un domaine de valeurs (fini ou continu). L'objectif est de déterminer s'il existe une affectation des variables satisfaisant simultanément toutes les contraintes.

4. Programmation mathématique

Le problème est modélisé sous forme de programme linéaire ou non linéaire, éventuellement avec des variables entières ou mixtes. Des solveurs spécialisés peuvent alors être employés pour trouver la solution optimale et prouver son optimalité. La programmation mathématique englobe différentes techniques :

- **Génération de colonnes** DANTZIG et WOLFE, 1960 : on démarre avec un jeu réduit de variables (colonnes), puis on résout un sous-problème qui permet d'introduire de nouvelles colonnes susceptibles d'améliorer la solution courante.
- **Méthodes de coupes** WOLSEY, 2020 : on résout la relaxation linéaire du modèle tout en ajoutant progressivement des inégalités valides (coupes) afin de restreindre l'espace de recherche du problème relaxé tout en préservant le domaine de réalisabilité initial.

1.3.2 MÉTHODES APPROCHÉES

Pour les problèmes de grande taille, le recours aux méthodes exactes peut devenir trop coûteux en temps de calcul. On se tourne alors vers des méthodes approchées, qui proposent une solution en un temps raisonnable, sans pour autant garantir l'optimalité. Parmi ces méthodes, on distingue principalement :

1. Heuristiques constructives

Elles construisent une solution de manière incrémentale, sans retour en arrière. Les heuristiques « gloutonnes » (ou « voraces ») en sont un exemple : à chaque étape, elles choisissent l'option localement la plus prometteuse. Bien que rapides, elles peuvent mener à des solutions de qualité variable selon le problème.

2. Méthodes de recherche locale

Partant d'une solution initiale, on l'améliore progressivement par l'exploration de solutions voisines. Le processus s'arrête dès qu'aucune amélioration n'est trouvée. Le principal inconvénient de ces méthodes est la possibilité de rester bloqué dans un *minimum local*.

3. Métaheuristiques

Ces approches visent à dépasser les limites de la recherche locale en autorisant, ponctuellement, des mouvements qui dégradent la solution afin d'éviter l'enfermement dans des minima locaux. Les plus courantes incluent :

- le recuit simulé (*Simulated Annealing*),
- la recherche avec tabou (*Tabu Search*),
- les algorithmes génétiques (*Genetic Algorithms*),
- les colonies de fourmis (*Ant Colony Optimization*),
- l'optimisation par essaims particuliers (*Particle Swarm Optimization*), etc.

Généralement génériques, ces méthodes s'adaptent à différents types de problèmes (dont l'ordonnancement) en modifiant l'espace de recherche ou les opérateurs de voisinage.

4. Algorithmes d'approximation

Contrairement aux heuristiques, qui ne fournissent pas nécessairement de garanties sur la qualité de la solution, les algorithmes d'approximation proposent souvent un *ratio*

de performance mesurant jusqu'à quel point la solution obtenue peut s'éloigner de l'optimum.

Définition 1.3.1 (Algorithme d'approximation). *Un algorithme d'approximation pour un problème d'optimisation (par exemple de minimisation) s'exécute en temps polynomial et, pour toute instance I , produit une solution réalisable S_I dont la valeur $C(S_I)$ ne dépasse qu'un certain facteur α de la valeur optimale $C(S_I^*)$. On dit qu'il s'agit d'un algorithme α -approché si, pour tout I :*

$$C(S_I) \leq \alpha \cdot C(S_I^*),$$

avec $\alpha \geq 1$. Dans le cas d'un problème de maximisation, la condition devient

$$C(S_I) \geq \frac{1}{\alpha} \cdot C(S_I^*).$$

La valeur α caractérise ainsi le « pire écart » possible entre la solution approchée et la solution optimale.

Ratios d'approximation. Les garanties offertes par ces algorithmes s'expriment via plusieurs types de *ratios* :

— **Rapport absolu (minimisation) :**

$$\rho = \sup_I \left\{ \frac{C(S_I)}{C(S_I^*)} \right\},$$

où S_I est la solution construite et S_I^* la solution optimale.

— **Rapport asymptotique :**

$$\rho_{\text{asympt}} = \limsup_{C(S_I^*) \rightarrow +\infty} \left(\frac{C(S_I)}{C(S_I^*)} \right).$$

— **Schéma d'approximation (PTAS) :**

$$\forall \varepsilon > 0, \rho(\varepsilon) = 1 + \varepsilon \quad \text{avec un temps de calcul } O\left(n^{g(\varepsilon)}\right),$$

où $g(\varepsilon)$ croît en fonction de $1/\varepsilon$. Un *PTAS* fournit donc une solution $(1 + \varepsilon)$ -près de l'optimum, au prix d'un temps polynomial dépendant de ε .

CHAPITRE II

REVUE DE LA LITTÉRATURE

Dans cette section, nous présentons une revue de la littérature consacrée aux problèmes d'atelier, en particulier les problèmes de type *flow shop* et *job shop*. Nous mettons d'abord l'accent sur leurs applications dans le secteur des services, puis nous abordons les variantes classiques du *flow shop* (permutation et non-permutation). Nous examinons ensuite les travaux portant sur les problèmes de *flow shop* avec opérations manquantes. Enfin, nous concluons en évoquant les différentes études consacrées aux problèmes de *job shop*.

2.1 APPLICATIONS INDUSTRIELLES DES PROBLÈMES D'ATELIER : DES CHAÎNES DE MONTAGE À L'INDUSTRIE 4.0

Depuis le milieu du XX^e siècle, le *flow shop* s'est imposé comme un modèle clé dans la production industrielle, notamment pour les lignes d'assemblage en série, où chaque produit suit une séquence précise d'opérations successives (NAMIKI et al., [1979a](#)). Par la suite, ce modèle a été adopté dans d'autres secteurs, comme la chimie et la pharmacie (DUDEK, PANWALKAR et SMITH, [1992](#)), ainsi que dans l'industrie papetière, pour structurer les étapes allant de la fabrication de la pâte à papier jusqu'au produit fini (CORWIN et ESGOBUE, [1974b](#)).

Cependant, le *flow shop*, avec sa séquence uniforme d'opérations sur toutes les machines, manque souvent de flexibilité dans les contextes industriels où la diversité des produits et des processus est élevée. En revanche, le *job shop* s'adapte davantage à ces environnements en autorisant chaque tâche à suivre un itinéraire spécifique à travers les machines. Ce modèle s'avère particulièrement adapté à des secteurs variés, tels que l'aérospatiale (JIANG et al.,

2022b), la maintenance industrielle (WANG et YU, 2010b), ainsi que l'industrie chimique et pharmaceutique (RUIZ-TORRES et al., 2012b).

Avec l'émergence de l'industrie 4.0, ces approches doivent s'adapter à de nouveaux défis. Par exemple, des recherches ont exploré le *flow shop* multi-objectifs, appliqué à des machines intelligentes avancées dans des systèmes de production reposant sur les principes de l'industrie 4.0 (Y. FU et al., 2018b). D'autres travaux se sont concentrés sur le problème d'ordonnancement du *job shop* dans les systèmes de fabrication intelligents, en intégrant des technologies telles que les jumeaux numériques (SERRANO-RUIZ et al., 2022b).

2.2 APPLICATIONS DES PROBLÈMES D'ATELIER DANS LES SERVICES

Au-delà du secteur industriel, l'ordonnancement d'atelier s'est révélé particulièrement pertinent dans les services, notamment dans le domaine de la santé. En effet, les problèmes de planification de rendez-vous (*multi-appointment scheduling problems*) en milieu hospitalier peuvent être modélisés sous forme de problèmes d'atelier, où chaque patient représente un *job* se déplaçant à travers une série de services médicaux (assimilés à des machines dans la terminologie classique). Cette approche permet, entre autres, de prendre en compte les contraintes de ressources limitées (salles, équipements, personnel), tout en réduisant les temps d'attente des patients et en améliorant la qualité de service (LIANG et al., 2015b; MARYNISSEN et DEMEULEMEESTER, 2019b; VOGL et al., 2019b). Dans ce contexte, les travaux rapportés dans la littérature ont exploré diverses méthodologies, allant des heuristiques et métaheuristiques (BURDETT et KOZAN, 2018a; KAZEMIAN et al., 2017a; REZAEIAHARI et KHASAWNEH, 2017a) à l'optimisation stochastique (LEEFTINK et al., 2019a; ZHOU et YUE, 2019a), en passant par des techniques de génération de colonnes et de relaxation lagrangienne (DIAMANT et al., 2018a; XIAO et al., 2022a).

Les problèmes de planification de rendez-vous multiples peuvent être formalisés et étudiés à travers des modèles issus des problèmes d’atelier. Dans LIANG et al., 2015b, un modèle de *flow shop* a été mobilisé pour optimiser la planification dans des cliniques d’oncologie, l’objectif étant de minimiser les temps d’attente et de gérer efficacement les ressources disponibles. De même, VOGL et al., 2019b s’est intéressé à un problème de type *job shop* élargi pour la planification de séances de radiothérapie, en intégrant des contraintes spécifiques liées au faisceau de particules, ressource critique dont l’exploitation doit être optimisée. Les auteurs ont proposé un modèle en nombres entiers et y ont associé des approches métaheuristiques (combinaison d’algorithmes génétiques et de recherche locale itérative) afin de réduire les temps de calcul et de s’approcher au mieux de la solution optimale. Les fonctions objectifs considérées dans ces travaux varient grandement selon le contexte hospitalier : minimisation des temps d’attente, maximisation du nombre de patients traités, réduction des coûts, ou encore prise en compte d’aspects éthiques et d’équité (ALA et al., 2022a; SAADANI et al., 2014a).

2.3 FLOW SHOP

Le problème classique de *flow shop*, abrégé FSP, peut être décrit comme suit. On considère un ensemble de tâches $J = \{J_1, J_2, \dots, J_n\}$ où n est le nombre de tâches, et un ensemble de m machines, avec $m > 1$. Chaque tâche doit être exécutée sur les machines dans le même ordre, c’est-à-dire machine 1, machine 2, et ainsi de suite jusqu’à la machine m . Ainsi, chaque tâche k est composée de m opérations, notées $O_{k1}, O_{k2}, \dots, O_{km}$. La durée de traitement de l’opération O_{ij} est notée $p_{ij} \in \mathbb{N}$. L’objectif est de déterminer un ordonnancement des n tâches qui minimise le critère étudié.

Le problème de *flow shop* à deux machines ($F2 \parallel C_{\max}$) est le seul problème de *flow shop* avec le critère du makespan qui soit résoluble en temps polynomial lorsque les durées de traitement sont arbitraires (JOHNSON, 1954a). En général, les problèmes de *flow shop*

sont difficiles à résoudre. Par exemple, le problème $F_m \parallel C_{\max}$ est fortement NP-difficile pour $m > 2$ (GAREY et al., 1976a). Cependant, certains cas particuliers peuvent être résolus en temps polynomial en imposant des contraintes spécifiques sur les durées de traitement. Par exemple, JOHNSON, 1954a montre que le problème $F_3 \parallel C_{\max}$ peut être résolu lorsque

$$\max(p_{i2}) \leq \max(\min(p_{i1}), \min(p_{i3})).$$

Pour le problème de *flow shop* avec la minimisation de la somme des temps de traitement, noté $F_m \parallel \bar{C}$, il est fortement NP-difficile lorsque les durées de traitement sont arbitraires, même pour $m = 2$ (GAREY et al., 1976a).

La littérature distingue deux grandes familles de problèmes de *flow shop* :

1. **Flow shop de permutation (PFS)** : la séquence des tâches reste identique sur toutes les machines.
2. **Flow shop de non-permutation (NPFS)** : la séquence des tâches peut varier d'une machine à l'autre.

Dans ce qui suit, nous présentons les deux principaux types de problèmes de *flow shop* : ordonnancement de permutation et ordonnancement de non-permutation. Nous explorons également les travaux relatifs aux différentes méthodes de résolution (exactes et approchées), en mettant l'accent sur les objectifs du makespan et sur le critère de la moyenne des dates d'achèvement.

2.3.1 FLOW SHOP DE PERMUTATION

Le *flow shop* de permutation (PFSP) impose que la séquence des jobs soit identique sur toutes les machines. Bien que ce problème soit NP-difficile pour $m \geq 3$ (GAREY et al., 1976a),

il bénéficie d'une riche littérature en raison de sa structure particulière. En effet, la contrainte imposant la même séquence sur chaque machine a fortement orienté la recherche vers des méthodes exactes, heuristiques et métaheuristiques dédiées.

Les approches exactes sont principalement limitées aux petites instances. L'algorithme de JOHNSON, [1954a](#) permet de résoudre optimalement le cas à deux machines en $\mathcal{O}(n \log n)$.

Les premières approches de type *branch-and-bound* remontent à IGNALL et SCHRAGE, [1965b](#), qui se sont intéressés à la minimisation de la moyenne des dates d'achèvement dans un *flow shop* à deux machines. LOMNICKI, [1965](#) a appliqué cette méthodologie pour minimiser le makespan dans un *flow shop* à trois machines, en se limitant aux ordonnancements de permutation (dominants pour $m = 3$). GRABOWSKI, [1980](#) a étudié le *flow shop* à deux machines avec des dates de disponibilité pour les tâches, visant à minimiser le retard. LADHARI et HAOUARI, [2005a](#) ont développé un algorithme *branch-and-bound* renforcé par de nouvelles bornes inférieures, conçu pour la minimisation du makespan. Cet algorithme permet de résoudre optimalement des instances allant jusqu'à 200 tâches et 10 machines (soit 2000 opérations) ou encore 2000 tâches et 4 machines (8000 opérations) pour certaines instances aléatoires. Plus récemment, GMYS et al., [2020](#) ont proposé un algorithme visant à minimiser le critère du makespan. Celui-ci offre plusieurs options pour les schémas de branchement et permet d'explorer l'arbre de recherche soit de manière séquentielle, soit en parallèle sur des processeurs multi-cœurs. Les auteurs montrent que la recherche parallèle joue un rôle clé dans la résolution d'instances de grande taille. Enfin, des formulations en programmation mathématique, notamment les modèles MILP, sont également utilisées pour des instances de taille limitée (STAFFORD, [1988](#)). Cependant, ces approches deviennent rapidement inviables à grande échelle du fait de leur complexité computationnelle.

Étant donné la complexité du *flow shop* de permutation, de nombreuses recherches se sont tournées vers des méthodes approchées, notamment les heuristiques et métaheuristiques. Parmi ces heuristiques, l'algorithme NEH (NAWAZ et al., 1983a) est considéré comme une référence en raison de sa simplicité et de son efficacité. Son principe est le suivant :

1. Tri des jobs par temps total décroissant,
2. Insertion gloutonne itérative,
3. Affinement par échanges locaux.

La complexité de cet algorithme est de l'ordre de $\mathcal{O}(n^3m)$. Cependant, E. TAILLARD, 1990 a montré qu'il était possible de réduire cette complexité à $\mathcal{O}(n^2m)$.

De nombreuses variantes de NEH ont également été proposées, tirant parti de techniques de recherche locale ou de perturbation pour améliorer la qualité des solutions (DONG et al., 2008 ; W. LIU et al., 2017).

D'autres heuristiques se concentrent sur la minimisation de la moyenne des dates d'achèvement (BAI et REN, 2013 ; DELLA CROCE et al., 1996 ; LAHA et CHAKRAVORTY, 2011). Les métaheuristiques incluent le recuit simulé (GLASS et POTTS, 1996), la recherche taboue (OSMAN, 1995), les algorithmes génétiques (CHEN et al., 1996 ; ETILER et al., 2004 ; MURATA et al., 1996), l'optimisation par colonies de fourmis (YAGMAHAN et YENISEY, 2010 ; YING et LIAO, 2004), les algorithmes par essaims particuliers (B. LIU et al., 2007), ainsi que des approches plus récentes comme les lucioles (MARICHELVAM et GEETHA, 2016 ; SAYADI et al., 2010) et la recherche coucou (ZHANG et al., 2020). Certaines stratégies parallélisées (MOU et al., 2022) ont été développées pour accélérer la résolution.

2.3.2 FLOW SHOP DE NON-PERMUTATION

La variante de non-permutation (NPFS) offre une plus grande flexibilité, permettant un ordre de tâches différent sur chaque machine. Alors que le *flow shop* de permutation avec n tâches génère au maximum $n!$ séquences possibles, un *flow shop* général avec m machines peut en produire jusqu'à $(n!)^m$. Cette explosion combinatoire rend la résolution du NPFS beaucoup plus exigeante en calcul.

Plusieurs études mettent en évidence l'intérêt théorique (POTTS et al., 1991b) et pratique (PUGAZHENDHI et al., 2003a) dans les environnements industriels de cette variante. POTTS et al., 1991b ont démontré qu'un ordonnancement par permutation peut être jusqu'à un facteur de $\sqrt{m/2}$ moins performant qu'un ordonnancement optimal sans permutation. De leur côté, C. LIAO et al., 2006a indiquent que la résolution de problèmes d'ordonnancement en *flow shop* visant à minimiser le retard total (*total tardiness*) avec une approche PFS entraîne une perte d'efficacité d'environ 10 % par rapport aux solutions obtenues avec une approche NPFS. En revanche, pour les fonctions objectifs basées sur le makespan, les améliorations observées restent limitées, ne dépassant pas 1 %. Par ailleurs, LIN et YING, 2009 montrent que dans des systèmes de type *flow shop* organisés en cellules de fabrication, pour des objectifs relatifs aux dates d'échéance, les gains d'efficacité obtenus grâce aux plannings NPFS peuvent atteindre jusqu'à 30 %, et se situent en moyenne autour de 10 %. Concernant les fonctions objectif liées aux dates de fin d'exécution, les gains se situent plutôt entre 5 % et 6 %.

Grâce aux progrès en matière de puissance de calcul et d'algorithmes efficaces, l'intérêt pour les problèmes d'atelier de *flow shop* de non-permutation s'est intensifié ces dernières années. Parmi les contributions récentes, citons les travaux de BENAVIDES et RITT, 2018a, qui proposent une heuristique constructive basée sur l'insertion, similaire à l'heuristique NEH (NAWAZ et al., 1983a), ainsi que ceux de BRUM et al., 2022a, qui ont mis en œuvre des

algorithmes gloutons itératifs et une configuration automatique afin de minimiser le temps de complétion total. Pour une revue exhaustive de la littérature, on pourra consulter ROSSIT et al., [2018a](#).

2.3.3 FLOW SHOP EN PRÉSENCE D'OPÉRATIONS MANQUANTES

Il existe deux façons distinctes d'interpréter une opération manquante. La première, telle qu'utilisée par POTTS et al., [1991b](#), la considère comme une opération de durée négligeable mais non nulle, exigeant que chaque tâche soit traitée sur toutes les machines. La seconde interprétation définit l'opération manquante comme ayant un temps de traitement nul, autorisant ainsi certaines tâches à sauter certaines machines. Sur la base de cette seconde interprétation, diverses approches ont été proposées ; par exemple, un algorithme de recuit simulé visant à minimiser le temps total d'achèvement est proposé dans SRIDHAR et RAJENDRAN, [1993a](#). LEISTEN et KOLBE, [1998a](#) suggèrent une relaxation de la permutation, en développant un algorithme qui inclut une procédure d'énumération pour générer des permutations partielles. SADJADI et al., [2008a](#) et RAMEZANIAN, SAIDI-MEHRABAD et RAHMANI, [2011a](#) présentent des modèles de programmation linéaire en nombres entiers mixtes pour le FSMOP. HENNEBERG et NEUFELD, [2016a](#) ont proposé une version améliorée de l'heuristique de PUGAZHENDHI et al., [2003a](#) pour des problèmes de type FSMOP, en utilisant un algorithme de recuit simulé en deux étapes afin d'optimiser le temps total d'achèvement. Plus récemment, RITT et ROSSIT, [2024b](#) ont introduit une nouvelle heuristique qui opère en deux phases : la première détermine une solution de permutation de haute qualité, tandis que la seconde autorise un ordonnancement de non-permutation pour améliorer la solution. Cette approche en deux phases a démontré de meilleurs résultats que ceux de HENNEBERG et NEUFELD, [2016a](#) pour la minimisation du temps total d'achèvement dans des environnements de *flow shop* avec opérations manquantes.

Dans le cadre de l’ordonnancement de type *flow shop* de permutation avec opérations manquantes, la définition classique d’un *flow shop*, selon laquelle « l’ordre des tâches est le même sur toutes les machines », ne s’applique pas totalement. En effet, avec des opérations manquantes, toutes les machines ne traitent pas nécessairement le même ensemble de tâches. D’autres définitions alternatives ont ainsi émergé dans la littérature. OUCHENE et al., 2023b ont proposé différentes manières de définir une permutation comme un ordre total (une chaîne) sur l’ensemble des tâches. La seconde définition impose une condition de non-dépassement (*no-overtaking*) entre les tâches : si une tâche précède une autre sur une machine, cet ordre doit être respecté sur toutes les machines. Dans la troisième définition, PINEDO, 2002c définissent une permutation au regard des machines de telle sorte que les listes de tâches en attente de traitement sur lesdites machines respectent la règle FIFO (*First In First Out*), signifiant qu’une tâche ne peut pas en dépasser une autre dans la file d’attente.

2.4 JOB SHOP

Contrairement au *flow shop*, dans le problème de *job shop*, chaque tâche (*job*) possède sa propre route technologique, distincte des autres. Chaque opération doit ainsi être exécutée dans un ordre spécifique, et les tâches se partagent l’ensemble des machines. Cette flexibilité supplémentaire rend le problème de *job shop* plus complexe encore que celui de *flow shop*.

De nombreuses approches exactes et heuristiques ont été proposées pour la résolution des problèmes de *job shop*. Parmi les méthodes exactes, la programmation linéaire en nombres entiers (MILP) est couramment employée. Trois grandes formulations MIP se distinguent dans la littérature :

- la formulation indexée en temps (*time-indexed formulation*) (BOWMAN, 1959a),
- la formulation basée sur le rang (*rank-based formulation*) (WAGNER, 1959a),
- et la formulation disjonctive (*disjunctive formulation*) (MANNE, 1960b).

Chacune propose une manière différente de représenter la séquence des opérations et la gestion des ressources. Une comparaison théorique (C.-H. PAN, 1997a) montre que la formulation disjonctive est la plus performante, tandis que la formulation *time-indexed* se révèle la moins efficace. Les tests expérimentaux réalisés dans KU et BECK, 2016a confirment que le modèle disjonctif obtient les meilleurs résultats pour la minimisation du makespan. Par ailleurs, d'autres techniques exactes, comme la relaxation lagrangienne (pour minimiser le retard total (YAN et al., 2018a)) ou les méthodes *branch-and-cut* (fondées sur la formulation disjonctive de APPLEGATE et COOK, 1991c), ont également été explorées.

En raison de la complexité intrinsèque des grandes instances, les méthodes exactes peinent à fournir des solutions en un temps raisonnable. La littérature se tourne alors vers des approches heuristiques ou métaheuristiques. Les heuristiques de priorités (règle du plus court temps de traitement, premier arrivé–premier servi, « Most Work Remaining », etc.) ont été largement étudiées (CHIANG et FU, 2007a; HAUPT, 1989b; NARASSIMA et al., 2022a). Les métaheuristiques comptent parmi les approches les plus efficaces pour améliorer la qualité des solutions : on peut citer la recherche taboue (DELL'AMICO et TRUBIAN, 1993a), le recuit simulé (KRISHNA et al., 1995a) et les algorithmes évolutionnaires (SALIDO et al., 2016a). Plus récemment, des algorithmes hybrides ont vu le jour (ALKHATEEB et al., 2022a; FONTES et al., 2023a), combinant divers mécanismes (exact/approché, heuristique/métaheuristique) pour mieux traiter des problèmes de grande dimension.

CHAPITRE III

AMBIGUITY OF THE DEFINITION OF PERMUTATION FLOW SHOPS IN THE PRESENCE OF MISSING OPERATIONS

Le présent chapitre a été publié sous forme d'article de revue depuis le 29 juin 2023, selon la référence suivante :

Ouchene, R., Rebaine, D. & Baptiste, P. (2023). *Ambiguity of the definition of permutation flow shops in the presence of missing operations*. **Computers & Industrial Engineering**, 182, 109387.

Disponible en ligne : <https://doi.org/10.1016/j.cie.2023.109387>

ABSTRACT

The majority of flow shop studies focused so far on permutation schedules, even though non-permutation schedules might offer better results at the expense however of additional computational effort. It is therefore crucial to determine whether the use of non-permutation schedules is justified by the potential improvements that might be obtained. This question becomes particularly interesting in the case of the flow shop problem with missing operations. But, before investigating this question, we must first clarify the definition of a permutation within this context. The objective of this paper consists in bringing out the ambiguity of the definition of a permutation within the flow shop model with missing operations. To overcome this ambiguity, we propose another way of looking at a permutation that includes the classical understanding. Even though, these two definitions may produce different solutions, we present cases where they are equivalent with respect to the makespan criterion.

keywords : Flow shop ; permutation ; missing operations ; makespan.

3.1 INTRODUCTION

The classical flow shop problem, FSP for short, can be described as follows. Given are a set $J = \{J_1, J_2, \dots, J_n\}$ of n jobs and $m > 1$ machines. Each job has to be processed on the machines in the same order, say machine 1, machine 2, and so on until it completes its processing on a machine m . Thus, each job, say k , comprises m operations, $O_{k1}, \dots, O_{km}$. The processing time of operation O_{ij} is denoted by $p_{ij} \in \mathbb{N}$. We seek a schedule of the n jobs that minimizes the overall completion time, also known as the makespan. The FSP is solvable in $O(n \log n)$ for $m = 2$ JOHNSON, [1954b](#), and strongly $\mathcal{NP}$ -hard for $m \geq 3$ GAREY et al., [1976c](#).

So far two main models of a flow shop have been distinguished in the literature *viz.* the permutation flow shop, without over-passing, where the job sequence is the same over the machines, and the flow shop with over-passing where the sequence of jobs can change on each machine, still having the same processing order over the machines. The possibility of varying the sequences on the machines leads to a tremendous enlargement of the feasible

solutions space. Indeed, in a flow shop with permutation, one seeks an optimal solution among $O(n!)$ feasible schedules. Whereas, in a flow shop, the search space is $O((n!)^m)$. Let us however observe that it is a common practice in the literature to focus on permutation schedules when studying the flow shop models, as discussed in ROSSIT et al., 2018b, even though this assumption is only true for regular criteria and $m \leq 3$. In the many papers that focused on permutation schedules, the approximate approach is by far the most used. There are indeed several constructive heuristics in the literature for the permutation flow shop; for an extensive review, see RUIZ et MAROTO, 2005b. There are also many articles that propose meta-heuristic methods as a solving approach; see e.g. AHMADIZAR, 2012, and HSU et al., 2015.

Nevertheless, non-permutation schedules can lead to significantly better results for various criteria C. LIAO et al., 2006b. Due to the rise in computing power and the development of efficient algorithms, the interest in studying non-permutation flow shop problems has increased over the years. Among the recent works, we may cite BENAVIDES et RITT, 2018b where three new heuristics for the problem are designed, including a constructive heuristic with local insertion search without permutation and two iterated greedy algorithms for the flow shop problem with permutation and without permutation. The authors show, through an experimental study, the proposed method produces better non-permutation schedules than other existing methods while using the same computation time. A new model for designing iterated greedy algorithms is proposed, using an automated approach as discussed in BRUM et al., 2022b.

In many industrial problems, for various technological reasons, jobs may have missing operations. In other words, these jobs may not require processing on all machines. In the literature, there are two different interpretations of defining a missing operation.

According to the first interpretation, an operation with a non-zero but negligible duration and can therefore be ignored is considered as a missing operation (each job must still be processed on each machine). With this assumption, POTTS et al., 1991c exhibit a family of instances for which the best permutation schedule is lower than that of an optimal schedule without the permutation constraint by a factor no greater than $\sqrt{m}/2$.

On the other hand, the second interpretation considers that a missing operation is an operation with a zero processing time (jobs may not pass through all the machines). With this assumption, SRIDHAR et RAJENDRAN, 1993b propose a simulated annealing (SA) algorithm with the objective of minimizing the sum of completion times for flow shop with missing operations, FSMOP for short. LEISTEN et KOLBE, 1998b propose a relaxation of the permutation. They develop an algorithm that contains an enumeration procedure to generate partial permutations. SADJADI et al., 2008b and RAMEZANIAN, SAIDI et RAHMANI, 2011 present mixed integer programming for FSMOP. HENNEBERG et NEUFELD, 2016b proposed for FSMOP problems an improved version of the heuristic of PUGAZHENDHI et al., 2003b, along with a two-step simulated annealing algorithm.

The concept of the permutation, where the sequence of jobs is the same on all the machines, is not clear for the flow shop with missing operations, in case the second definition of a missing operation is used. Indeed, in this case, the list of jobs varies on each machine, hence the need to resort to a new definition of a permutation in the FSMOP problem. After pointing out the reason of this ambiguity, two ways of defining a permutation, within the context of flow shop scheduling with missing operations, have almost naturally emerged. We then show that the two definitions do not necessarily generate equivalent job sequences, neither they produce the same optimal makespan.

In what follows, we discuss in Section 2, the ambiguity of the definition of a permutation in the case of missing operations. In Section 3, we present special cases where there is no ambiguity in the definition of a permutation. Concluding remarks are followed in Section 4.

3.2 AMBIGUITY OF THE DEFINITION OF A PERMUTATION

In this section, we use the second definition of a missing operation, which considers zero processing time as a missing operation, to illustrate the ambiguity of the concept of permutation in the flow shop in this case. Before proceeding, we first introduce the following notation :

- $J = \{J_1, \dots, J_n\}$ denotes the set of jobs.
- The processing time of job j on machine i is denoted by p_{ij} .
- The operation of job j on machine i is denoted by O_{ji} .
- E_j is the set of machines such that for $i \in E_j$ we have $p_{ij} \neq 0$.
- M_i is the set of jobs such that for $j \in M_i$ we have $p_{ij} \neq 0$.

In a classic flow shop with n jobs and m machines, we usually assume that each job comprises m operations. The order of operations on a machine can be considered as a total order (reflexive, anti-symmetric, and transitive binary relation such that all pairs (x, y) are in relation) on the jobs (since each job has one operation on each machine). As a result, a schedule is entirely defined by m total orders on the set of jobs, one on each machine. By merging these orders, we get a binary relation that is :

- A total order if all the orders are the same, and we refer to it as a permutation flow shop.

- A reflexive, non anti-symmetric ¹, and non-transitive relation, in which case we refer to a non-permutation flow shop.

We use a directed graph $G = (V, E)$ to represent the binary relations between jobs. The set V of vertices represents the set of jobs, and the set E of arcs is defined as follows : an arc is drawn from vertex a to vertex b if job a precedes job b on at least one machine.

When there are missing operations, the problem becomes more complex as on each machine there is only a partial order on the set of jobs (a total order on a subset of jobs, those that are used on that machine). If we merge the m partial orders, several situations may occur :

Case 1 : The transitive closure of the resulting relation is a partial or total order as illustrated by the following Gantt chart pictured by Figure 3.1.

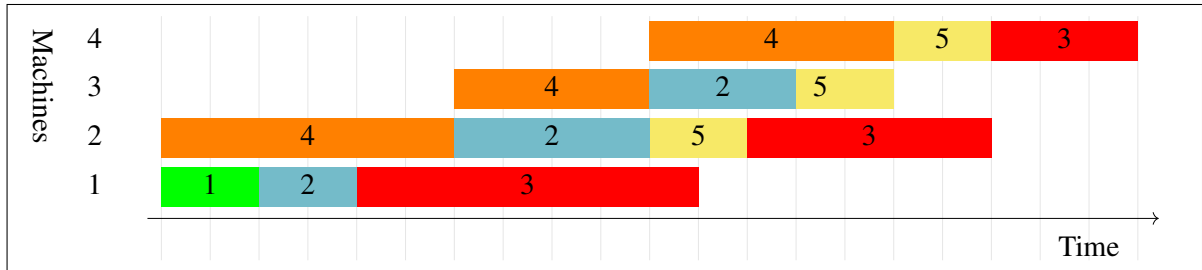


FIGURE 3.1 : Illustrative example for case 1

1. In our context, anti-symmetry means that if a job a precedes a job b on a machine, b cannot precede a on any other machine.

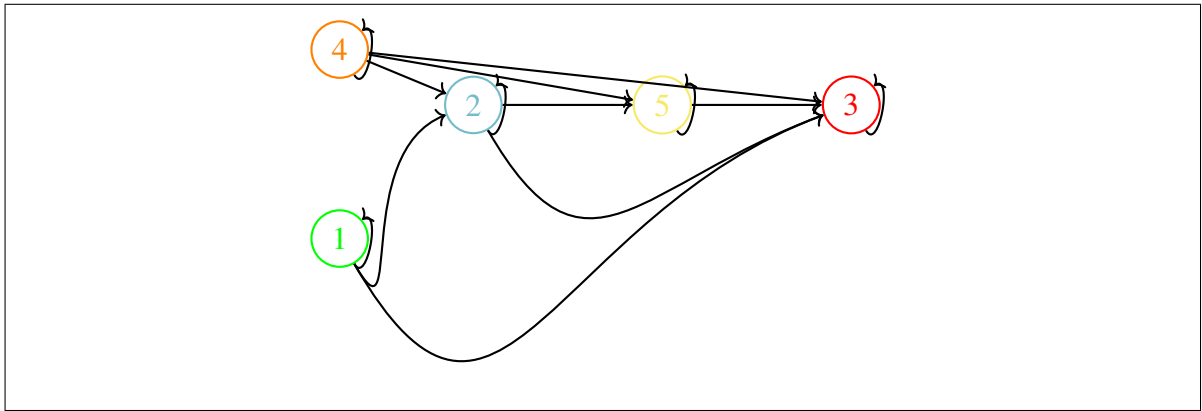


FIGURE 3.2 : Visualizing graph $G = (V, E)$ representing the binary relations between jobs as in Case 1

In this case, the transitive closure of the resulting relation is a partial order. Therefore, there are 2 possible underlying orders : $1 - 4 - 2 - 5 - 3$ and $4 - 1 - 2 - 5 - 3$.

Case 2 : The resulting relation is antisymmetric but not transitive. To make it clearer, we provide the following Gantt chart pictured by Figure 3.3.

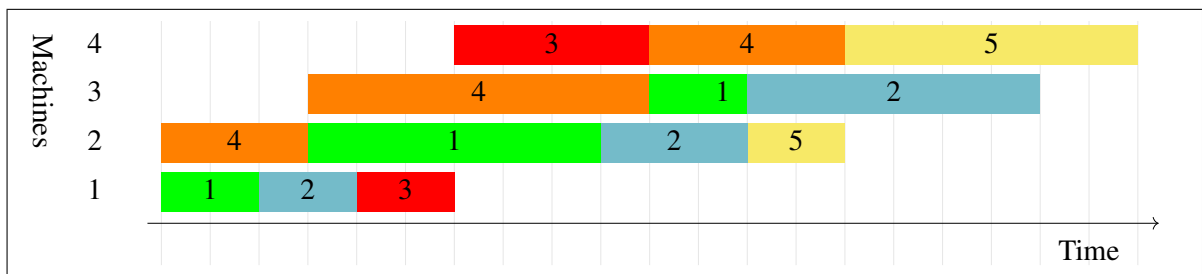


FIGURE 3.3 : Illustrative example for case 2

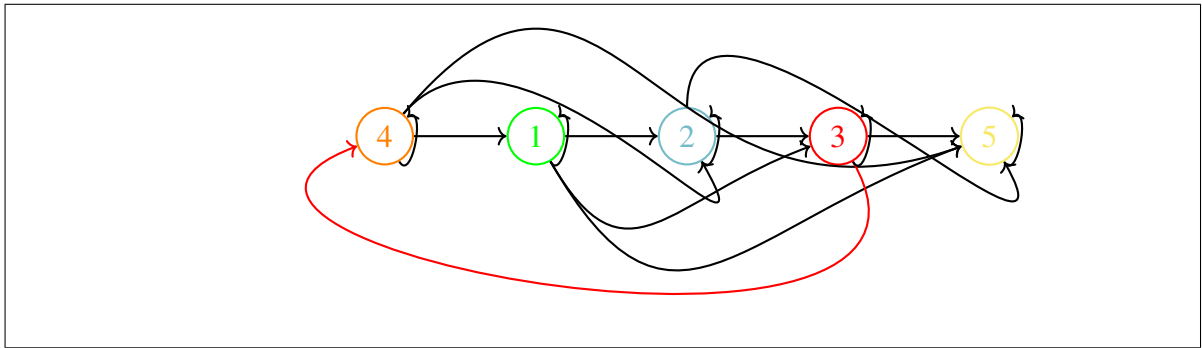


FIGURE 3.4 : Visualizing graph $G = (V, E)$ representing the binary relations between jobs as in Case 2

In this case, the union of the 4 partial orders is a binary relation that is anti-symmetric, but the relation is not transitive : there are cycles, related to the arc from 3 to 4 as pictured in graph G as below. Therefore, there is no total order underlying the different partial orders defined on each of the machines.

Case 3 : The resulting relation is not anti-symmetric. To clarify, we again provide the following Gantt chart pictured by Figure 3.5.

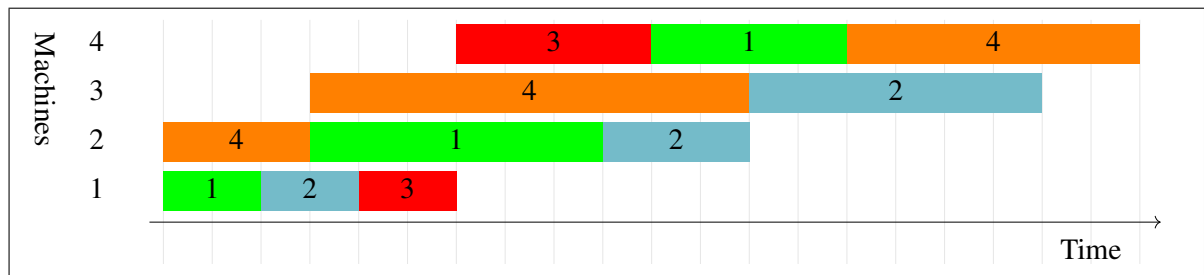


FIGURE 3.5 : Illustrative example for case 3

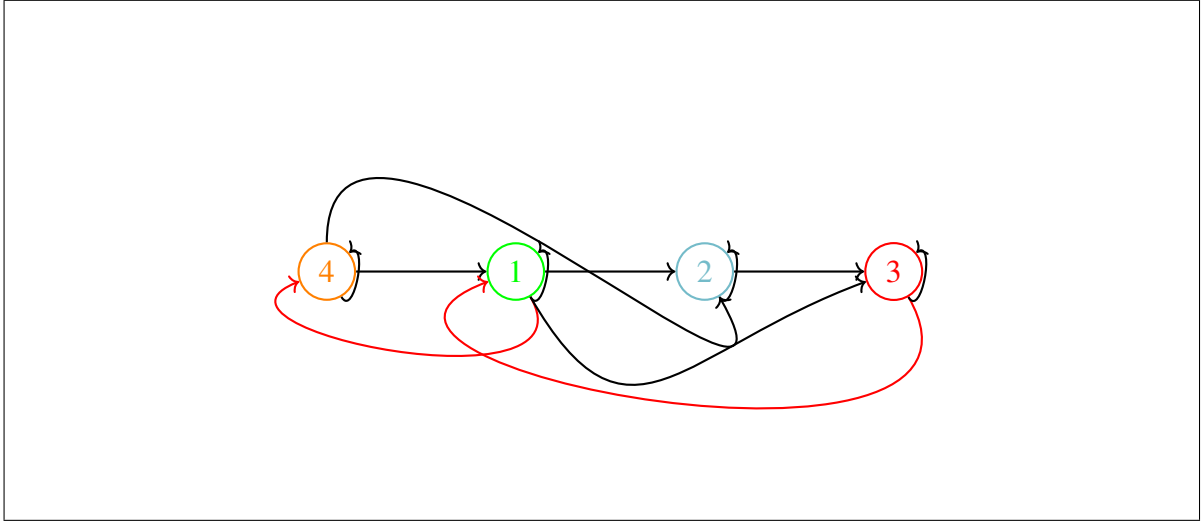


FIGURE 3.6 : Visualizing graph $G = (V, E)$ representing the binary relations between jobs as in case 3

In this case, the union of the 4 partial orders creates a binary relation that is not antisymmetric as there are arcs between 1 and 3, 3 and 1, 1 and 4, 4 and 1.

The two first cases presented above illustrate the emergence of two distinct concepts of a permutation : strong and weak permutation defined as follows.

Définition 3.2.1. *In a flow shop with missing operations, a permutation is strong if the transitive closure of the union of the graphs $G = (V, E)$ is a partial or total order. As a result, at least one total order is underlying the partial orders on each machine.*

Let us note if, for every pair of jobs a and b , there is at least one machine used by both jobs, then the strong permutations are those for which there exists a unique underlying total order.

Définition 3.2.2. *In the case of flow shop with missing operations, a permutation is weak if there is no over passing between jobs i.e. if job a precedes job b in one machine, then this is true on all the remaining machines*

Let us note Definition 2 is particularly useful in service settings where it is important to avoid situations that could be seen as unfair, such as jobs passing.

The following two definitions are introduced as they are needed in the proof of the results we are presenting below.

Définition 3.2.3. *Let I be the set of instances of FSMOP. $D_1(x)$ and $D_2(x)$, $x \in I$, are the search space of the permutation flow shop problem with missing operations using Definition 1 and Definition 2, respectively.*

Définition 3.2.4. *Definition 1 and Definition 2 are equivalent if they have the same search space.*

An immediate consequence of the above definitions is the following result.

Théorème 3.2.1. *Any permutation schedule according to Definition 1 is a permutation schedule according to Definition 2.*

Proof. The result follows from Definition 1 and Definition 2, since from a permutation having a total precedence order for a set of jobs, we can derive a precedence order for all pairs of jobs. ■

The two definitions are not equivalent, even if the number of machines is restricted to 3, as illustrated by Figure 3.7

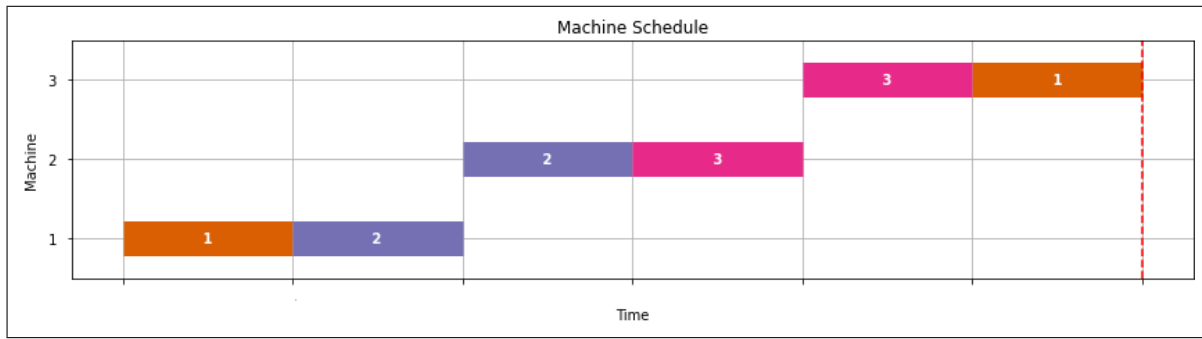


FIGURE 3.7 : Permutation scheme according to Definition 2 but not to Definition 1

The ambiguity of the definition of a permutation is worsened by the fact that the optimal makespan values of the FSMOP problem might be different, depending on the permutation definition used, for $m \geq 4$. Indeed, we show that the two definitions do not generate the same optimal makespan values for $m = 4$. To do so, let us consider the following instance of a permutation flow shop problem with missing operations with 3 jobs, say J_1 , J_2 , and J_3 , to be processed on 4 machines. The processing times of these jobs are displayed as in Table D.1.

	M_1	M_2	M_3	M_4
J_1	0	1	2	1
J_2	1	3	0	0
J_3	1	0	0	1

TABLE 3.1 : PROCESSING TIMES OF THE JOBS

- The optimal solution for the permutation flow shop problem with missing operations, using Definition 2, is the following sequence order : J_1 precedes J_2 , J_2 precedes J_3 , and J_3 precedes J_1 . The corresponding makespan value is 4, as shown by the following Gantt chart pictured by Figure 3.8.

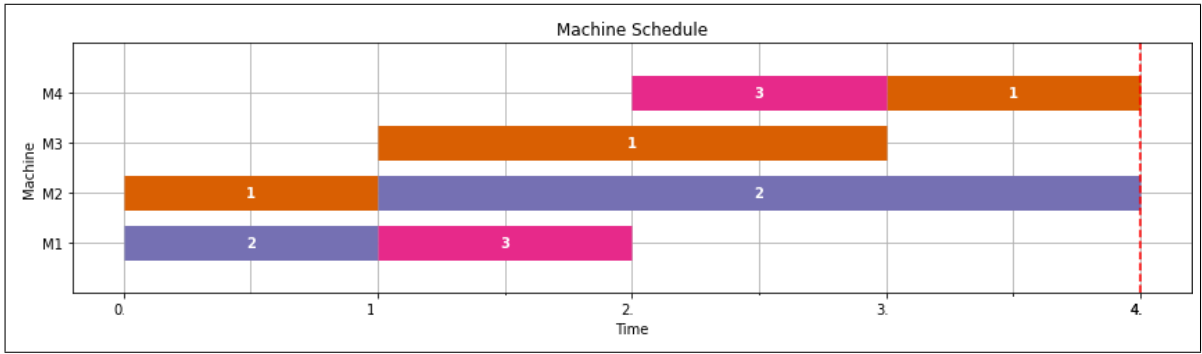


FIGURE 3.8 : Optimal solution using Definition 2

- The optimal solution for the permutation flow shop problem with missing operations, using Definition 1, is the following sequence order : J_1 , J_2 and then J_3 . The corresponding makespan value is thus 5, as shown by the following Gantt chart pictured by Figure 3.9.

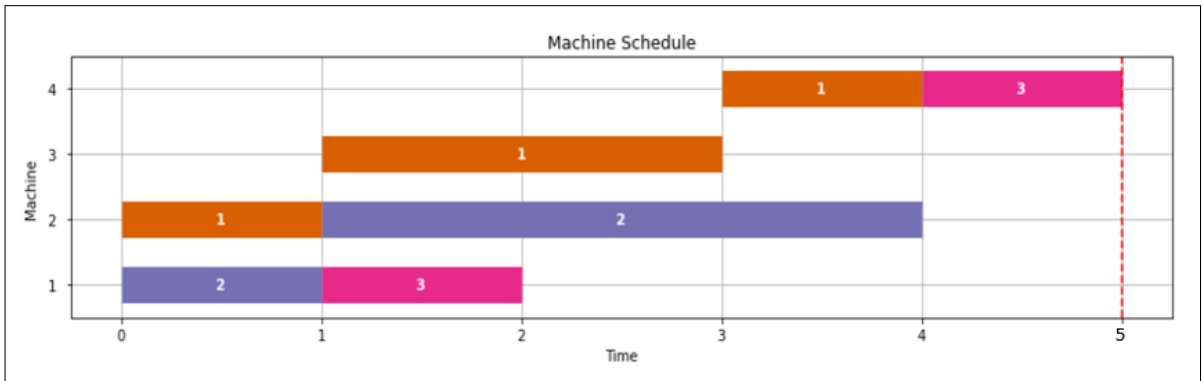


FIGURE 3.9 : Optimal solution using Definition 1

3.3 SPECIAL CASES OF NON AMBIGUITY

In this section, we show that, in some cases, the above two definitions are equivalent, as shown by the following result.

Théorème 3.3.1. *Definition 1 and Definition 2 are equivalent, if there exists at least one machine that processes all the jobs.*

Proof. In the case of the presence of at least one machine in which all jobs are processed, we must show that the relation derived from Definition 2 is a total order over the set of jobs. Let h be the machine that contains all the jobs. Let us now show that the relation generated by Definition 2 is transitive and total.

- Transitivity : For any job J_a, J_b and J_c of J , we have $h \in E_a \cap E_b \cap E_c$. If job J_a precedes job J_b on all machines in $E_b \cap E_a$, and job J_b precedes job J_c on all machines in $E_b \cap E_c$, this implies that J_a precedes J_c on machine h , and therefore J_a precedes job J_c in all the machines of $E_c \cap E_a$. It then follows that the relation is transitive.
- Total : For any job J_a and J_b of J , we have $E_b \cap E_a \neq \emptyset$ as $h \in E_b \cap E_a$. Therefore, according to Definition 2, either J_b precedes J_a on machine h and on all machines in $E_b \cap E_a$, or J_b precedes J_a on machine h and on all machines in $E_b \cap E_a$.

Therefore, the relation deduced from Definition 2 is a total order in the presence of at least one machine, that processes all the jobs. Hence, Definition 1 and Definition 2 are equivalent. ■

The above two definitions generate, for some other cases, solutions with the same makespan value as shown by the next result.

Théorème 3.3.2. *In the case of $m = 3$ machines, the optimal makespan values of the schedules, produced according to Definition 1 and Definition 2, respectively, are of the same value.*

Proof. Let I be the set of instances for the FSMOP. For an instance $x \in I$, let $opt_1(x)$ and $opt_2(x)$ be the optimal makespan values for the permutation flow shop problem with missing operations using Definition 1 and Definition 2, respectively. Since Definition 2 is more general than Definition 1, it follows that for an instance $x \in I$, we have $D_1(x) \subseteq D_2(x)$. Therefore, if there is at least one instance $x \in I$ with $opt_1(x) \neq opt_2(x)$, then there exists $s \in D_2(x)$ and

$s \notin D_1(x)$, with s being the optimal schedule using Definition 2. This means that, within s , there are at least three jobs J_a , J_j and J_k , such that J_a precedes J_j , J_j precedes J_k , and J_k precedes J_a .

Without loss of generality, let J_a and J_j be the two jobs that run on machine 1. We show below that with an optimal permutation schedule according to Definition 2, we derive a valid permutation order according to Definition 1 without increasing the value of the makespan. We proceed as follows. We start from a permutation order according to Definition 2, we apply a set of exchange rules aimed at correcting the case or cases of non-transitivity of the jobs in the machines. We denote by $T_{h\ell}$ and $c_{h\ell}$, respectively, the start time and the completion time of J_ℓ on machine h within schedule s . Let NT be the set of job triplets (n_1, n_2, n_3) generating a form of non-transitivity. We chose the first non-transitivity case in the following order :

Among all the triplets (n_1, n_2, n_3) , we consider (J_a, J_j, J_k) such that $J_a = \arg \min\{T_{1,n_1} | (n_1, n_2, n_3) \in NT\}$, $J_j = \arg \min\{T_{1,n_2} | (J_a, n_2, n_3) \in NT\}$ and $J_k = \arg \min\{T_{2,n_3} | (J_a, J_j, n_3) \in NT\}$. Let J_a , J_j , and J_k be the first jobs where the non-transitivity property is encountered. There are two cases to consider Figure 3.10 :

1. J_a precedes J_j on machine 1, J_j precedes J_k on machine 2 and J_k precedes J_a on machine 3. This is possible only if J_a , J_j and J_k have respectively a missing operation on machines 2, 3 and 1.

To correct this case of non transitivity, we proceed as follows (see Figure 3.11). Let $J_{a_1}, \dots, J_{a_h}$ be the list of jobs processed between jobs J_a and J_j on machine 1. We process J_j at time T_{1a} followed, in this order, by $J_a, J_{a_1}, \dots, J_{a_h}$. We denote by s' the resulting schedule. We denote by $T'_{h\ell}$ and $c'_{h\ell}$, respectively, the start time and the completion

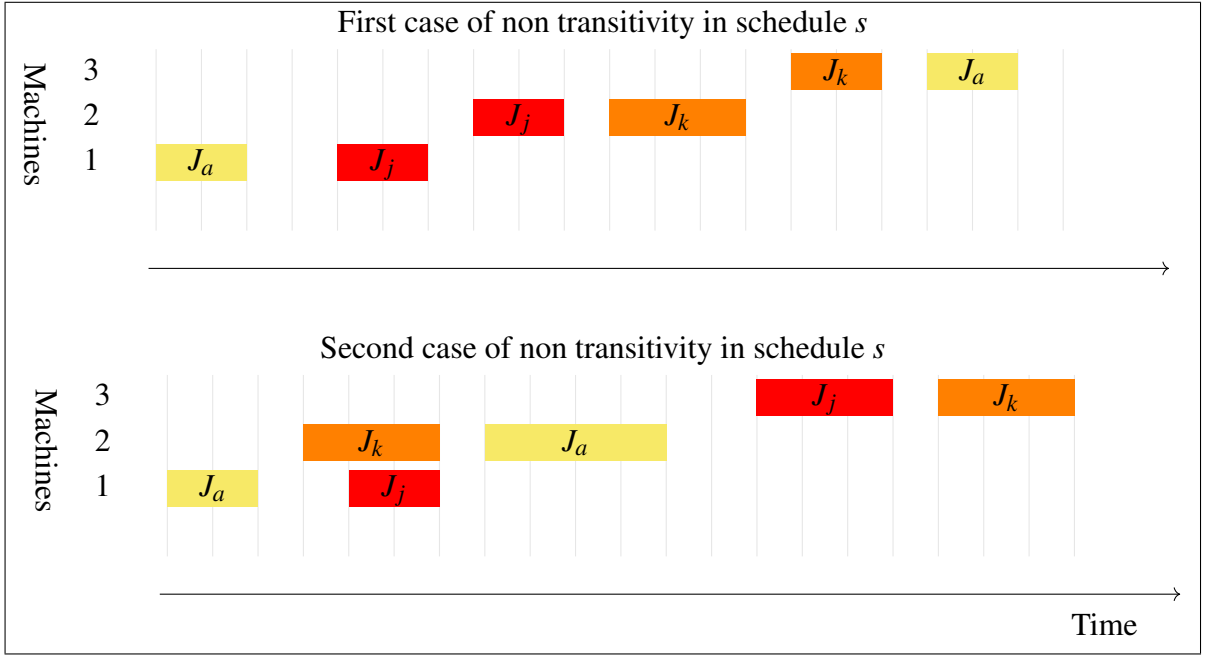


FIGURE 3.10 : Cases of non transitivity.

time of J_ℓ on machine h within s' . We have $T_{jh} = T'_{jh}$ and $c_{jh} = c'_{jh}$, $h \in M_i$ with $i \in \{2, 3\}$, and $T_{1h} = T'_{1h}$ and $c_{1h} = c'_{1h}$, $h \in M_1 \setminus \{J_j, J_a, J_{a_1}, \dots, J_{a_h}\}$. The completion time of $J_a, J_{a_\ell}, \dots, J_{a_h}$ on machine 1 is at most equal to c_{1j} .

We now verify that schedule s' is feasible and satisfies the permutation constraints of Definition 1. To do this, we need to check the following two conditions. First, every job in $J_a, J_{a_1}, \dots, J_{a_h}$ has an operation on the same machine as job J_j only on machine 1, *i.e.* each job in $J_a, J_{a_1}, \dots, J_{a_h}$ has a missing operation on machine 2. Second, for every job $j_\ell \in \{J_a, J_{a_1}, \dots, J_{a_h}\} \cap E_3$, we have $c'_{1\ell} \leq T_{3\ell}$.

For J_a , in s' , we have $c'_{1a} \leq T_{3a} = T'_{3a}$, since $c'_{1a} \leq c_{1j} \leq c_{2j} = c'_{2j} \leq c_{2k} = c'_{2k} \leq c_{3k} = c'_{3k} \leq T_{3a} = T'_{3a}$. For the job list $(J_{a_1}, \dots, J_{a_h})$, the following cases are to be distinguished :

- One of the jobs is processed on machine 2 : Let J_h be this job, there are two possible configurations to consider.

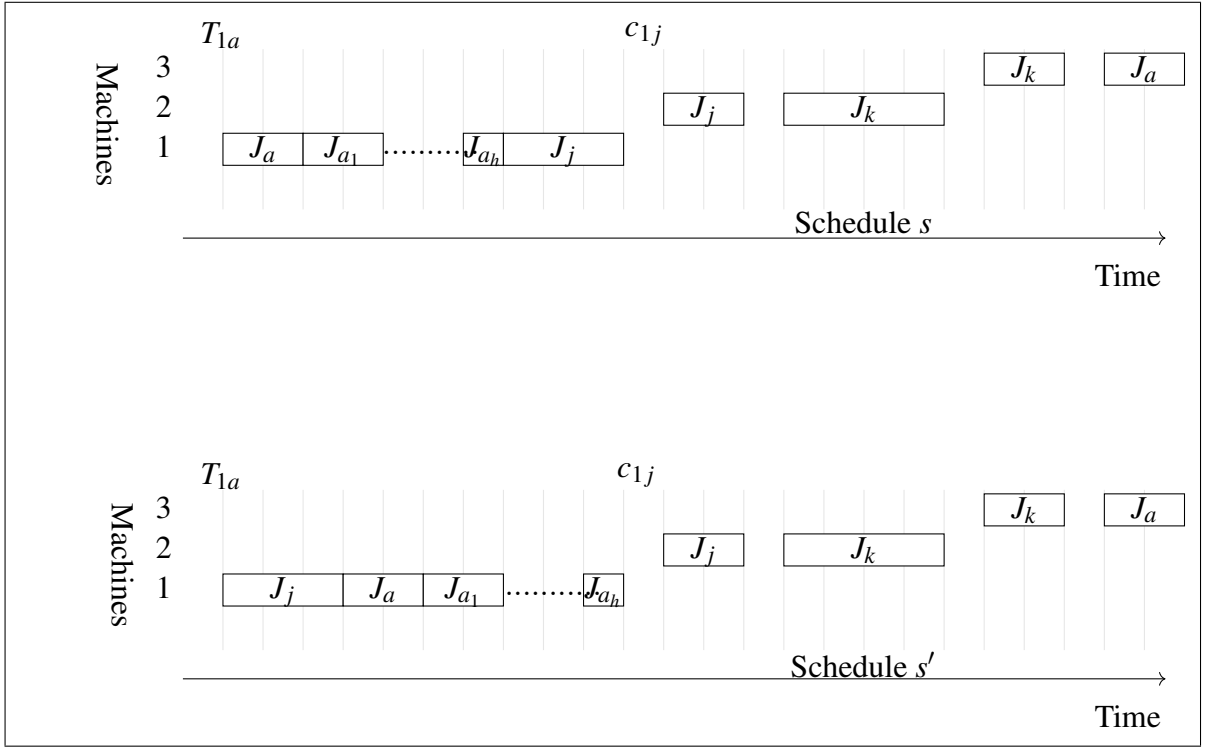


FIGURE 3.11 : Correction of the case of non transitivity

- (a) J_h has a missing operation on machine 3 (see Figure 3.12). Job J_a precedes J_h on machine 1 ; J_h precedes J_k on machine 2, and J_k precedes J_a on machine , since $s \in D_2(x)$. This configuration is impossible to occur as the job triplet, J_a, J_j and J_k , constitute the first non-transitive encountered case.

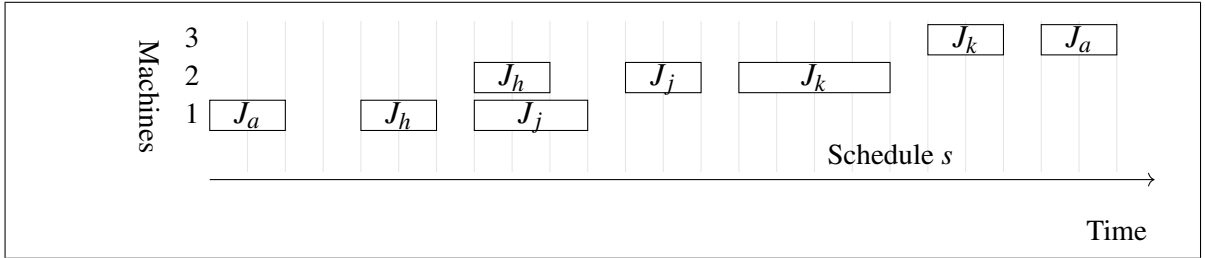


FIGURE 3.12 : (J_a, J_h, J_k) is the first non-transitive encountered case.

- (b) J_h is processed on machine 3 (see Figure 3.13). Thus, within s , with $s \in D_2(x)$, we have the same order of succession between each pair of jobs on all

machines. Hence, J_a precedes J_h on machine 1 and, consequently, on machine 3. Similarly, J_h precedes J_j on machine 1 and, therefore, on machine 2 as well. Since J_j precedes J_k on machine 2, we can deduce that J_h precedes J_k on machine 2 and, therefore, on machine 3 as well. Therefore, we can conclude that J_a precedes J_h , which precedes J_k on machine 3. However, J_k precedes J_i on machine 3. This configuration is not possible.

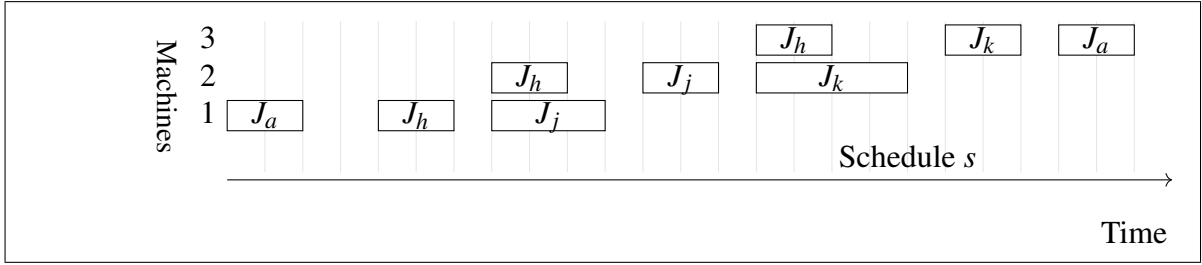


FIGURE 3.13 : This configuration is not possible for a schedule $s \in D_2(x)$.

- None of the jobs is processed on machine 2 : There are two possible configurations : the jobs are processed on either machine 1 or machine 1 and machine 3. In the first case, the conditions mentioned above are verified. In the second case, these jobs are processed on machine 3 after the end of processing of J_a . We know that $J_a, J_{a_1}, \dots, J_{a_h}$ have a completion time on machine 1 at most equal to c_{1j} . In addition, we also have $c_{1j} \leq c'_{2j} \leq c'_{2k} \leq c'_{3k} \leq c'_{3a}$.

Compared to s , the only change in the permutation relation between the jobs in s' is that job J_j precedes $J_a, J_{a_1}, \dots, J_{a_h}$ in all the machines. We conclude that $s' \in D_1(x)$. The modifications, therefore, does not generate an increase in the value of the makespan. In other words, we have $C_{\max}(s) = C_{\max}(s')$.

2. Job J_a precedes job J_j on machine 1, J_k precedes J_a on machine 2 and J_j precedes J_k on machine 3. This is possible only if J_a, J_j and J_k have respectively a missing operation

on machine 3, 2 and 1, as illustrated by Figure 3.10. To correct the non transitivity, we use the same argument as above.

After having corrected the first case of non-transitivity, we set $s = s'$. Then, we repeat this procedure until getting a schedule without transitivity relation between the jobs. At each modification, the value of the makespan is not increased. ■

3.4 CONCLUSION

In this work, we pointed out the ambiguity of the classical definition of a permutation for the flow shop problem with missing operations. We illustrated our study with respect to the makespan criterion. This definition implicitly assumes that the order of jobs in a permutation is transitive. This assumption is not necessarily valid within the context of missing operations. We presented examples to validate this claim. To take into account the context of missing operations, we had to modify the definition of a permutation. Two definitions have then emerged : the classical one forces the transitivity relation between the jobs and the other does not. We have shown that even if the first definition is more natural, the second definition is more general and more realistic than the first definition as it generates more solutions. However, the two definitions are shown to be equivalent in some cases and in other cases, even though they may generate different optimal solutions, they still generate the same makespan value.

An immediate research of interest would be to investigate first the impact of limiting to permutation schedules under the definition of a permutation that does not force the existence of the transitivity property for solving flow shop problems with missing operations, and, second, the impact of missing operations, through the variation of the processing times, with or without the no-wait constraint, in the flow shop problem with and without permutation. The

second point to look at is to what extent our study is relevant with respect to other regular criteria. Finally, it may also be interesting to study different scenarios for distributing missing operations, for example assuming that only certain machines would have missing operations.

ACKNOWLEDGMENTS

This research is partially financed by the Natural Sciences and Engineering Research Council of Canada (NSERC).

BIBLIOGRAPHY

- AHMADIZAR, F. (2012). A new ant colony algorithm for makespan minimization in permutation flow shops. *Computers & industrial engineering*, 63(2), 355-361.
- BENAVIDES, A. J., & RITT, M. (2018b). Fast heuristics for minimizing the makespan in non-permutation flow shops. *Computers & Operations Research*, 100, 230-243.
- BRUM, A., RUIZ, R., & RITT, M. (2022b). Automatic generation of iterated greedy algorithms for the non-permutation flow shop scheduling problem with total completion time minimization. *Computers & Industrial Engineering*, 163, 107843.
- GAREY, M. R., JOHNSON, D. S., & SETHI, R. (1976c). The complexity of flowshop and jobshop scheduling. *Mathematics of operations research*, 1(2), 117-129.
- HENNEBERG, M., & NEUFELD, J. (2016b). A constructive algorithm and a simulated annealing approach for solving flowshop problems with missing operations. *International Journal of Production Research*, 54(12), 3534-3550.
- HSU, C.-Y., CHANG, P.-C., & CHEN, M.-H. (2015). A linkage mining in block-based evolutionary algorithm for permutation flowshop scheduling problem. *Computers & Industrial Engineering*, 83, 159-171.
- JOHNSON, S. M. (1954b). Optimal two-and three-stage production schedules with setup times included. *Naval research logistics quarterly*, 1(1), 61-68.
- LEISTEN, R., & KOLBE, M. (1998b). A note on scheduling jobs with missing operations in permutation flow shops. *International journal of production research*, 36(9), 2627-2630.
- LIAO, C., LIAO, L., & TSENG, C. (2006b). A performance evaluation of permutation vs. non-permutation schedules in a flowshop. *International Journal of Production Research*, 44(20), 4297-4309.
- POTTS, C. N., SHMOYS, D. B., & WILLIAMSON, D. P. (1991c). Permutation vs. non-permutation flow shop schedules. *Operations Research Letters*, 10(5), 281-284.
- PUGAZHENDHI, S., THIAGARAJAN, S., RAJENDRAN, C., & ANANTHARAMAN, N. (2003b). Performance enhancement by using non-permutation schedules in flowline-based manufacturing systems. *Computers & Industrial Engineering*, 44(1), 133-157.
- RAMEZANIAN, R., SAIDI, M. M., & RAHMANI, D. (2011). Flow shop scheduling problem with missing operations : Genetic algorithm and tabu search.
- ROSSIT, D. A., TOHMÉ, F., & FRUTOS, M. (2018b). The non-permutation flow-shop scheduling problem : a literature review. *Omega*, 77, 143-153.
- RUIZ, R., & MAROTO, C. (2005b). A comprehensive review and evaluation of permutation flowshop heuristics. *European journal of operational research*, 165(2), 479-494.

- SADJADI, S., ARYANEZHAAD, M., & ZIAEE, M. (2008b). The general flowshop scheduling problem : mathematical models. *J. of app. sci*, 8(17), 3032-3037.
- SRIDHAR, J., & RAJENDRAN, C. (1993b). Scheduling in a cellular manufacturing system : a simulated annealing approach. *International Journal of Production Research*, 31(12), 2927-2945.

CHAPITRE IV

FLOW SHOP AVEC OPÉRATIONS MANQUANTES

Dans ce chapitre, nous abordons diverses propriétés et variantes du problème de flow shop avec opérations manquantes. La première partie prolonge le Chapitre 3, où nous avons souligné l’ambiguïté de la notion de permutation et introduit deux nouvelles définitions : la permutation faible et la permutation forte, dans un contexte de flow shop avec opérations manquantes. Nous y présentons également une troisième définition de permutation, telle que décrite par PINEDO, [2002c](#), où les tâches respectent un principe de passage en file d’attente (FIFO) sur les machines. Les implications et propriétés de cette approche sont ensuite discutées dans le cadre du problème de flow shop avec des opérations manquantes en minimisant le critère de makespan.

Dans la seconde partie, nous nous focalisons sur le cas du flow shop unitaire avec opérations manquantes. Nous y exposons certaines propriétés spécifiques du modèle et analysons le pire cas de quelques heuristiques simples.

4.1 NOTION DE PERMUTATION SELON PINEDO

Un problème de flow shop peut être défini comme un ensemble de n tâches $\mathcal{J} = \{J_1, J_2, \dots, J_n\}$ qui doivent être traitées sur un ensemble de m machines $\mathcal{M} = \{M_1, M_2, \dots, M_m\}$. Chaque tâche J_i est composée d’une séquence d’opérations $\{O_{i1}, O_{i2}, \dots, O_{im}\}$, où O_{ij} représente l’opération de la tâche J_i sur la machine M_j .

Le flow shop de permutation est traditionnellement défini en imposant la même séquence de tâches sur toutes les machines. Cependant, OUCHENE et al., [2023b](#) soulignent une ambiguïté dans cette définition classique lorsqu’on considère la possibilité d’opérations manquantes.

Deux variantes apparaissent alors : la *permutation faible* (*weak permutation*) et la *permutation forte* (*strong permutation*). Dans le cadre d'un flow shop avec opérations manquantes :

- Une **permutation faible** se définit par l'absence de dépassement de tâches : si la tâche i précède la tâche j sur une machine, alors elle précède nécessairement la tâche j sur toute machine suivante ;
- Une **permutation forte**, quant à elle, impose un ordre global des tâches sur l'ensemble des machines, indépendamment du fait que des opérations puissent manquer.

Par ailleurs, Pinedo (voir PINEDO, [2002c](#)) propose une définition basée sur la discipline dite **FIFO** (*First In, First Out*) pour la gestion de la file d'attente des machines. Cette définition est équivalente à celle du flow shop permutation en l'absence d'opérations manquantes, mais elle peut différer dès lors que certaines opérations sont manquantes pour certaines tâches.

Relation d'inclusion : Comme démontré dans OUCHENE et al., [2023b](#), le domaine des solutions qui respectent la permutation forte est inclus dans celui des solutions qui respectent la permutation faible. Autrement dit, toute solution forte est nécessairement faible, mais l'inverse n'est pas garanti, en particulier en présence d'opérations manquantes.

Il est à noter que ces différences entre permutation faible, permutation forte et FIFO au sens de Pinedo ne se manifestent qu'à partir de trois machines. Pour des instances à deux machines, les notions coïncident ou se réduisent à des cas triviaux.

Les relations entre ces différentes notions peuvent être résumées ainsi : le domaine des solutions qui respectent la permutation forte est inclus dans celui qui respecte la permutation faible, tandis que la définition de permutation selon Pinedo n'est ni incluse, ni inclusive vis-à-vis de la permutation faible et forte, en présence d'opérations manquantes. Il est donc possible de trouver des ordonnancements qui respectent la permutation forte mais pas la définition

FIFO de Pinedo, ou inversement, des ordonnancements qui respectent Pinedo sans respecter la permutation faible.

Exemple 1 : *Une solution qui respecte la permutation forte mais pas la définition FIFO de Pinedo.*

Dans cet exemple, l'ordre sous-jacent global est 1–2 sur toutes les machines (permutation forte). Cependant, l'ordre d'arrivée des tâches sur chaque machine est le suivant :

- Machine 1 : tâche 1 = 0, tâche 2 = 0
- Machine 2 : tâche 1 = 2
- Machine 3 : tâche 1 = 7, tâche 2 = 4

L'ordonnancement ne respecte donc pas toujours l'ordre d'arrivée des tâches sur chaque machine (la FIFO de Pinedo n'est pas respectée).

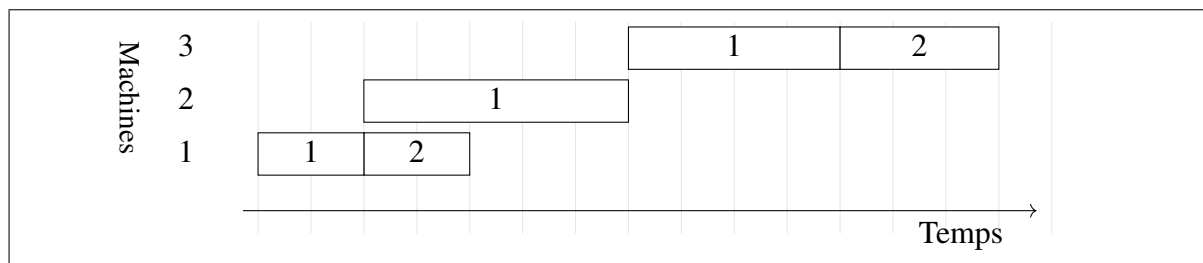


FIGURE 4.1 : Ordonnancement respectant la permutation forte mais pas la définition FIFO de Pinedo

Exemple 2 : *Une solution qui respecte la définition FIFO de Pinedo mais pas la permutation faible.*

Ici, l'ordonnancement respecte l'ordre d'arrivée des tâches sur chaque machine (FIFO), mais l'ordre relatif entre deux tâches change d'une machine à l'autre : par exemple, la tâche 1

précède la tâche 2 sur la machine 1, mais la tâche 2 précède la tâche 1 sur la machine 3 (ni permutation faible, ni forte).

Ordres d'arrivée des tâches sur chaque machine :

- Machine 1 : tâche 1 = 0, tâche 2 = 0
- Machine 2 : tâche 1 = 2
- Machine 3 : tâche 1 = 7, tâche 2 = 4

L'ordre d'arrivée est respecté sur chaque machine, mais la permutation faible n'est pas respectée.

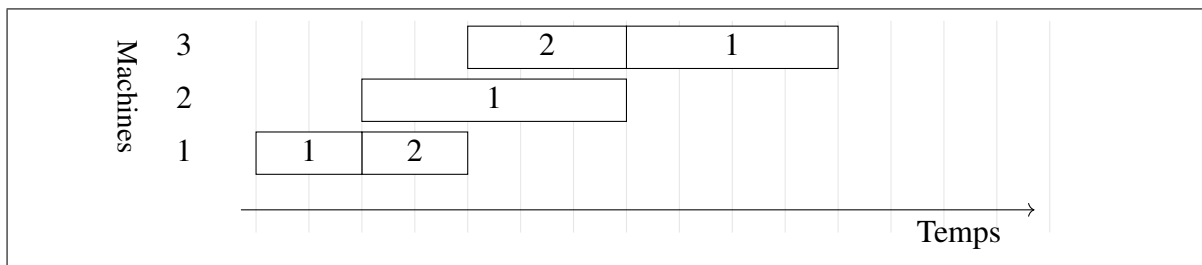


FIGURE 4.2 : Ordonnancement respectant la FIFO de Pinedo, mais n'étant ni une permutation faible ni forte

En résumé, le schéma 4.4 d'inclusion des domaines de solutions peut être illustré comme suit :

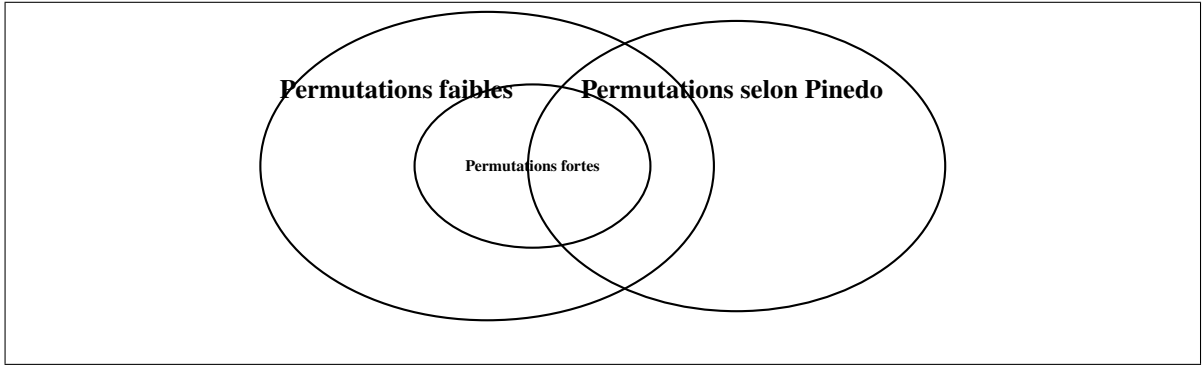


FIGURE 4.4 : Inclusion des permutations faibles, fortes et selon Pinedo.

Les domaines "faible" et "Pinedo" se recouvrent partiellement : ni inclusion stricte, ni exclusion, sauf pour "forte" qui est strictement incluse dans "faible".

Dans la suite, nous illustrons plusieurs propriétés relatives au flow shop de permutation au sens de Pinedo, dans un contexte où certaines opérations peuvent être manquantes.

Théorème 4.1.1. *Dans l'ordonnancement d'un problème de flow shop avec opérations manquantes, sous le critère du makespan, tout ordonnancement où les tâches respectent la discipline FIFO sur la dernière machine M_m est dominant.*

Démonstration. Considérons un système de flow shop avec opérations manquantes. Lorsque nous nous focalisons sur l'ordonnancement des tâches sur la dernière machine M_m , le problème se ramène à un problème d'ordonnancement sur une machine unique, noté $1 \mid r_i \mid C_{\max}$, où r_i désigne la date de libération (ou *release time*) de la tâche i sur la machine M_m . Cette date de libération correspond à l'instant où la tâche termine son traitement sur les machines précédentes.

La solution optimale du problème $1 \mid r_i \mid C_{\max}$ consiste à séquencer les tâches dans l'ordre croissant de leurs dates de libération r_i . En d'autres termes, il convient d'ordonner les tâches suivant leur ordre d'arrivée à la machine M_m , ce qui correspond précisément à la discipline

FIFO (premier arrivé, premier servi). Ainsi, dans un flow shop avec opérations manquantes, tout ordonnancement où les tâches suivent une discipline FIFO sur M_m n'augmente pas la valeur du makespan et est donc dominant. ■

Corollaire 4.1.1.1 (Cas de deux machines). *Pour un problème d'ordonnancement de flow shop à deux machines, sous le critère du makespan, les solutions de type flow shop de permutation au sens de Pinedo sont dominantes.*

Théorème 4.1.2. *Dans l'ordonnancement d'un problème de flow shop à deux machines sous le critère du makespan, les ordonnancements respectant la discipline FIFO sur la machine M_2 sont dominants.*

Démonstration. Les tâches arrivent dans la file d'attente de la deuxième machine, soit depuis la machine 1, soit directement si la machine 2 correspond à leur première opération. Pour garantir que la file d'attente de la deuxième machine respecte la discipline FIFO, il faut montrer que si une tâche j termine son exécution après une tâche i sur la machine M_1 , alors i doit commencer son exécution avant j sur la machine M_2 . Considérons la première paire de tâches (i, j) telle que i précède j sur la machine M_1 mais où j précède i sur la machine M_2 . Supposons qu'entre i et j se trouvent les tâches $i_1, \dots, i_k$ sur la machine M_1 . Nous retirons alors la tâche i de la file sur la machine M_1 , décalons les tâches $i_1, \dots, i_k$ ainsi que j et plaçons ensuite la tâche i après j . Cette opération permet de rétablir l'ordre (i, j) sur les deux machines sans perte de faisabilité ni augmentation du makespan. En répétant ce procédé pour toutes les paires de tâches mal ordonnées, nous obtenons un ordonnancement respectant la discipline FIFO sur M_2 , prouvant ainsi la dominance de ce type de solution. ■

Théorème 4.1.3 (Cas de trois machines). *Pour un problème de flow shop à trois machines avec opérations manquantes, sous le critère du makespan, les ordonnancements de permutation selon Pinedo sont dominants.*

Démonstration. Dans un flow shop à trois machines, pour qu'un ordonnancement respecte la définition de Pinedo, il faut que les files d'attente devant les machines 2 et 3 suivent la discipline FIFO. Les Théorèmes 4.1.2 et 4.1.1 montrent respectivement que la discipline FIFO est dominante pour les deuxième et troisième machines. En combinant ces deux résultats, on en déduit que tout ordonnancement respectant la discipline FIFO sur M_2 et M_3 est dominant sous le critère du makespan. ■

Remarque : Dans le cas où aucune opération n'est manquante, le Théorème 4.1.3 est équivalent au théorème classique énoncé dans FRENCH, 1982a. Pour un problème de flow shop à trois machines sans opérations manquantes, il est bien établi que les ordonnancements en permutation sont dominants sous le critère du makespan.

Nous présentons à présent un modèle mathématique de formulation pour le problème de flow shop permutation au sens de Pinedo, sous le critère du makespan.

PARAMÈTRES

- n : nombre de tâches.
- m : nombre de machines.
- p_{ij} : durée de traitement de la tâche i sur la machine M_j .
- mpr_{ij} : machine qui précède immédiatement la machine j pour la tâche i . (Pour la première opération d'une tâche, $\text{mpr}_{i,m_1} = -1$ indiquant qu'il n'y a pas d'opération antérieure.)
- K : une constante suffisamment grande (big-M).

VARIABLES DE DÉCISION

- x_{ij} : instant de début de la tâche i sur la machine j .
- L : valeur du makespan (date de fin de la dernière opération parmi toutes les tâches).
- $y_{ji_2i_1}$: variable binaire valant 1 si la tâche i_1 est traitée **après** la tâche i_2 sur la machine j , et 0 sinon.
- $y_{i_1i_2}$: variable binaire valant 1 si la tâche i_1 est traitée après la tâche i_2 sur l'ensemble des machines, et 0 sinon.

FORMULATION

$$\min \quad L \quad (1)$$

$$L \geq x_{ij} + p_{ij} \quad \forall i \in \{1, \dots, n\}, j \in \{1, \dots, m\}. \quad (2)$$

$$x_{ij} \geq x_{i,j-1} + p_{i,j-1} \quad \forall i \in \{1, \dots, n\}, j \in \{2, \dots, m\}. \quad (3)$$

$$x_{i_2j} + p_{i_2j} - (1 - y_{i_2i_1j})K \leq x_{i_1j} \quad \forall i_1 \neq i_2, p_{i_2j} \neq 0, p_{i_1j} \neq 0, j \in \{1, \dots, m\}. \quad (4)$$

$$x_{i_1j} + p_{i_1j} - y_{i_2i_1j}K \leq x_{i_2j} \quad \forall i_1 \neq i_2, p_{i_2j} \neq 0, p_{i_1j} \neq 0, j \in \{1, \dots, m\}. \quad (5)$$

$$y_{i_1i_2j} + y_{i_2i_1j} = 1 \quad \forall i_1 \neq i_2, j \in \{1, \dots, m\}. \quad (6)$$

$$x_{i_1, \text{mpr}_{i_1j}} + p_{i_1, \text{mpr}_{i_1j}} + y_{i_1i_2j}K \geq x_{i_2, \text{mpr}_{i_2j}} + p_{i_2, \text{mpr}_{i_2j}} \quad \forall i_1 \neq i_2, \text{mpr}_{i_1j}, \text{mpr}_{i_2j} \neq -1, j \in \{1, \dots, m\}. \quad (7)$$

$$x_{ij} \geq 0 \quad \forall i \in \{1, \dots, n\}, j \in \{1, \dots, m\}. \quad (8)$$

$$y_{i_1i_2j} \in \{0, 1\} \quad \forall i_1, i_2 \in \{1, \dots, n\}, j \in \{1, \dots, m\}. \quad (9)$$

La fonction objectif vise à minimiser le makespan L . Les contraintes (2) permettent de déterminer la date de fin de toutes les tâches, tandis que les contraintes (3) garantissent la

séquentialité des opérations d'une même tâche sur les différentes machines. Les contraintes (4) et (5) assurent qu'une machine ne peut traiter qu'une seule tâche à la fois. Ensuite, les contraintes (6) fixent l'ordre d'exécution des tâches i_1 et i_2 sur la machine M_j , garantissant que l'une soit exécutée après l'autre. Enfin, les contraintes (7) imposent le respect de la discipline FIFO entre les tâches.

4.2 FLOW SHOP UNITAIRE AVEC OPÉRATIONS MANQUANTES

Nous considérons le problème d'ordonnancement flow shop avec opérations manquantes et temps de traitement unitaires, dans le but de minimiser le makespan ($C_{\max}$).

NOTATIONS

- $T = \{1, \dots, n\}$: ensemble des tâches.
- T_j : ensemble des tâches ayant un temps de traitement non nul sur la machine M_j . Son cardinal est noté $|T_j|$, avec $|T_j| > 0$.
- n_i : nombre d'opérations de la tâche i , avec $n_i > 0$.
- m : nombre de machines.
- $opt(I)$: makespan optimal d'une instance I .

Proposition 4.2.1. *Soit I une instance quelconque du problème flow shop avec opérations manquantes et temps unitaires, comportant n tâches et m machines. Alors,*

$$opt(I) \geq \frac{n}{m}.$$

Démonstration. Soit I une instance avec n tâches et m machines, où chaque tâche possède au moins une opération. Nous avons :

$$\sum_{j=1}^m |T_j| \geq n.$$

Posons

$$M = \max_{j=1, \dots, m} (|T_j|).$$

On obtient alors

$$\text{opt}(I) \geq M.$$

D'autre part,

$$m \times M \geq \sum_{j=1}^m |T_j| \geq n,$$

d'où

$$M \geq \frac{n}{m}.$$

Par conséquent,

$$\text{opt}(I) \geq \frac{n}{m}.$$

■

Dans la suite, nous analysons la performance dans le « pire cas » de plusieurs heuristiques constructives élémentaires :

- un ordonnancement selon un ordre aléatoire des tâches,
- une règle privilégiant les tâches ayant le plus petit nombre d'opérations,
- et, à l'inverse, une règle privilégiant celles ayant le plus grand nombre d'opérations.

Bien que ces approches soient rapides à mettre en œuvre, nous verrons qu'elles peuvent aboutir à un *makespan* dont la valeur peut être jusqu'à un facteur m supérieure à celle d'une solution optimale.

HEURISTIQUE 1 : ALÉATOIRE

Considérons l'heuristique H , composée de deux étapes :

1. Générer une liste aléatoire des tâches.
2. Insérer les tâches une par une dans le flow shop le plus tôt possible.

On note $H(I)$ le makespan obtenu en appliquant cette heuristique à l'instance I .

Théorème 4.2.1. *Pour l'heuristique H , le rapport de performance absolu vaut $\rho = m$, et cette borne est atteinte.*

Démonstration. Soit I une instance du problème de flow shop avec opérations manquantes et temps unitaires, comportant n tâches et m machines.

Montrons que

$$\sum_{j=1}^m |T_j| \geq H(I).$$

Considérons une liste arbitraire de tâches L . Construisons alors un ordonnancement réalisable O qui respecte l'ordre de la liste L , de la manière suivante :

- Sur la machine M_1 , on ordonnance le plus tôt possible toutes les tâches qui nécessitent une opération sur cette machine.
- Pour chaque machine j (avec $j = 2, \dots, m$), on planifie les opérations le plus tôt possible à partir de l'instant $|T_{j-1}|$.

Cette construction fournit un ordonnancement réalisable O (illustré à la figure 4.5), dont le *makespan* est

$$C_{\max}(O) = \sum_{j=1}^m |T_j|.$$

Il respecte l'ordre de la liste L , mais les tâches n'y sont pas nécessairement insérées au plus tôt sur chaque machine. Ainsi, puisque

$$C_{\max}(O) \geq H(I),$$

il s'ensuit que

$$\sum_{j=1}^m |T_j| \geq H(I).$$

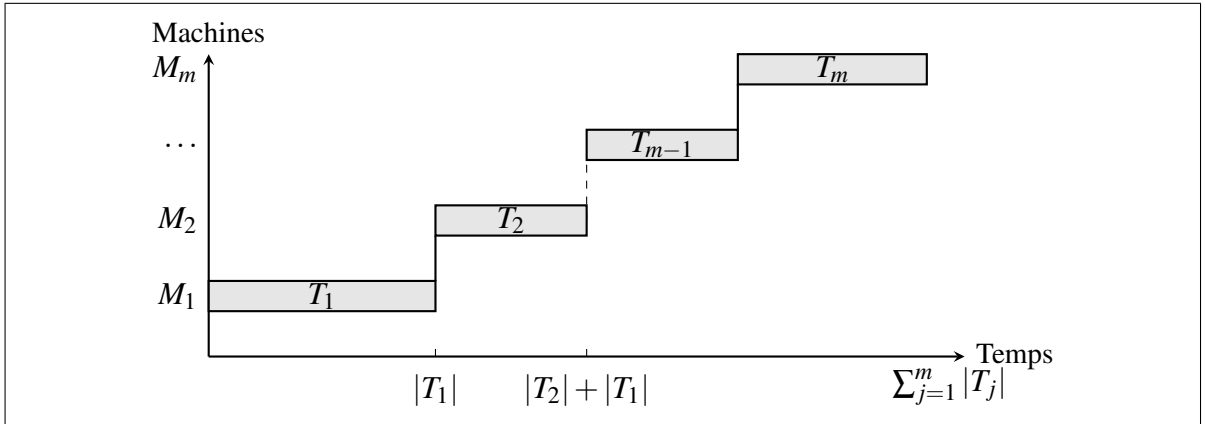


FIGURE 4.5 : Structure d'un ordonnancement O respectant l'ordre de L .

Pour chaque machine j , on a $\text{opt}(I) \geq |T_j|$. En sommant ces inégalités pour l'ensemble des machines, on obtient :

$$m \times \text{opt}(I) \geq \sum_{j=1}^m |T_j| \geq H(I).$$

Il en résulte que

$$m \geq \frac{H(I)}{\text{opt}(I)},$$

ce qui montre que le rapport de performance absolu de l'heuristique H est égal à m .

Borne atteinte : Pour voir que cette borne peut être atteinte, considérons l'instance I avec m machines et $n = m + 1$ tâches, définie comme suit :

$$p_{1,j} = \begin{cases} 1 & \text{si } j = 1, \\ 0 & \text{sinon,} \end{cases} \quad p_{m+1,j} = \begin{cases} 1 & \text{si } j = m, \\ 0 & \text{sinon,} \end{cases} \quad p_{i,j} = \begin{cases} 1 & \text{si } j = i-1 \text{ ou } j = i, \\ 0 & \text{sinon,} \end{cases} \quad i = 2, \dots, m.$$

En prenant la liste $L = (1, \dots, n)$ (i.e. la séquence de tâches $\{1, 2, \dots, m, m+1\}$), on obtient un ordonnancement de makespan $C_{\max} = 2m$. Or, la solution optimale a un makespan $\text{opt}(I) = 2$. Les figures 4.6 et 4.7 illustrent respectivement la solution approximative fournie par l'heuristique H et la solution optimale pour cette instance de référence.

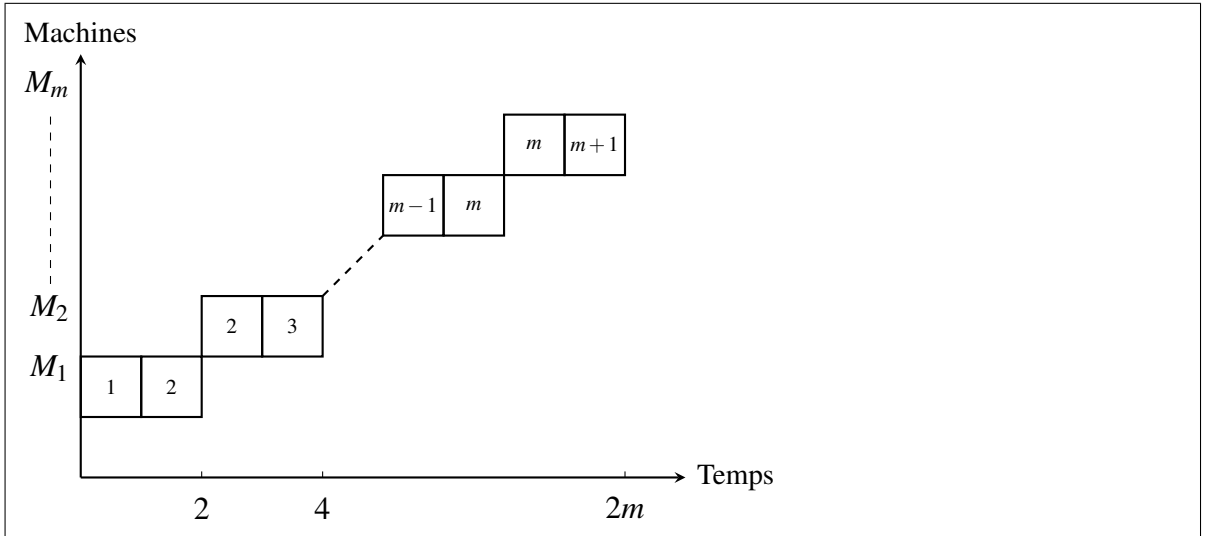


FIGURE 4.6 : Solution produite par l'heuristique H .

■

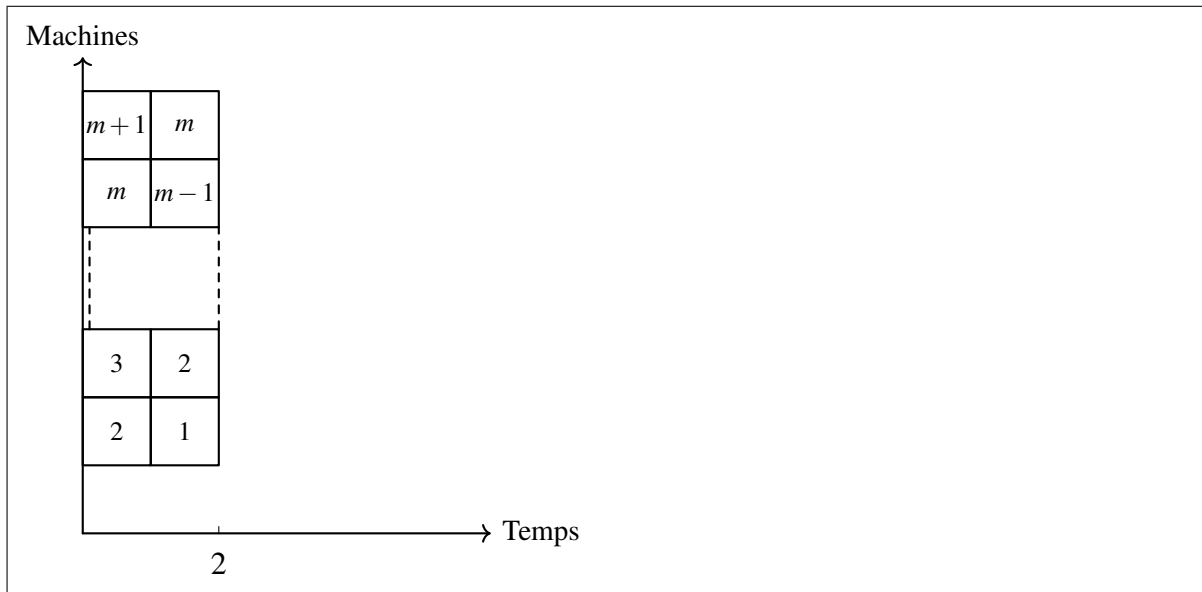


FIGURE 4.7 : Solution optimale de l'instance I avec $C_{\max} = 2$.

HEURISTIQUE 2 : ORDRE CROISSANT DU NOMBRE D'OPÉRATIONS

Considérons l'heuristique H_1 , une heuristique constructive ; elle est composée de deux étapes :

1. Ordonner les tâches dans l'ordre croissant du nombre d'opérations pour obtenir une liste L .
2. Insérer les tâches une par une dans le système flow shop le plus tôt possible suivant la liste L .

On note par $H_1(I)$ le temps final d'achèvement du traitement pour l'instance I , en appliquant l'heuristique H_1 .

Théorème 4.2.2. *Pour $m \geq 2$, on a $H_1(I) \leq m * \text{opt}(I) - 1$, et cette borne est atteinte.*

Démonstration. À partir du Théorème 4.2.1, il en résulte que

$$H_1(I) \leq m \times \text{opt}(I).$$

Nous allons maintenant montrer que

$$H_1(I) < m \times \text{opt}(I) \quad \text{pour } m \geq 2.$$

Posons

$$M = \max_{j=1,\dots,m} |T_j|,$$

et désignons par L la liste obtenue en appliquant l'heuristique H_1 . Nous allons distinguer les cas suivants :

— **Cas 1 : Il existe un indice j_1 tel que $M > |T_{j_1}|$.**

Par un raisonnement semblable à celui utilisé dans la preuve du Théorème 4.2.1, on obtient

$$\sum_{j=1}^m |T_j| \geq H_1(I).$$

Par ailleurs, on a $\text{opt}(I) \geq M$. Or, puisque $M = \max_{j=1,\dots,m} |T_j|$, nous avons $\text{opt}(I) \geq |T_j|$ pour tout $j \neq j_1$, et $\text{opt}(I) > |T_{j_1}|$. Par conséquent, nous obtenons

$$m \times \text{opt}(I) > \sum_{j=1}^m |T_j| \geq H_1(I),$$

pour tout $m \geq 2$.

— **Cas 2 : $|T_1| = \dots = |T_m|$.** Dans cette situation, on considère deux sous-cas :

1. $|T_1| = \dots = |T_m| = 1$.

Dans ce cas, chaque machine ne comporte qu'une seule opération. Il est alors immédiat de vérifier que

$$H_1(I) = \text{opt}(I).$$

2. $|T_1| = \dots = |T_m| > 1$.

Soit O un ordonnancement réalisable qui respecte l'ordre de la liste L construite selon l'heuristique H_1 . On le construit de la manière suivante (voir Figure 4.8) :

- **Machine M_1** : on ordonne le plus tôt possible toutes les opérations qui s'exécutent sur cette machine.
- **Machine M_j (avec $j = 2, \dots, m$)** : on ordonne, le plus tôt possible, les opérations sur la machine M_1 à partir de l'instant $|T_{j-1}|$.

Avec cette construction, on obtient un ordonnancement réalisable O (Figure 4.8) dont le makespan satisfait

$$C_{\max}(O) = \sum_{j=1}^m |T_j|.$$

De plus, il est clair que $C_{\max}(O) \geq H_1(I)$. On en déduit

$$m \times \text{opt}(I) \geq \sum_{j=1}^m |T_j| \geq H_1(I).$$

Nous allons maintenant montrer qu'il est possible de construire à partir de O un nouvel ordonnancement réalisable O' (Figure 4.9), avec

$$C_{\max}(O) = C_{\max}(O') + 1.$$

Pour cela, on part de l'ordonnancement O ; notons e_j la première tâche exécutée sur la machine M_j et f_j la dernière tâche exécutée sur la machine M_j .

On distingue deux cas :

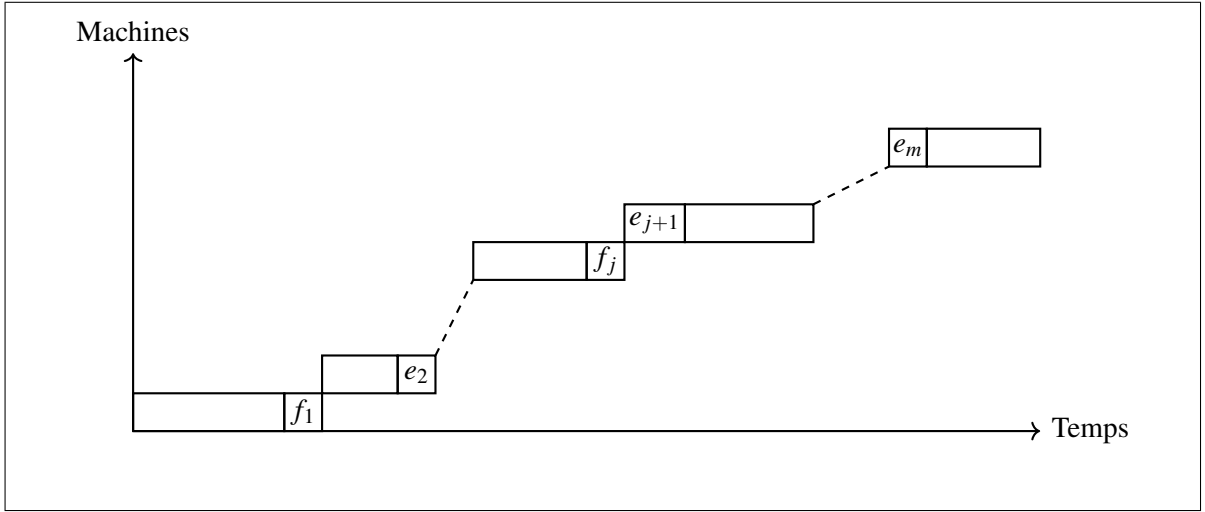


FIGURE 4.8 : Structure de l'ordonnancement O

- **Cas 1 :** Il existe une machine j telle que $e_{j+1} \neq f_j$. Dans ce cas, sans perdre la réalisabilité de l'ordonnancement O , on décale d'une unité de temps toutes les tâches prévues sur les machines $j+1$ à m . On obtient ainsi un nouvel ordonnancement O' pour lequel

$$C_{\max}(O) = C_{\max}(O') + 1.$$

Cet ordonnancement O' respecte toujours l'ordre de la liste L , mais les tâches n'y sont pas nécessairement placées au plus tôt sur chaque machine. Dès lors, on a

$$C_{\max}(O') \geq H_1(I).$$

Il s'ensuit que

$$m \times \text{opt}(I) - 1 \geq C_{\max}(O') \geq H_1(I).$$

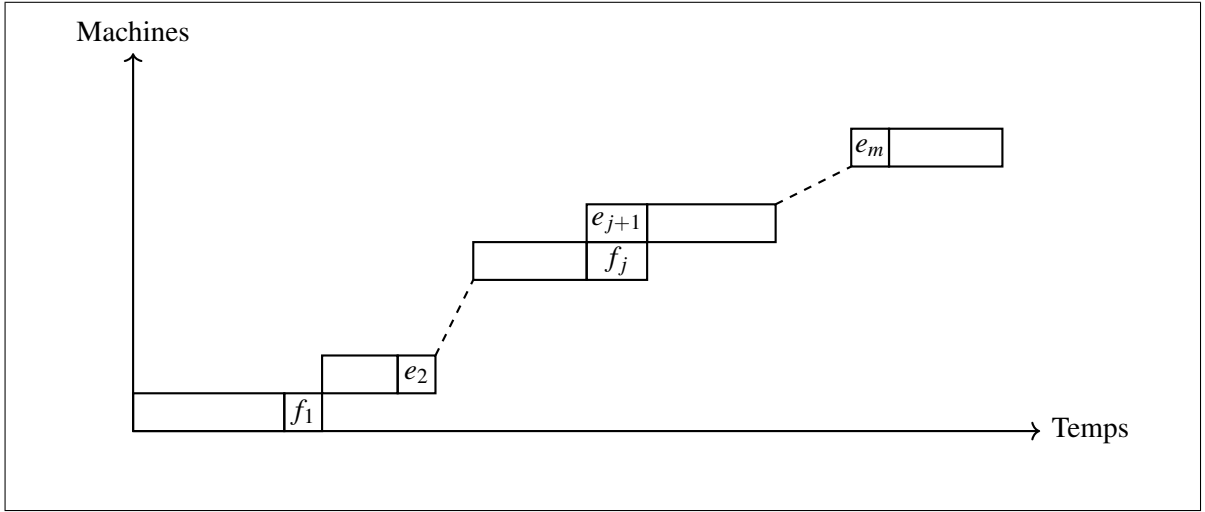


FIGURE 4.9 : Structure de l'ordonnancement O'

— **Cas 2 :** Pour tout $j \in \{1, \dots, m-1\}$, on a $e_{j+1} = f_j$. Nous allons montrer par l'absurde que ce cas est impossible.

Tout d'abord, notons que $e_j \neq f_j$ pour tout $j \in \{1, \dots, m\}$ puisque $|T_1| = \dots = |T_m| > 1$.

Par ailleurs, les tâches de l'ensemble $T_{m-1} \setminus \{f_{m-1}\}$ précèdent la tâche f_{m-1} sur la machine M_{m-1} ; elles la précèdent donc aussi dans la liste L . Or, comme $e_m = f_{m-1}$ précède f_m sur la machine M_m , il la précède également dans L . Par conséquent, $f_m \notin T_{m-1}$.

De même, les tâches de $T_{m-2} \setminus \{f_{m-2}\}$ précèdent f_{m-2} dans la liste L , et $e_{m-1} = f_{m-2}$ précède f_{m-1} sur la machine M_{m-1} . Il s'ensuit que $e_{m-1} = f_{m-2}$ précède f_{m-1} dans L . Dès lors, $T_{m-2} \cup (T_{m-1} \setminus \{f_{m-1}\})$ précèdent f_{m-1} dans L , ce qui implique $f_m \notin T_{m-2} \cup T_{m-1}$.

En poursuivant ce raisonnement, on constate finalement que

$$T_1 \cup T_2 \cup \dots \cup T_{m-2} \cup (T_{m-1} \setminus \{f_{m-1}\}) \text{ précèdent } f_{m-1} \text{ dans la liste } L.$$

D'où $f_m \notin T_1 \cup T_2 \cup \dots \cup T_{m-1}$.

Il s'ensuit que la tâche f_m ne s'exécute que sur la machine M_m , donc $n_{f_m} = 1$.

Par ailleurs, e_m précède f_m dans L , et les tâches de L sont ordonnées selon le nombre croissant d'opérations. On a alors

$$n_{f_m} \geq n_{e_m}.$$

Toutefois, $e_m = f_{m-1}$ et, puisque $|T_{m-1}| > 1$, il vient $n_{e_m} \geq 2$. On aboutit donc à la contradiction

$$1 = n_{f_m} \geq n_{e_m} \geq 2.$$

Ce second cas est ainsi impossible, ce qui achève la preuve.

Cette borne est atteinte pour l'instance I_1 comportant m machines et $n = m$ tâches, définie comme suit :

$$p_{1,j} = \begin{cases} 1 & \text{si } j = 1, \\ 0 & \text{sinon,} \end{cases}$$

et, pour tout $i = 2, \dots, m$,

$$p_{i,j} = \begin{cases} 1 & \text{si } j \in \{i-1, i\}, \\ 0 & \text{sinon.} \end{cases}$$

Si l'on retient la liste de tâches $L = (1, \dots, n)$, l'ordonnancement obtenu présente un makespan de $2m - 1$, alors que l'ordonnancement optimal a pour makespan 2. Les figures 4.10 et 4.11 illustrent ce cas.

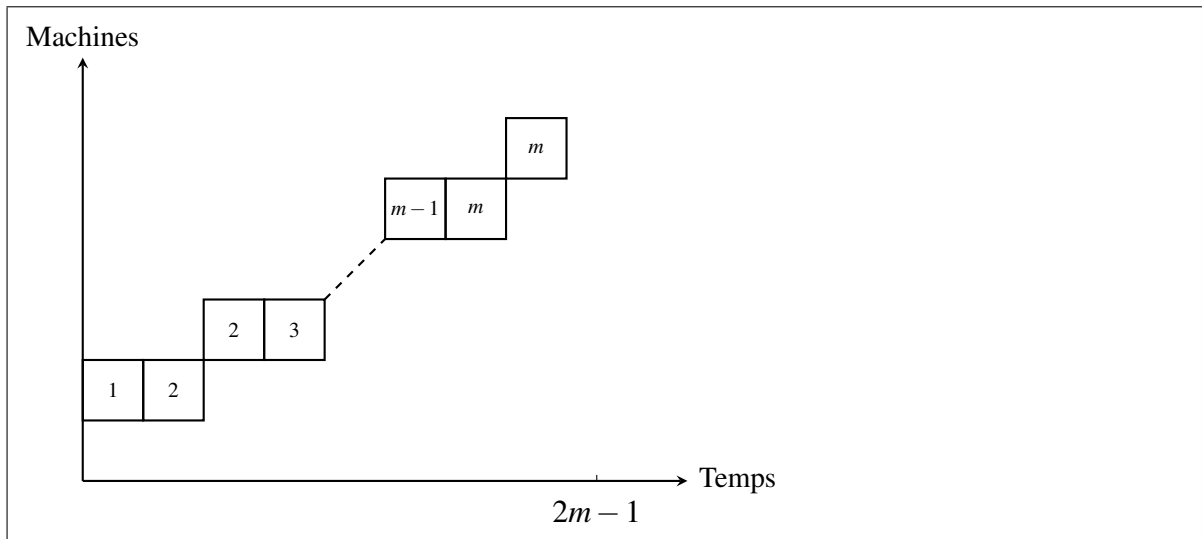


FIGURE 4.10 : Solution produite par l'heuristique H_1 .

■

HEURISTIQUE 3 : ORDRE DÉCROISSANT DU NOMBRE D'OPÉRATIONS

Considérons l'heuristique H_2 , une heuristique constructive, elle est composée de deux étapes :

1. Ordonner les tâches dans l'ordre décroissant du nombre d'opérations pour obtenir une liste L .
2. Insérer les tâches une par une dans le système flow shop le plus tôt possible suivant la liste L .

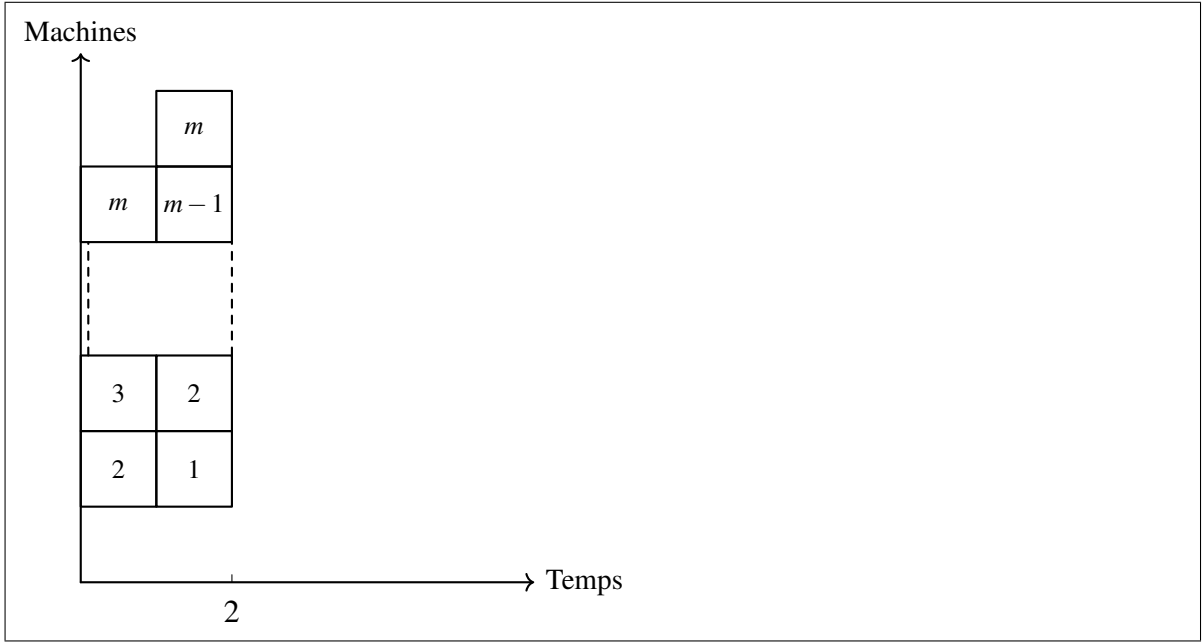


FIGURE 4.11 : Solution optimale de l'instance I_1 avec $C_{\max} = 2$.

On note par $H_2(I)$ le temps final d'achèvement du traitement pour l'instance I , en appliquant l'heuristique H_2 .

Théorème 4.2.3. Pour $m \geq 2$, on a $H_2(I) \leq m * \text{opt}(I) - 1$ et cette borne est atteinte.

Démonstration. Pour tout $m \geq 2$, on a

$$H_2(I) < m \times \text{opt}(I).$$

La démonstration s'effectue suivant une argumentation analogue à celle du Théorème 4.2.2, et n'est donc pas répétée ici.

Atteinte de la borne. Considérons l'instance I_2 ayant m machines et $n = m$ tâches, définie comme suit :

$$p_{m,j} = \begin{cases} 1 & \text{si } j = m, \\ 0 & \text{sinon,} \end{cases}$$

et, pour chaque machine $i = 1, \dots, m-1$,

$$p_{i,j} = \begin{cases} 1 & \text{si } j \in \{i, i+1\}, \\ 0 & \text{sinon.} \end{cases}$$

En prenant la liste $L = (1, \dots, n)$, on obtient un ordonnancement de makespan $C_{\max} = 2m - 1$, alors que l'ordonnancement optimal satisfait $\text{opt}(I) = 2$ (voir Figures 4.12 et 4.13). Ainsi, la borne précédente est effectivement atteinte.

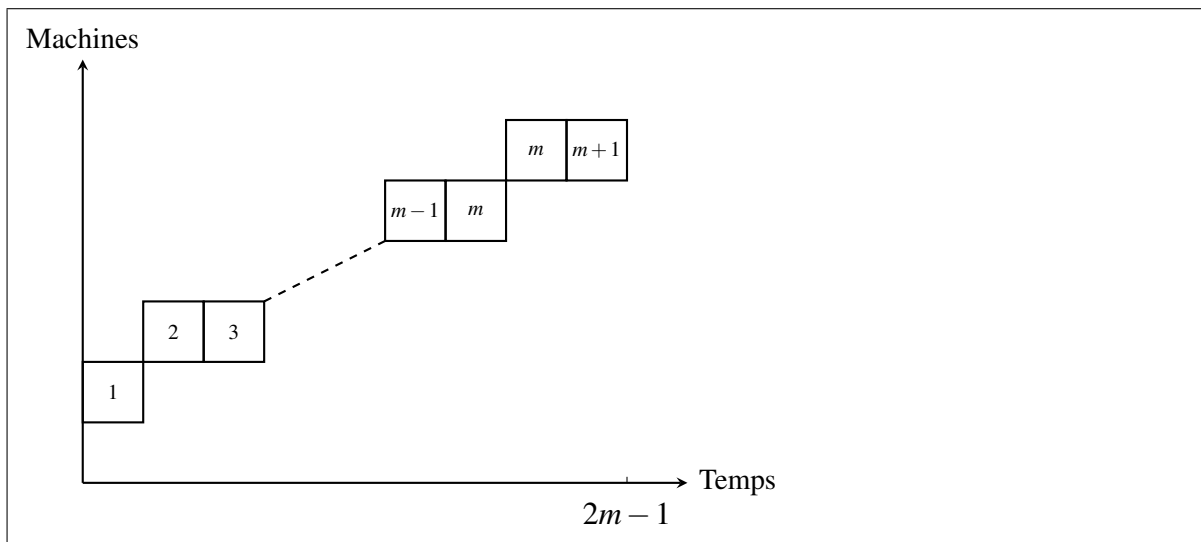


FIGURE 4.12 : Solution produite par l'heuristique H_2 .

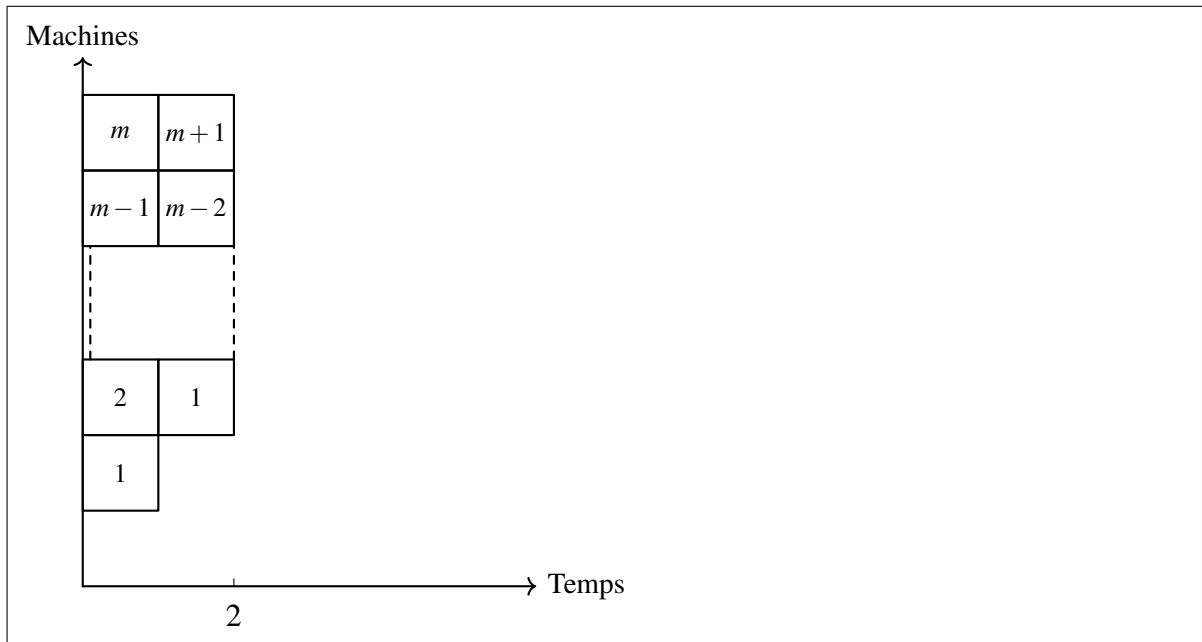


FIGURE 4.13 : Solution optimale de l'instance I_2 avec $C_{\max} = 2$.

■

4.3 CONCLUSION

Ce chapitre a permis d'approfondir la compréhension du problème de flow shop avec opérations manquantes, en clarifiant les différentes notions de permutation, notamment dans les situations où la définition classique s'avère ambiguë. Nous avons distingué les permutations faible et forte, et mis en lumière la définition alternative de permutation basée sur la discipline FIFO proposée par Pinedo. Les relations d'inclusion et de recouvrement partiel entre ces différentes classes de solutions ont été formellement établies et illustrées à l'aide d'exemples. Sur le plan théorique, nous avons démontré que, sous certaines conditions, en particulier pour les cas à deux ou trois machines que les ordonnancements respectant la discipline FIFO sont dominants pour le critère du makespan. Enfin, l'analyse du flow shop unitaire avec opérations manquantes a permis d'évaluer la performance de plusieurs heuristiques classiques,

mettant en évidence que celles-ci peuvent aboutir à des solutions sensiblement sous-optimales, notamment dans le pire des cas.

CHAPITRE V
THE IMPACT OF MISSING OPERATIONS ON THE FLOW SHOP :
PERMUTATION VS NON PERMUTATION MODES

Ce chapitre est basé sur un article publié dans la revue **RAIRO - Operations Research**.

Ouchene, R., Rebaine, D. & Baptiste, P., *The impact of missing operations on the flow shop : permutation vs non-permutation*. **RAIRO - Operations Research**, **59**(3), 1273–1293, 2025.

DOI : <https://doi.org/10.1051/ro/2025032>

ABSTRACT

The flow shop scheduling problem is traditionally studied under the assumption that the optimal solution is a permutation schedule. However, for many practical problems, this may not be the case. Since finding a flow shop schedule is significantly simpler with the permutation consideration restriction, it becomes interesting to know a priori at what extent relaxing this constraint could be beneficial. In this paper, we study the conditions under which this assumption holds, particularly in flow shops with missing operations. We focus our analysis on the makespan and total completion time. Preliminary results show that, in certain cases, non-permutation schedules can yield significantly better outcomes.

keyword : Flow shop ; permutation ; missing operations ; makespan ; total completion time ; no-wait.

5.1 INTRODUCTION

In a flow shop (FSP) environment, a set of jobs must be processed on a set of machines in a specific order. Each job consists of a set of operations. Each operation must be executed on one machine, and each machine can process only one job at a time. The two-machine flow shop with respect to the makespan criterion is solvable in $O(n \log n)$ -time JOHNSON, [1954c](#). However, when the number of machines is greater than two, the flow shop becomes strongly $\mathcal{NP}$ -hard GAREY et al., [1976b](#). Unlike the makespan criterion, the two-machine flow shop with respect to the total completion time criterion is already strongly $\mathcal{NP}$ -hard GAREY et al., [1976b](#).

Flow shop is traditionally associated with manufacturing processes, but it can also be applied to service environments such as those related to health centres. In these applications, the jobs, refer to patients, usually follow a fixed order of treatment, ranging from admission to consultation, examinations, treatments, etc. There are two main types of flow shops studied in the literature : the permutation flow shop (PFSP) and the non-permutation flow shop (NFSP). In the NFSP, the sequence of jobs may vary between different machines, resulting in a significant

increase in the space of feasible solutions compared to the PFSP, where the order of jobs is the same on each machine. In the NFSP, additional computational effort may be required. However, this effort can improve the quality of the solution. Several authors have highlighted the interest of considering non-permutation schedules in real flow shop environments PUGAZHENDHI et al., 2003c. In particular, C.-J. LIAO et al., 2006a compared the performance of schedules with and without the permutation constraint concerning criteria based on completion times and due dates. They highlighted that, for due date-based criteria, the PFSP approach leads to a significant gap, with permutation solutions performing notably worse than non-permutation solutions.

A variant of the flow shop problem is the model with missing operations (FSMOP). In the literature on flow shops with missing operations, two definitions of a missing operation exist. The first considers a missing operation as one with a negligible, but not zero, processing time POTTS et al., 1991d. The second considers the processing time as zero, allowing tasks to bypass certain machines if needed a2_3 ; a2_4. In this article, we use the second definition of a missing operation, which is particularly relevant in contexts such as healthcare. For instance, in a hospital, depending on their health condition and specific needs, some patients might not require all the typical steps in the care pathway ; they may skip certain consultations or tests. This models a flow shop environment where some operations are missing.

In the context of missing operations, the conventional definition of a permutation proves to be not suitable due to the variability in the job list on each machine. As a consequence, OUCHENE et al., 2023c investigated various interpretations of a permutation within the presence of missing operations. In flow shops with missing operations, permutation schedules are not dominant with respect to the makespan, even for three machines OUCHENE et al., 2023c. This is in contrast to flow shops without missing operations, where permutation schedules are dominant for three machines FRENCH, 1982b. Non-permutation schedules can

yield significantly improved results across various criteria C.-J. LIAO et al., 2006a. However, solving approaches for permutation flow shops are significantly simpler and faster than those for non-permutation flow shops. Therefore, it is valuable to identify, *a priori*, the extent to which relaxing this constraint is advantageous.

In this paper, we are interested in the relevance of considering non-permutation schedules. We determine in the following sections indicators that evaluate the relevance of using these types of schedules, with respect to the makespan and the total completion time, respectively.

The remainder of this paper is organized as follows. Section 1 presents a brief literature review. The problems of flow shop scheduling with missing operations and constraint are described in Section 2. In Section 3, we describe four parameters indicating the relevance of non-permutation over the permutation modes in the case of missing operations. Details on the computational experiment we conducted are presented in Section 4. A statistical analysis focused on the flow shop is conducted in Section 5. A conclusion is presented in Section 6.

5.2 REVIEW OF THE LITERATURE

The study of the flow shop problem has often focused on the development of procedures for obtaining permutation schedules, often through various exact methods (CHENG et al., 1997; IGNALL et SCHRAGE, 1965a; LADHARI et HAOUARI, 2005b). In terms of heuristics, the NEH heuristic NAWAZ et al., 1983b is widely regarded as one of the most effective in the literature, and numerous studies have proposed enhancements to this method VASILJEVIC et DANILOVIC, 2015. Among the metaheuristics applied to the permutation flow shop are tabu search EKŞIOĞLU et al., 2008, genetic algorithms CHANG et al., 2013.

Nevertheless, non-permutation schedules can lead to significantly improved results across various criteria, particularly those based on due dates C.-J. LIAO et al., 2006a. With the

advancements in computing power and the development of efficient algorithms, interest in non-permutation flow shop problems has intensified in recent years. While the literature on this topic has been reviewed exhaustively ROSSIT et al., 2018c, recent contributions include the work of BENAVIDES et RITT, 2018c, who proposed a constructive heuristic based on insertion, similar to the NEH heuristic NAWAZ et al., 1983b, complemented by a set of rapid local search heuristics, and BRUM et al., 2022c, who applied iterated greedy algorithms and an automatic algorithm configuration for minimizing total completion time.

One of the variants of the flow shop is the FSMOP. There are two distinct ways to interpret a missing operation. The first, as used by Potts et al. POTTS et al., 1991d, considers it as a negligible, non-zero duration operation, requiring each job to be processed on all the machines. The second interpretation defines it as an operation with zero processing time, allowing jobs to bypass some machines. Based on this interpretation, various approaches have been developed ; for instance, a simulated annealing algorithm to minimize the total completion time is proposed in a2_3. LEISTEN et KOLBE, 1998c propose a relaxation of the permutation. They develop an algorithm that contains an enumeration procedure to generate partial permutations. SADJADI et al., 2008c and RAMEZANIAN, SAIDI-MEHRABAD et RAHMANI, 2011b present mixed-integer linear programming models for FSMOP. HENNEBERG et NEUFELD, 2016c proposed an enhanced version of the PUGAZHENDHI et al., 2003c heuristic for FSMOP problems, using a two-step simulated annealing algorithm to optimize total completion time. More recently, RITT et ROSSIT, 2024c introduced a new heuristic that operates in two phases : the first phase identifies a high-quality permutation solution, which is then refined in the second phase by allowing non-permutation scheduling. This two-phase approach has demonstrated superior results compared to a2_16 for minimizing total completion time in flow shop settings with missing operations.

In the context of permutation flow shop scheduling with missing operations, the classical definition of a flow shop in which *the job sequence is the same over all the machines* does not fully apply. This definition is inadequate as, with missing operations, not all the machines process the same set of jobs. Three alternative definitions arised in the literature. OUCHENE et al., 2023c presented different ways to define a permutation as a total order (a chain) over the entire set of jobs. The second definition imposes a no-overtaking condition among jobs : if one job precedes another on a machine, this order must be respected all over the machines. In a third definition, a2_19 defines a permutation with respect to the machines in such a way the lists of jobs that are waiting for processing on the corresponding machines operate under the FIFO rule, meaning a job cannot overtake another one while in the queue a2_19.

The no-wait flow shop problem with missing operations is less studied in the literature. It is shown to be NP-hard in the strong sense, even for two machines SAHNI et CHO, 1979a, with respect to the makespan criterion. GLASS et al., 1999a proposed and analyzed heuristics for the two-machine no-wait flow shop problem, where some jobs only require processing on the first machine. They proposed two heuristics : the first consists in adding dummy operations with zero processing times, while the second is a two-phase heuristic, called GGL, which consists in solving the sub-problem of jobs without missing operations and inserting the remaining jobs in the schedule. These heuristics have a computation time of $O(n \log n)$ and a worst performance ratio of 2 and $4/3$, respectively.

5.3 FLOW SHOP PROBLEM WITH MISSING OPERATIONS

The flow shop problem can be described as follows. Given are $M = \{1, \dots, m\}$ a set of $m > 1$ machines and a set $J = \{J_1, J_2, \dots, J_n\}$ of $n \geq 1$ jobs. Each job is first processed on machine 1, then on machine 2, and so on until it completes processing on a machine m . So,

each job, say k , consists of m operations, $O_{k1}, \dots, O_{km}$. The processing time of operation O_{ij} is denoted by $p_{ij} \in \mathbb{N}$.

The core of the present study is about the flow shop problem with missing operations, adopting the second interpretation of a missing operation ²

5.3.1 MATHEMATICAL PROGRAMMING APPROACH

This section is devoted to the presentation of mixed-integer linear programming models flow shop scheduling problem with missing operations. Let us recall when the number of decision variables and constraints is polynomially bounded, the corresponding mathematical programming approach can be helpful in solving efficiently instances of reasonable sizes.

Parameters

- n represents the number of jobs.
- m represents the number of machines.
- K is a large constant.

Decision Variables

- T_{ij} denotes the starting time of job i on machine j .
- $C_{\max}$ denotes the value of the makespan.
- $y_{i_2 i_1 j}$ is 1 if job i_1 is processed after job i_2 on machine j , and 0 otherwise.
- $y_{i_1 i_2}$ is 1 if job i_1 is processed after job i_2 over all the machines, and 0 otherwise.

2. Let us recall a missing operation is defined in the present study as an operation with zero processing time, which permits jobs to skip certain machines.

MATHEMATICAL MODEL WITHOUT THE PERMUTATION RESTRICTION

The mathematical model, with respect to the makespan, for the non-permutation flow shop scheduling problem with missing operations is

$$\min C_{\max} \quad (1)$$

subject to

$$C_{\max} \geq p_{im} + T_{im}, \quad \forall i \in \{1, \dots, n\} \quad (2)$$

$$T_{ij} \geq p_{ij-1} + T_{ij-1} \quad \forall i \in \{1, \dots, n\}, j \in \{2, \dots, m\} \quad (3)$$

$$T_{i_2j} + p_{i_2j} - (1 - y_{i_2i_1j})K \leq T_{i_1j}, \quad \forall i_1, i_2 \in \{1, \dots, n\}, j \in \{1, \dots, m\}, i_1 \neq i_2, p_{i_2j} \neq 0, p_{i_1j} \neq 0 \quad (4)$$

$$T_{i_1j} + p_{i_1j} - y_{i_2i_1j}K \leq T_{i_2j}, \quad \forall i_1, i_2 \in \{1, \dots, n\}, j \in \{1, \dots, m\}, i_1 \neq i_2, p_{i_2j} \neq 0, p_{i_1j} \neq 0 \quad (5)$$

$$y_{i_1i_2j} + y_{i_2i_1j} = 1, \quad \forall i_1, i_2 \in \{1, \dots, n\}, j \in \{1, \dots, m\}, i_1 \neq i_2 \quad (6)$$

$$T_{i_1j} \geq 0, \quad \forall i_1 \in \{1, \dots, n\}, j \in \{1, \dots, m\} \quad (7)$$

$$y_{i_1i_2j} \in \{0, 1\}, \quad \forall i_1, i_2 \in \{1, \dots, n\}, j \in \{1, \dots, m\} \quad (8)$$

With the following details :

- Objective function (1) refers to the minimization of the makespan.
- Constraint set (2) determines the maximum completion time.
- Constraint set (3) corresponds to the constraint of the passage of the operations of the same job through the machines.
- Constraint sets (4) and (5) force to start the processing each job only when its precedent job has been completed on the same machine.
- Constraint set (6) represent the priority constraint between job pairs executed on the same machine.

With regard to the size of the above mathematical formulation, we have a total of $n^2m + 1$ variables with $n(4nm - 1)$ constraints.

Based on the above formulation, the mathematical model for the flow shop missing operations scheduling, with respect to the total completion time, is therefore as follows :

$$\min \sum_{i=1}^n (T_{im} + p_{im}) \quad (9)$$

subject to

$$(3), (4), (5), (6), (7), (8).$$

Regarding the size of the above model, there are n^2m variables and $n(4nm - m - 1)$ constraints.

MATHEMATICAL MODEL WITH THE PERMUTATION RESTRICTION

In our study, we used the first definition of permutation presented in OUCHENE et al., 2023c. Based on this work, we formulated as follows the mathematical model for the permutation flow shop scheduling problem, with missing operation, with respect to the makespan.

$$\min C_{\max} \quad (1)$$

subject to

$$(2), (3), (7),$$

$$T_{i_2j} + p_{i_2j} - (1 - y_{i_2i_1})K \leq T_{i_1j}, \quad \forall i_1 \neq i_2 \in \{1, \dots, n\}, j \in \{1, \dots, m\}, p_{i_2j} \neq 0, p_{i_1j} \neq 0 \quad (10)$$

$$T_{i_1j} + p_{i_1j} - y_{i_2i_1}K \leq T_{i_2j}, \quad \forall i_1 \neq i_2 \in \{1, \dots, n\}, j \in \{1, \dots, m\}, p_{i_2j} \neq 0, p_{i_1j} \neq 0 \quad (11)$$

$$y_{i_1i_2} + y_{i_2i_1} = 1, \quad \forall i_1 \neq i_2 \in \{1, \dots, n\} \quad (12)$$

$$y_{i_1i_2} \in \{0, 1\}, \quad \forall i_1, i_2 \in \{1, \dots, n\} \quad (13)$$

Again, regarding the size of the above two mathematical formulations, we have here a total of $2nm + 1$ variables with $n(2nm + 2n + m - 2)$ constraints.

Based on the above formulation, the mathematical model for the permutation flow shop scheduling problem with missing operation, with respect to the total completion time, is straightforward and is as below :

$$\begin{aligned} \min \quad & \sum_{i=1}^n (T_{im} + p_{im}) \\ \text{subject to} \quad & (3), (7), (10), (11), (12), (13). \end{aligned} \tag{9}$$

The description of each constraint is the same as in the previous model described in 5.3.1.

Again, with regard to the size of the above model, we have here a total of $2nm$ variables and $n(2nm + 2n - 2)$ constraints.

5.4 DESCRIPTION OF THE INDICATORS

This paper seeks to determine in which cases non-permutation schedules may be more advantageous than permutation schedules within a flow shop context, despite the increased computational demands often associated with the former. To address this, we investigate whether certain instance-related indicators can serve as reliable predictors of when non-permutation scheduling offers substantial benefits. We aim to establish, through the analysis of these indicators, a framework for discerning when non-permutation approaches might yield superior results. Before delving into our findings, we present and define the primary indicators that form the basis of this analysis.

Intuitively, we expect the number of missing operations will impact the gap between solutions with and without the permutation constraint. To investigate this point, we first considered in this section three indicators to quantify the presence of missing operations : one focusing on their number and the other two focusing on their variability. This approach is aimed to test our hypothesis that the number of missing operations is a key predictor for the gap between schedules involving permutations and those that do not. We therefore identified the following three indicators.

- The $IND_1(I)$ indicator concerns the overall percentage of missing operations.
- The $IND_2(I)$ indicator concerns the variability in the set of jobs of the number of missing operations. It measures this variability using the standard deviation of the number of missing operations in the set of jobs of instance I .
- The $IND_3(I)$ indicator concerns the variability in the set of machines of the number of missing operations. It measures this variability using the standard deviation of the number of missing operations in the set of machines of instance I .

Before introducing indicator IND_4 , let us first define a new notation.

- EM_i is the set of machines such that if $j \in EM_i$ then $p_{ij} \neq 0$.
- E_{i_1, i_2} is a set containing all pairs of machines (h, k) such that h and k are in EM_{i_1} and EM_{i_2} with $h < k$, where k is the smallest machine such that h and k are in EM_{i_1} and EM_{i_2} .

In other words, for a given pair of jobs (i_1, i_2) , with non-null operations on machine h , k represents the next machine where both jobs also have non-null operations.

The example illustrated in Figure 1 aims to clarify the notation introduced above. For job 1, the set of machines on which it has non-zero processing times is $EM_1 = \{1, 2, 4, 5\}$. For job 2, this set is $EM_2 = \{1, 3, 5\}$. For job 3, we have $EM_3 = \{2, 3, 4, 5\}$. The set E_{i_1, i_2} denotes pairs of machines where both jobs have non-zero processing times. For instance :

$E_{1,2} = \{(1,5)\}$ captures the shared machine pairs for jobs 1 and 2. $E_{1,3} = \{(2,4), (4,5)\}$ represents the common machines for jobs 1 and 3. For jobs 2 and 3, we have $E_{2,3} = \{(3,5)\}$.

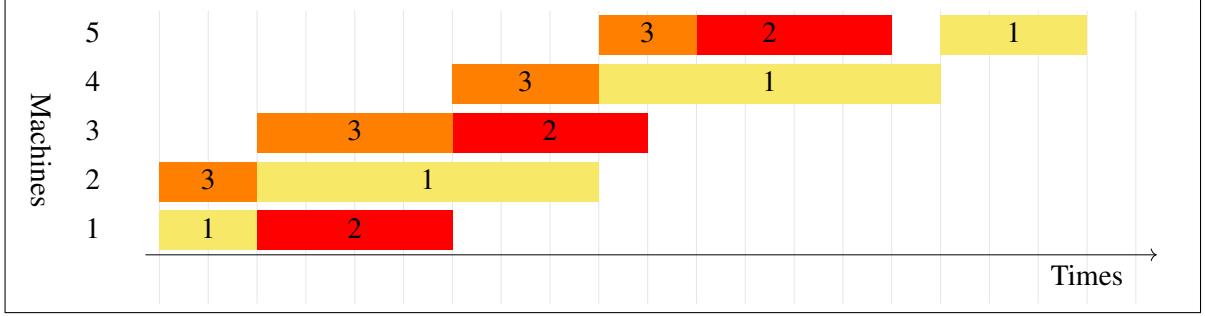


FIGURE 5.1 : Gantt chart illustrating notation

Before introducing indicator IND_4 , we state some propositions necessary for its description.

Théorème 5.4.1. *In a flow shop with missing operations involving only two jobs, i_1 and i_2 and the subset of machines, between h and k , such that $(h,k) \in E_{i_1,i_2}$, permutation schedules are dominant, with respect to the makespan, if $\sum_{j=h+1}^k p_{i_1,j} \leq \sum_{j=h}^{k-1} p_{i_2,j}$ and $\sum_{j=h+1}^k p_{i_2,j} \leq \sum_{j=h}^{k-1} p_{i_1,j}$.*

Proof. Let $T_{i_1,j}$, $C_{i_1,j}$, $T_{i_2,j}$, and $C_{i_2,j}$ be the start and completion times of jobs i_1 and i_2 on machine j , respectively, for a given schedule S . We now examine the possible job sequences on machines h and k . There are four possible orderings of these jobs :

- **Case 1 :** i_1 precedes i_2 on both machines h and k : In this case, we have : $C_{\max} = C_{i_2,k} = \max(C_{i_1,k}, C_{i_2,k'}) + p_{i_2,k}$, where k' is the machine on which job i_2 has an operation that precedes its operation on machine k . We also have : $C_{i_1,k} = \sum_{j=h}^k p_{i_1,j}$, $C_{i_2,k'} = \sum_{j=h}^{k-1} p_{i_2,j} + p_{i_1,h}$. Given that $\sum_{j=h+1}^k p_{i_1,j} \leq \sum_{j=h}^{k-1} p_{i_2,j}$, it follows that $C_{i_2,k'} \geq C_{i_1,k}$. Thus, the makespan is : $C_{\max} = \sum_{j=h}^k p_{i_2,j} + p_{i_1,h}$.

- **Case 2 :** i_1 precedes i_2 on machine h and i_2 precedes i_1 on machine k : In this case, we have : $C_{\max} = C_{i_1,k} = \max(C_{i_2,k}, C_{i_1,k'}) + p_{i_1,k}$, where k' is the machine on which job i_1 has an operation that precedes its operation on machine k . We also have : $C_{i_2,k} = \sum_{j=h}^k p_{i_2,j} + p_{i_1,h}$, $C_{i_1,k'} = \sum_{j=h}^{k-1} p_{i_1,j}$. Given that $\sum_{j=h+1}^k p_{i_1,j} \leq \sum_{j=h}^{k-1} p_{i_2,j}$, it follows that $C_{i_2,k} \geq C_{i_1,k'}$. Thus, the makespan is : $C_{\max} = \sum_{j=h}^k p_{i_2,j} + p_{i_1,h} + p_{i_1,k}$.
- **Case 3 :** i_2 precedes i_1 on both machines h and k : Following similar reasoning, we find : $C_{\max} = \sum_{j=h}^k p_{i_1,j} + p_{i_2,h}$.
- **Case 4 :** i_2 precedes i_1 on machine h and i_1 precedes i_2 on machine k : Similarly, we find : $C_{\max} = \sum_{j=h}^k p_{i_1,j} + p_{i_2,h} + p_{i_2,k}$.

Given that the values $p_{i_2,k}$, $p_{i_2,h}$, $p_{i_1,k}$, and $p_{i_1,h}$ are strictly positive, the minimum makespan occurs when i_1 precedes i_2 on both machines h and k , or i_2 precedes i_1 on both machines h and k . Hence, the permutation schedule is dominant. ■

Théorème 5.4.2. *In a flow shop with missing operations involving only two jobs, i_1 and i_2 and the subset of machines, between h and k , such that $(h,k) \in E_{i_1,i_2}$, permutation schedules are dominant, with respect to the makespan, if $\sum_{j=h+1}^k p_{i_1,j} > \sum_{j=h}^{k-1} p_{i_2,j}$ and $\sum_{j=h+1}^k p_{i_2,j} > \sum_{j=h}^{k-1} p_{i_1,j}$.*

Proof. Using the same principles as in the previous proof, we examine the four possible job sequences on machines h and k .

- **Case 1 :** i_1 precedes i_2 on both machines h and k : $C_{\max} = C_{i_2,k} = \max(C_{i_1,k}, C_{i_2,k'}) + p_{i_2,k}$, where k' is the machine on which job i_2 has an operation preceding its operation on machine k . We also have $C_{i_1,k} = \sum_{j=h}^k p_{i_1,j}$ and $C_{i_2,k'} = \sum_{j=h}^{k-1} p_{i_2,j} + p_{i_1,h}$. Given that $\sum_{j=h+1}^k p_{i_1,j} > \sum_{j=h}^{k-1} p_{i_2,j}$, it follows that $C_{i_2,k'} \leq C_{i_1,k}$. Thus, the makespan is $C_{\max} = \sum_{j=h}^k p_{i_1,j} + p_{i_2,k}$.

- **Case 2 :** i_1 precedes i_2 on machine h and i_2 precedes i_1 on machine k : $C_{\max} = C_{i_1,k} = \max(C_{i_2,k}, C_{i_1,k'}) + p_{i_1,k}$, where k' is the machine on which job i_1 has an operation preceding its operation on machine k . We also have $C_{i_2,k} = \sum_{j=h}^k p_{i_2,j} + p_{i_1,h}$ and $C_{i_1,k'} = \sum_{j=h}^{k-1} p_{i_1,j}$. Given that $\sum_{j=h+1}^k p_{i_2,j} > \sum_{j=h}^{k-1} p_{i_1,j}$, it follows that $C_{i_2,k} > C_{i_1,k'}$. Thus, the makespan is $C_{\max} = \sum_{j=h}^k p_{i_2,j} + p_{i_1,h} + p_{i_1,k}$.
- **Case 3 :** i_2 precedes i_1 on both machines h and k : Following similar reasoning, we find $C_{\max} = \sum_{j=h}^k p_{i_2,j} + p_{i_1,k}$.
- **Case 4 :** i_2 precedes i_1 on machine h and i_1 precedes i_2 on machine k : Similarly, $C_{\max} = \sum_{j=h}^k p_{i_1,j} + p_{i_2,h} + p_{i_2,k}$.

Given that the values $p_{i_2,k}$, $p_{i_2,h}$, $p_{i_1,k}$, and $p_{i_1,h}$ are strictly positive, the minimum makespan occurs when i_1 precedes i_2 on both machines h and k , or i_2 precedes i_1 on both machines h and k . Hence, the permutation schedule is dominant. ■

Théorème 5.4.3. *In a flow shop with missing operations involving only two jobs, i_1 and i_2 and the subset of machines, between h and k , such that $(h,k) \in E_{i_1,i_2}$, permutation schedules are not dominant, with respect to the makespan, if and only if $\sum_{j=h}^k p_{i_2,j} < \sum_{j=h+1}^{k-1} p_{i_1,j} + \min(p_{i_2,h}, p_{i_2,k})$.*

Proof. We begin by showing that if permutation schedules are not dominant, then $\sum_{j=h}^k p_{i_2,j} < \sum_{j=h+1}^{k-1} p_{i_1,j} + \min(p_{i_2,h}, p_{i_2,k})$. From Proposition 3.1 and Proposition 3.2, if permutation schedules are not dominant, then one of the following conditions holds : $\sum_{j=h+1}^k p_{i_1,j} > \sum_{j=h}^{k-1} p_{i_2,j}$ or $\sum_{j=h+1}^k p_{i_2,j} > \sum_{j=h}^{k-1} p_{i_1,j}$. Without loss of generality, assume $\sum_{j=h+1}^k p_{i_1,j} > \sum_{j=h}^{k-1} p_{i_2,j}$. We now examine the possible job sequences on machines h and k . There are four cases to consider :

- **Case 1.** i_1 precedes i_2 on both machines h and k .

In this scenario, the makespan is given by $C_{\max} = C_{i_2,k} = \max(C_{i_1,k}, C_{i_2,k'}) + p_{i_2,k}$, where k' is the machine on which job i_2 has an operation immediately preceding its operation on machine k . Furthermore, $C_{i_1,k} = \sum_{j=h}^k p_{i_1,j}$, $C_{i_2,k'} = \sum_{j=h}^{k-1} p_{i_2,j} + p_{i_1,h}$. Since $\sum_{j=h+1}^k p_{i_1,j} > \sum_{j=h}^{k-1} p_{i_2,j}$, it follows that $C_{i_2,k'} \leq C_{i_1,k}$. Hence, $C_{\max} = \sum_{j=h}^k p_{i_1,j} + p_{i_2,k}$.

— **Case 2.** i_1 precedes i_2 on machine h and i_2 precedes i_1 on machine k .

Here, the makespan is $C_{\max} = C_{i_1,k} = \max(C_{i_2,k}, C_{i_1,k'}) + p_{i_1,k}$. Additionally, we have $C_{i_2,k} = \sum_{j=h}^k p_{i_2,j} + p_{i_1,h}$, $C_{i_1,k'} = \sum_{j=h}^{k-1} p_{i_1,j}$. We distinguish two sub-cases :

— **Sub-case 2.1 :** If $\sum_{j=h}^k p_{i_2,j} > \sum_{j=h+1}^{k-1} p_{i_1,j}$, then $C_{\max} = \sum_{j=h}^k p_{i_2,j} + p_{i_1,k} + p_{i_1,h}$.

— **Sub-case 2.2 :** If $\sum_{j=h}^k p_{i_2,j} \leq \sum_{j=h+1}^{k-1} p_{i_1,j}$, then $C_{\max} = \sum_{j=h}^k p_{i_1,j}$.

— **Case 3.** i_2 precedes i_1 on both machines h and k .

By a similar reasoning to Case 1, the makespan in this situation is $C_{\max} = \sum_{j=h}^k p_{i_1,j} + p_{i_2,h}$.

— **Case 4.** i_2 precedes i_1 on machine h and i_1 precedes i_2 on machine k .

Analogously, the makespan is $C_{\max} = \sum_{j=h}^k p_{i_1,j} + p_{i_2,h} + p_{i_2,k}$.

Because permutation schedules are not dominant, it must be the case that i_1 precedes i_2 on machine h and i_2 precedes i_1 on machine k , which yields the minimal makespan. Therefore, we obtain $\sum_{j=h}^k p_{i_2,j} + p_{i_1,h} + p_{i_1,k} < \sum_{j=h}^k p_{i_1,j} + p_{i_2,k}$. In addition, $\sum_{j=h}^k p_{i_2,j} + p_{i_1,h} + p_{i_1,k} < \sum_{j=h}^k p_{i_1,j} + p_{i_2,h}$. This is equivalent to

$$\sum_{j=h}^k p_{i_2,j} < \sum_{j=h+1}^{k-1} p_{i_1,j} + \min(p_{i_2,h}, p_{i_2,k}),$$

which implies that, if the permutation is not dominant, the above condition must be satisfied.

We now show that if $\sum_{j=h}^k p_{i_2,j} < \sum_{j=h+1}^{k-1} p_{i_1,j} + \min(p_{i_2,h}, p_{i_2,k})$, then permutation schedules are not dominant. Using the same reasoning, we have four possible cases :

— **Case 1.** i_1 precedes i_2 on both machines h and k .

Here, the makespan is

$$C_{\max} = C_{i_2,k} = \max(C_{i_1,k}, C_{i_2,k'}) + p_{i_2,k} = \max\left(\sum_{j=h}^k p_{i_1,j}, \sum_{j=h}^{k-1} p_{i_2,j} + p_{i_1,h}\right) + p_{i_2,k}.$$

Since $\sum_{j=h}^k p_{i_2,j} \leq \sum_{j=h+1}^{k-1} p_{i_1,j} + \min(p_{i_2,h}, p_{i_2,k})$, we obtain $\sum_{j=h}^{k-1} p_{i_2,j} < \sum_{j=h+1}^{k-1} p_{i_1,j} < \sum_{j=h+1}^k p_{i_1,j}$. It follows that $C_{i_2,k'} < C_{i_1,k}$. Hence, $C_{\max} = \sum_{j=h}^k p_{i_1,j} + p_{i_2,k}$.

— **Case 2.** i_1 precedes i_2 on machine h and i_2 precedes i_1 on machine k .

The makespan is $C_{\max} = \max\left(\sum_{j=h}^k p_{i_1,j}, \sum_{j=h}^k p_{i_2,j} + p_{i_1,k} + p_{i_1,h}\right)$.

— **Case 3.** i_2 precedes i_1 on both machines h and k .

By analogous reasoning, $C_{\max} = \sum_{j=h}^k p_{i_1,j} + p_{i_2,h}$.

— **Case 4.** i_2 precedes i_1 on machine h and i_1 precedes i_2 on machine k .

Similarly, $C_{\max} = \sum_{j=h}^k p_{i_1,j} + p_{i_2,h} + p_{i_2,k}$.

Given that $\sum_{j=h}^k p_{i_2,j} < \sum_{j=h+1}^{k-1} p_{i_1,j} + \min(p_{i_2,h}, p_{i_2,k})$, we deduce $\sum_{j=h}^k p_{i_1,j} + p_{i_2,k} > \sum_{j=h}^k p_{i_2,j}$, and $\sum_{j=h}^k p_{i_1,j} + p_{i_2,k} > \sum_{j=h}^k p_{i_2,j} + p_{i_1,k} + p_{i_1,h}$. Hence, $\sum_{j=h}^k p_{i_1,j} + p_{i_2,k} > \max\left(\sum_{j=h}^k p_{i_1,j}, \sum_{j=h}^k p_{i_2,j} + p_{i_1,k} + p_{i_1,h}\right)$.

On the other hand, since $\sum_{j=h}^k p_{i_2,j} < \sum_{j=h+1}^{k-1} p_{i_1,j} + p_{i_2,h}$, we get $\sum_{j=h}^k p_{i_2,j} + p_{i_1,h} + p_{i_1,k} < \sum_{j=h}^k p_{i_1,j} + p_{i_2,h}$.

It follows that the minimum makespan occurs in Case 2, where

$$C_{\max} = \max\left(\sum_{j=h}^k p_{i_1,j}, \sum_{j=h}^k p_{i_2,j} + p_{i_1,k} + p_{i_1,h}\right).$$

Therefore, permutation schedules are not dominant if the stated condition is satisfied. ■

Let us point out Proposition 5.4.3 is a necessary and sufficient condition to produce a non-permutation schedule that performs better than a permutation schedule in a flow shop involving only the two jobs (i_1, i_2) between machines h and k , where $(h, k) \in E_{i_1, i_2}$ ³. This condition, based solely on the processing time of the jobs, makes it possible to quickly obtain for each instance I the number $NB(I)$ ⁴, Denoting the number of job pairs that can potentially improve the completion time if they are interchanged in their processing order. To this end, we have used this information to describe the fourth indicator, denoted hereafter by $IND_4(I)$, as follows :

$$IND_4(I) = \frac{NB(I)}{A(I)} \times 100,$$

where $A(I)$ denotes the total number of job pairs in instance I .

For a given instance I , with n jobs, $A(I)$ is calculated as follows :

$$A(I) = \frac{n(n-1)}{2}.$$

3. It is worth mentioning that, in a no-wait environment, a necessary condition for the existence of non-permutation schedules in a flow shop with missing operations is presented in Appendix A. With the no-wait constraint, reordering jobs across the machines may lead to an invalid schedule. This contrasts with the classical flow shop problem, where job pairs can be swapped within a machine without affecting the feasibility of the schedule, as it is possible to construct a valid schedule with any job sequence on a machine. More details are presented in Appendix A.

4. To determine $NB(I)$, we proceed as follows : for every pair of jobs (i, j) , we verify, for each pair of machines in $E_{i,j}$, if the condition stated in Proposition 5.4.3 is met at least once. The worst-case time complexity for this procedure is therefore $(\frac{m(m-1)}{2} - m + 1) \times \frac{n(n-1)}{2} = O(m^2n^2)$.

5.5 COMPUTATIONAL STUDY

We conducted an experimental study to investigate the effectiveness of certain instance indicators as predictors in determining whether permutation schedules are disadvantageous. Let us emphasize the aim of our study was not to develop an effective method to solve flow shop scheduling problems. This evaluation is crucial when it comes to opt for a heuristic or metaheuristic approach, where restricting the search to permutations is more straightforward compared to non-permutation approaches

In this study, we used the IBM's Cplex 20.0.1 solver for the implementation of the mathematical models we designed in the previous sections. We conducted a computational study using 11 DellPowerEdge R740 machines, each outfitted with 512GB of memory and a pair of Intel Xeon Gold 6258R CPUs running at 2.70GHz. We addressed both permutation and non-permutation modes, focusing on the criteria of the makespan and the total completion time.

This study incorporates the following key terms and abbreviations we used throughout this section.

- % : Percentage of missing operations, indicating the proportion of tasks that do not require processing on certain machines.

- E_1 : Gap between permutation and non-permutation schedules with respect to the makespan $C_{\max}$ in a flow shop. It is calculated as :

$$E_1 = \frac{C_{\max}(\text{permutation}) - C_{\max}(\text{non-permutation})}{C_{\max}(\text{non-permutation})} \times 100$$

- E_2 : Gap between permutation and non-permutation schedules with respect to the total completion time in a flow shop. It is calculated similarly :

$$E_2 = \frac{\text{Total completion time (permutation)} - \text{Total completion time (non-permutation)}}{\text{Total completion time (non-permutation)}} \times 100$$

- RPD (Relative Proportion of Deviations) : The ratio of instances where the non-permutation schedule outperforms the permutation schedule to the total number of instances analyzed. It is calculated as :

$$\text{RPD} = \frac{\text{Number of instances where non-permutation outperforms permutation}}{\text{Total number of instances}} \times 100$$

In this study, we aim to evaluate the efficiency of instance indicators as well as the impact of variability in the number of jobs, the number of machines, the variability of processing times, and the distribution of missing operations on the gap between permutation and non-permutation solutions for flow shop problems.

We utilized two groups of instances :

- The first group consists of 800 instances with the number of jobs set to 10 or 15, and the number of machines set to 5 or 10. The processing times are randomly and uniformly generated within four intervals : $[1, 10]$, $[1, 50]$, $[1, 70]$, and $[1, 100]$. In this group, the missing operations are uniformly distributed, with percentages of missing operations varying at 0%, 10%, 30%, 50%, and 70%. This group is further divided into :

- Group 1.1 : Instances with 0% missing operations, containing a total of 160 instances.
- Group 1.2 : Instances with missing operations, containing a total of 640 instances.
- The second group includes 320 instances and differs from the first in the way missing operations are considered. Unlike the first group, missing operations in this set are not generated in a fully uniform manner. We define four cases :
 - (0) Missing operations are uniformly distributed.
 - 1. No missing operations on the first machine.
 - 2. No missing operations on the last machine.
 - 3. No missing operations on both the first and last machines.

In this second group, we consider task numbers of 10 and 15, with 5 and 10 machines. The processing times are uniformly generated within the interval $[1, 100]$.

This setup allows us to analyze the effect of different factors on the performance and the gap between permutation and non-permutation schedules in flow shop problems. We analyzed the collected data to assess the relevance of the instance indicators across various scenarios of E_1 , E_2 , using Python's statistical library, "Lib/statistics.py".

We analyzed the collected data to evaluate the relevance of the instance indicators across various scenarios of E_1 and E_2 . Using Python's statistical library, "Lib/statistics.py", we applied ANOVA and Spearman correlation tests with a p-value threshold set at 0.05.

5.6 RESULTS AND DISCUSSION

In this section, we conduct a statistical analysis to explore the potential relationship between E_1 and E_2 with IND_1 , IND_2 , IND_3 , and IND_4 introduced in Section 3. In addition to these variables, we also take into account the variability in the number of tasks, the number of

machines, the intervals for processing times, and the distribution of missing operations across the instances.

We begin our analysis with the makespan gap E_1 , followed by our findings on E_2 .

5.6.1 STATISTICAL ANALYSIS WITH RESPECT TO THE MAKESPAN

Only instances that reached optimality within 6000 seconds for both permutation and non-permutation schedules are included in the subsequent analysis. In Group 1, the average gap between permutation and non-permutation schedules for the makespan (E_1) was 1.2 when no missing operations were present, increasing to 2.7 when missing operations were introduced. These results highlight the significant influence of missing operations on the gap between permutation and non-permutation schedules. In the following analysis, we focus on Groups 1.2 and 2, which include missing operations, to examine the variability and impact of these operations on the observed gap E_1 . The table below provides a detailed summary of the results for these groups.

TABLE 5.1 : Summary of Group 1.2 and Group 2 for E_1

Metric	Group 1.2	Group 2
% Instances solved (Permutation)	98.90%	95.93%
% Instances solved (Non-Permutation)	87.73%	71.56%
Avg. Gap E_1	2.70	4.18
Std. Dev. Gap E_1	3.50	4.24
Max Gap E_1	20.94	17.50
Min Gap E_1	0	0
Avg. Resolution Time (Permutation)	7.76 s	3.42 s
Avg. Resolution Time (Non-Permutation)	168.02 s	283.52 s
RDP	55.8	70.3

Table 5.1 demonstrates notable differences between Group 1.2 and Group 2 in the classical flow shop with makespan minimization. While both groups show a significant gap

between permutation and non-permutation schedules, Group 2 exhibits a larger average gap and standard deviation compared to Group 1.2. Additionally, Group 2 has a lower percentage of instances solved to optimality and a longer execution time for non-permutation schedules. This suggests that Group 2 instances are generally more challenging to solve. For both groups, the execution time for non-permutation schedules is significantly higher than for permutation schedules. However, despite the longer execution times for non-permutation schedules, the maximum gap between permutation and non-permutation solutions is quite large in both groups. The variability in the gap, as shown by the standard deviation, is also high in both groups. This suggests that for a certain number of instances, non-permutation schedules may be preferable despite the longer execution times. To address this trade-off, we examine instance indicators that will help us determine whether it is preferable to use non-permutation schedules or stick with permutation schedules for certain instances.

We begin our analysis with Group 1.2. In this group, 87.73% of instances were solved to optimality in both the permutation and non-permutation models and will be considered in the subsequent study. The statistical study that follows will examine the relationship between the four indicators IND_1 , IND_2 , IND_3 , and IND_4 , as well as the variability in the number of tasks, the number of machines, and the intervals of processing times with the gap E_1 . We aim to understand how these factors influence E_1 the gap between the flow shop with and without permutation constraints with respect to the makespan $C_{\max}$.

We begin by visualizing the quantitative indicators IND_3 , IND_2 , and IND_4 using scatter plots (indicator IND_1 is not visualized as it is not quantitative).

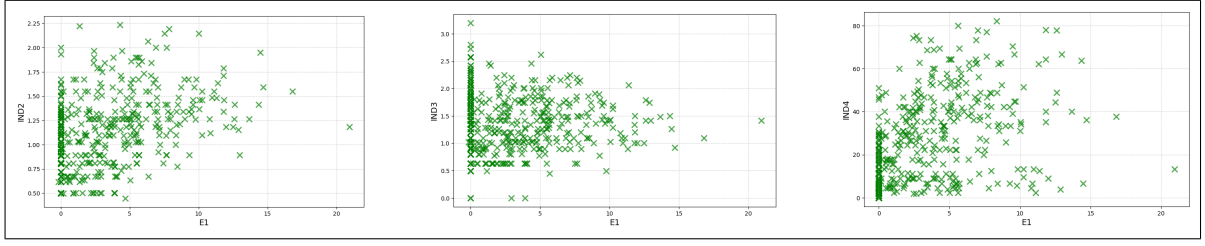


FIGURE 5.5 : Relationship between E_1 and IND_2 , IND_3 , and IND_4

These visual representations suggest some level of interaction between the indicators and the gap E_1 , particularly with IND_4 , where a greater spread in values indicates a potential correlation with larger makespan gaps, as shown in Figure 5.5.

We now proceed with the application of statistical correlation tests to verify these observations and assess the significance of the relationships between these indicators and E_1 . We will perform a Spearman correlation test, which will provide a more concrete statistical analysis of the relationships observed in the plots. The results of the Spearman correlation test are presented in Table 5.2.

TABLE 5.2 : Spearman correlation results

Variable	Spearman's ρ value	P-value
IND_1	-0.22	9.64×10^{-08}
IND_2	0.39	1.02×10^{-21}
IND_3	0.0111	0.7935
IND_4	0.63	2.34×10^{-62}

The Spearman correlation test shows a strong positive correlation between E_1 and IND_4 with a ρ value of 0.63, indicating that higher values of IND_4 are associated with larger E_1 values. Additionally, IND_2 also shows a moderate positive correlation with E_1 , as highlighted by a ρ value of 0.39. On the other hand, IND_3 does not show a significant correlation with E_1 , as indicated by the very low ρ value and high p-value. Finally, IND_1 exhibits a weak negative

correlation with E_1 ($\rho = -0.22$), suggesting that higher percentages of missing operations are associated with slightly smaller E_1 values.

In summary, the statistical analyses confirm our initial visual assessments. The Spearman correlation test provides strong evidence of the relationships between E_1 and IND_2 , as well as E_1 and IND_4 . The strong correlation observed with IND_4 suggests that for instances with high indicator values, adopting non-permutation schedules is recommended to reduce completion times. Additionally, the moderate correlation with IND_2 suggests that the dispersion of missing operations across tasks impacts the gap between permutation and non-permutation schedules.

Given the statistical significance of the Spearman test results, we have categorized the variables IND_2 , IND_3 , and IND_4 into quartiles. This categorization is intended to facilitate the comparison between IND_1 , IND_2 , IND_3 , and IND_4 , as IND_2 , IND_3 , and IND_4 are quantitative variables, unlike IND_1 , which is categorical.

The box plots in Figure 5.10 illustrate the distribution of E_1 across different categories of the indicators IND_1 , IND_2 , IND_3 , and IND_4 . With regard to IND_1 , instances with a very high percentage of missing operations (70%) exhibit small E_1 gaps, while intermediate percentages (30% and 50%) show greater variability and larger gaps. To better understand this trend, we calculated the average values of IND_4 for each IND_1 category. The results indicate that for 10% missing operations, the average IND_4 is 27.08. It reaches a maximum at 30% (35.54), then decreases to 19.90 at 50%, and drops significantly to 3.75 at 70%. These observations suggest that intermediate percentages of missing operations encourage task interchanges, contributing to larger E_1 gaps. In contrast, very high percentages restrict interchange flexibility, resulting in smaller gaps.

When it comes to IND_2 , the analysis reveals that increased variability in missing operations across tasks corresponds to larger gaps in E_1 ; however, variability across machines

within IND_3 has minimal impact on the gap. At last, IND_4 exerts the strongest influence on E_1 ; higher values, which represent a greater number of interchangeable task pairs, are linked to significantly larger gaps between permutation and non-permutation schedules.

The next hypothesis contends the mean values of IND_1 , IND_2 , IND_3 , and IND_4 differ significantly across various E_1 categories. ANOVA tests were performed to investigate this hypothesis.

TABLE 5.3 : ANOVA test results for E_1 based on IND_1 , IND_2 , IND_3 , and IND_4

Indicator	F-Statistic	P-Value
IND_1	9.29	5.29×10^{-6}
IND_2	38.62	1.11×10^{-22}
IND_3	1.16	0.325
IND_4	66.61	8.44×10^{-37}

The ANOVA results show that IND_4 has the most significant effect on E_1 , with a highly significant p-value of 9.78×10^{-41} , confirming its strong influence on the makespan gap. Similarly, IND_2 demonstrates a highly significant impact on E_1 (p-value 1.11×10^{-22}). While IND_1 shows a significant effect (p-value 5.29×10^{-6}), its impact is less pronounced compared to IND_4 and IND_2 . Lastly, IND_3 does not have a statistically significant effect on E_1 , with a p-value of 0.325, indicating that the distribution of missing operations across machines, as represented by IND_3 , plays a minimal role in the discrepancies between permutation and non-permutation schedules, in contrast to the stronger impacts of task pair interchangeability (IND_4).

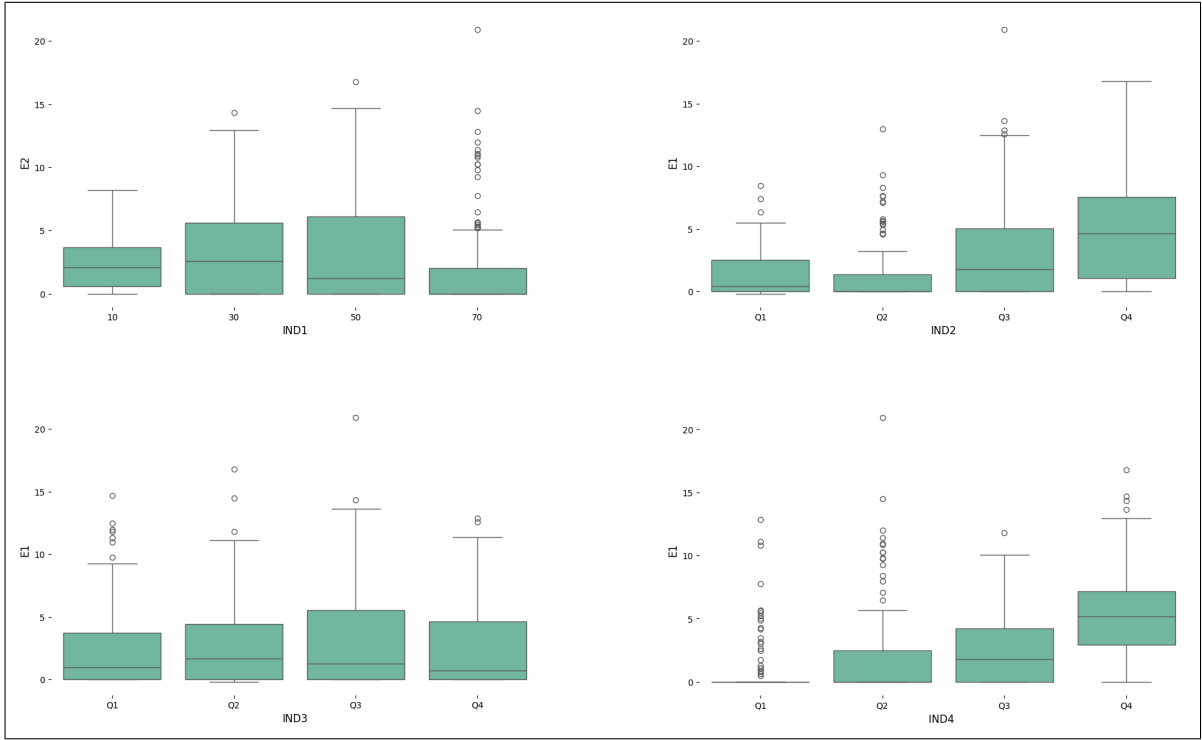


FIGURE 5.10 : Distribution of E_1 across different categories of IND_1 , IND_2 , IND_3 , and IND_4

In summary, we used both the Spearman and ANOVA tests to assess the effect of four distinct indicators on the variable E_1 . The different tests, along with the visualizations, revealed a strong correlation between IND_4 and the gap between permutation and non-permutation solutions, E_1 . This suggests that instances with a higher number of interchangeable task pairs, as measured by IND_4 , tend to generate a larger gap between the permutation and non-permutation schedules.

We now proceed with the analysis of the impact of the number of machines, the number of jobs, and the variability in processing time intervals on the makespan gap E_1 . The following box plots illustrate the results.

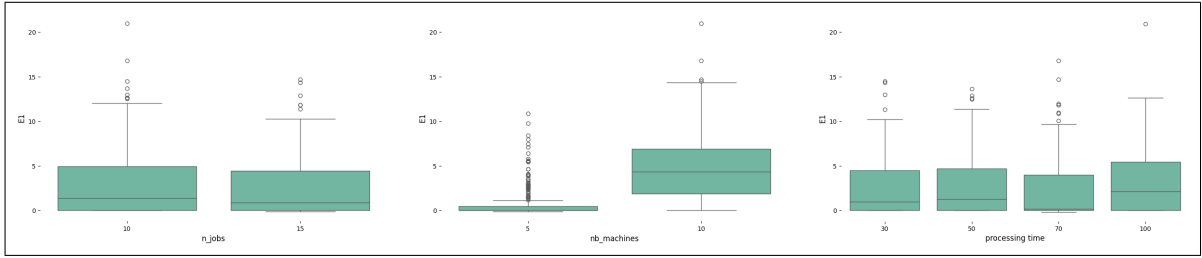


FIGURE 5.14 : Impact of the number of machines, jobs, and processing time intervals on E_1

The box plots in Figure 5.14 reveal several key observations regarding the impact of the number of machines, processing time intervals, and the number of jobs on the makespan gap E_1 :

- Impact of the number of machines : Instances with 10 machines exhibit a significantly larger gap E_1 compared to instances with 5 machines.
- Impact of processing time intervals : The variability in processing time intervals does not have a major effect on the gap E_1 .
- Impact of the number of jobs : Instances with 15 jobs tend to have an E_1 gap comparable to those with 10 jobs.

To further validate the impact of the number of machines and the number of jobs on the makespan gap E_1 , we conducted ANOVA tests. These tests allow us to assess whether the observed differences in E_1 between different groups (based on the number of machines, jobs, and processing time intervals) are statistically significant. The results of the ANOVA tests are presented below :

TABLE 5.4 : ANOVA test results for E_1 based on the number of machines, jobs, and processing time intervals

Factor	F-Statistic	P-Value
Number of jobs	0.99	0.321
Number of machines	245.83	3.69×10^{-46}
Processing time intervals	0.89	0.445

As shown in Table 5.4, the ANOVA test for the number of machines yielded a very high F-statistic value and an extremely small p-value, indicating that the number of machines significantly influences the gap E_1 . Conversely, the number of jobs and processing time intervals do not exhibit a statistically significant effect on E_1 .

We now proceed with the analysis of Group 2. 71.56% of the instances were solved to optimality within 6000 seconds in both the permutation and non-permutation models and will be considered in the subsequent analysis. In this group, we aim to investigate whether the arrangement of missing operations affects the makespan gap E_1 between permutation and non-permutation schedules. This study seeks to assess the extent to which the specific configuration of missing operations influences the difference in performance between the two scheduling strategies.

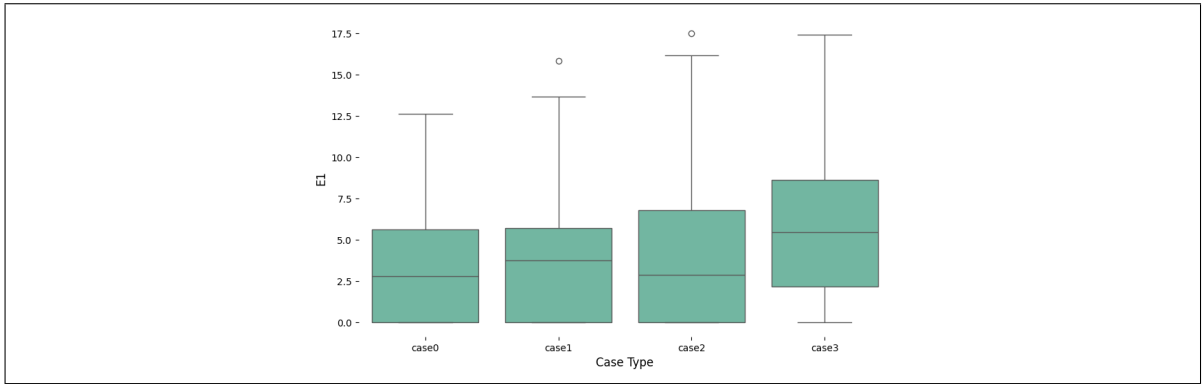


FIGURE 5.15 : Impact of case types on the makespan gap E_1

As we can see from the box plot of Figure 5.15, case 3, where no missing operations are allowed on both the first and last machines, tends to have the largest median gap E_1 and shows higher variability in comparison to the other cases. To better understand why Case 3 stands out, we calculated the average IND_4 values for each case type, as previous analyses have shown that higher values of IND_4 are strongly associated with larger E_1 gaps. The results indicate that

Case 0 (missing operations are uniformly distributed) has an average IND_4 of 17.85, while Case 1 (no missing operations on the first machine) and Case 2 (no missing operations on the last machine) exhibit slightly higher averages of 23.67 and 23.31, respectively. In contrast, Case 3 shows the highest average IND_4 value of 33.73, which aligns with its distinctly larger median E_1 gap.

These findings indicate that the various positions of missing operations significantly influence the gap between permutation and non-permutation schedules.

We performed an ANOVA test to examine the impact of the different case types on the makespan gap E_1 . The results, with an F -statistic of 2.89 and a p-value of 0.036, indicate that the configuration of missing operations has a statistically significant effect on E_1 . This suggests that the arrangement of missing operations has a statistically significant effect on the makespan gap E_1 .

5.6.2 STATISTICAL ANALYSIS WITH RESPECT TO THE TOTAL COMPLETION TIME

Only instances that reached optimality within 6000 seconds for both permutation and non-permutation schedules are included in the subsequent analysis. In Group 1, the average gap between permutation and non-permutation schedules for the total completion time (E_2) was 0.72 when no missing operations were present, increasing to 1.9 when missing operations were introduced. These results underscore the significant influence of missing operations on the gap between permutation and non-permutation schedules.

In the following analysis, we focus on Groups 1.2 and 2, which include missing operations, to examine the variability and impact of these operations on the observed gap E_2 . The table below summarizes the results for Group 1.2 and Group 2.

TABLE 5.5 : Summary results of Group 1.2 and Group 2 for E_2

Metric	Group 1.2	Group 2
% Instances solved (Permutation)	88.75%	85%
% Instances solved (Non-Permutation)	74.73%	58.75%
Avg. gap E_2	1.91	3.60
Std. dev. gap E_2	2.28	3.20
Max gap E_2	11.06	13.79
Min gap E_2	0.00	0.00
Avg. time (permutation)	4.22 s	6.78s
Avg. time (non-permutation)	263.02 s	241.20 s
RDP	68.35	87.23

As shown in Table 5.5, non-permutation schedules require significantly longer execution times than permutation schedules for both groups. However, the large and variable gap between these solutions suggests that, for certain instances, non-permutation schedules may still be preferable.

As in Section 5.6.1, we start our analysis with Group 1.2. In this group, 74.73% of instances were solved to optimality in both the permutation and non-permutation models and will be considered in the subsequent study. The statistical study aims to examine the relationships between the indicators IND_1 , IND_2 , IND_3 , and IND_4 , along with the variability in tasks, machines, and processing time intervals. The goal is to understand how these factors influence the gap E_2 between flow shops with and without permutation constraints, focusing on the minimization of the total completion time.

Figure 5.19 illustrates an interaction between the indicators and the gap E_2 . A large gap in E_2 appears to be associated with high values of IND_4 , suggesting a potential correlation between these two elements.

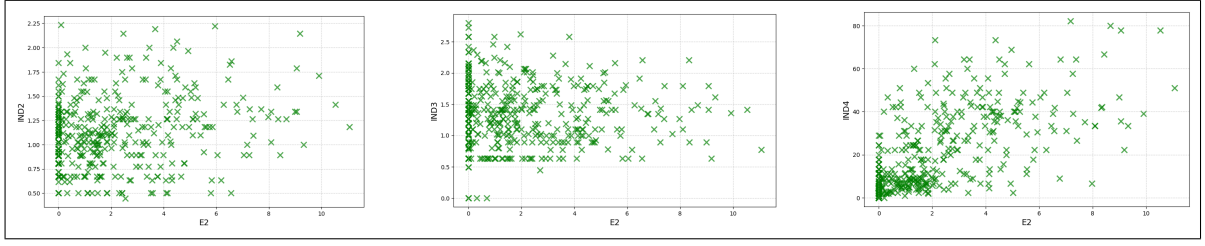


FIGURE 5.19 : Relationship between E_1 and IND_2 , IND_3 , and IND_4

The following table provides the results of the Spearman correlation test between E_2 and the different indicators :

TABLE 5.6 : Spearman correlation test results for E_2

Indicator	Spearman's ρ	P-Value
IND_1	-0.45	3.28×10^{-25}
IND_2	0.214	2.73×10^{-6}
IND_3	-0.011	0.809
IND_4	0.74	6.13×10^{-80}

The results presented in Table 5.6 show that IND_4 has the strongest positive correlation with E_2 , followed by IND_2 , which also shows a significant correlation. Meanwhile, IND_1 exhibits a statistically significant negative correlation ($\rho = -0.45$) with E_2 , indicating that as IND_1 increases, the gap in total completion time between permutation and non-permutation schedules decreases. However, IND_3 shows no meaningful correlation with E_2 , as indicated by the small Spearman's ρ value and the high p-value. Finally, the correlation of IND_4 with the gap between permutation and non-permutation schedules is stronger when considering the total completion time objective, compared to that obtained with the makespan objective.

The following figure presents box plots to illustrate the relationship between E_2 and the different categorical values of the indicators IND_1 , IND_2 , IND_3 , and IND_4 . These visualizations will help us assess the influence of each indicator on the total completion.

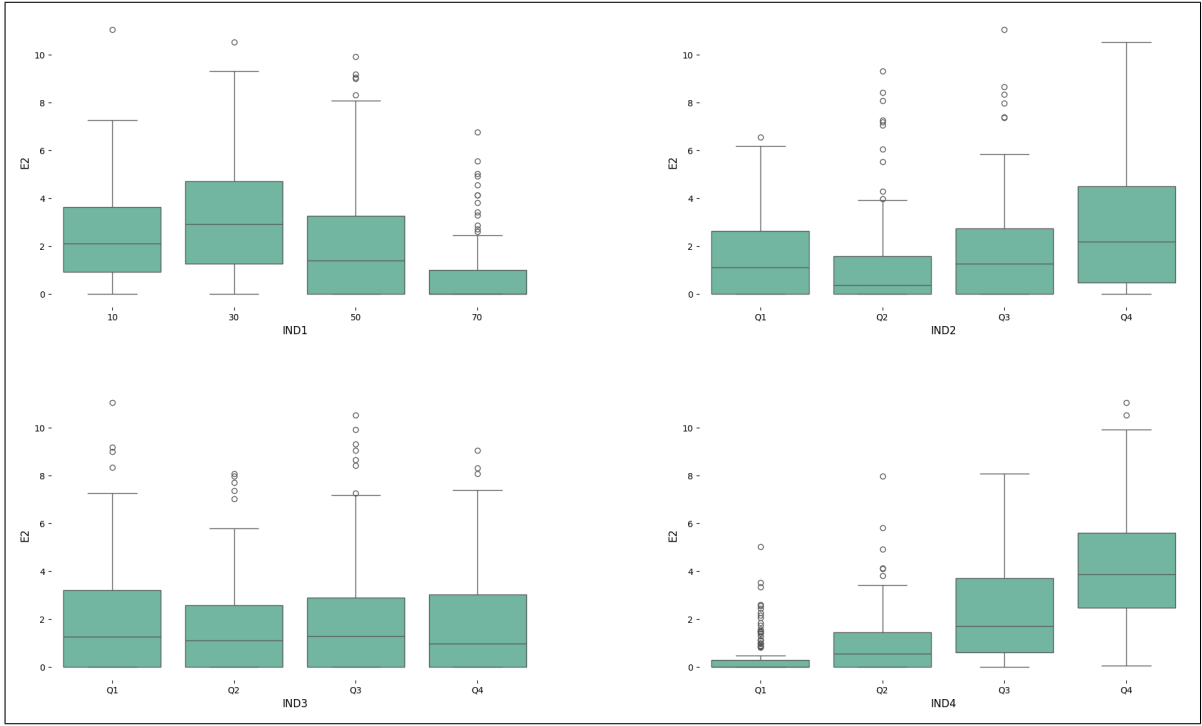


FIGURE 5.24 : Distribution of E_2 across different categories of IND_1 , IND_2 , IND_3 , and IND_4

The box plots reveal patterns in E_2 across the various categories of each indicator, showing results closely aligned with those observed for E_1 . For IND_1 , we see that instances with very high (70%) percentages of missing operations tend to have smaller gaps E_2 , while those with intermediate values (50%) exhibit greater variability. The IND_4 remains the most impactful, with higher values consistently associated with the largest gaps E_2 . This reinforces the notion that instances with a greater number of interchangeable task pairs are more likely to generate significant differences between the two scheduling approaches.

Next, we performed ANOVA tests to analyze the significance of the relationships between the discrepancies E_2 and the four indicators. The results are presented in Table 5.7.

TABLE 5.7 : ANOVA Test Results for E_2 Across Indicators

Indicator	F-Statistic	P-Value
IND_1	33.20	1.77×10^{-19}
IND_2	10.18	1.66×10^{-6}
IND_3	0.66	0.575
IND_4	124.6	2.18×10^{-59}

The results in Table 5.7 indicate that IND_4 , IND_1 , and IND_2 have significant effects on the variation of E_2 , with very small p-values providing strong evidence of the relationship between these indicators and the total completion time. Among them, IND_4 has the most significant effect on E_2 . However, IND_3 does not show a statistically significant relationship with E_2 , as indicated by its high p-value.

Based on the previous results, the strong correlation between E_2 and IND_4 as well as the ANOVA tests, which showed significant differences across IND_4 categories, indicate that IND_4 remains the most impactful factor. Higher values of IND_4 are consistently associated with the largest gaps. Therefore, it is recommended to consider non-permutation scheduling in instances with high IND_4 values.

Finally, we studied the impact of the number of machines, the number of jobs, and the variability in processing time intervals on E_2 . As shown in Figure 5.28, the number of machines has a clear effect on the total completion time gap, with larger gaps observed for instances with 10 machines compared to 5 machines. The processing time intervals do not exhibit a statistically significant effect on E_2 , whereas the number of jobs demonstrates a minor yet significant influence, as confirmed by the ANOVA results.

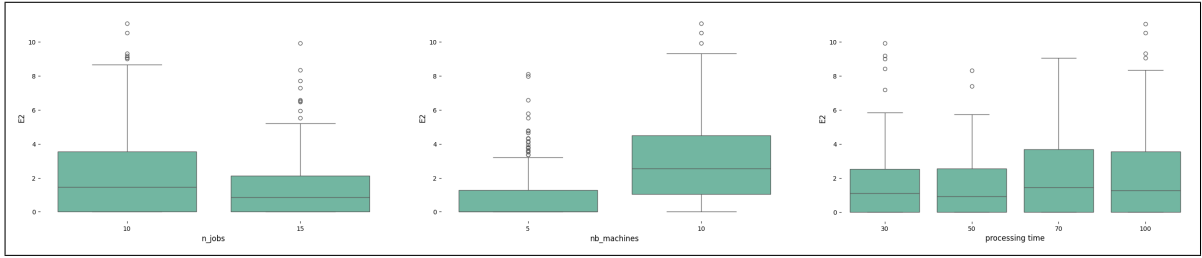


FIGURE 5.28 : Impact of the number of jobs, number of machines, and processing time intervals on E_2

The ANOVA test results for these factors are shown in Table 5.8. The test for the number of machines yields a highly significant F -statistic of 141.87 and a p-value of 8.77×10^{-29} , indicating a strong effect on E_2 .

TABLE 5.8 : ANOVA test results for E_2 based on the number of jobs, machines, and processing time intervals

Factor	F-Statistic	P-Value
Number of jobs	8.91	0.0029
Number of machines	141.87	8.77×10^{-29}
Processing time intervals	2.71	0.044

We now proceed with the analysis of Group 2. 58.75% of the instances were solved to optimality within 6000 seconds in both the permutation and non-permutation models and will be considered in the subsequent analysis. The results are summarized as follows.

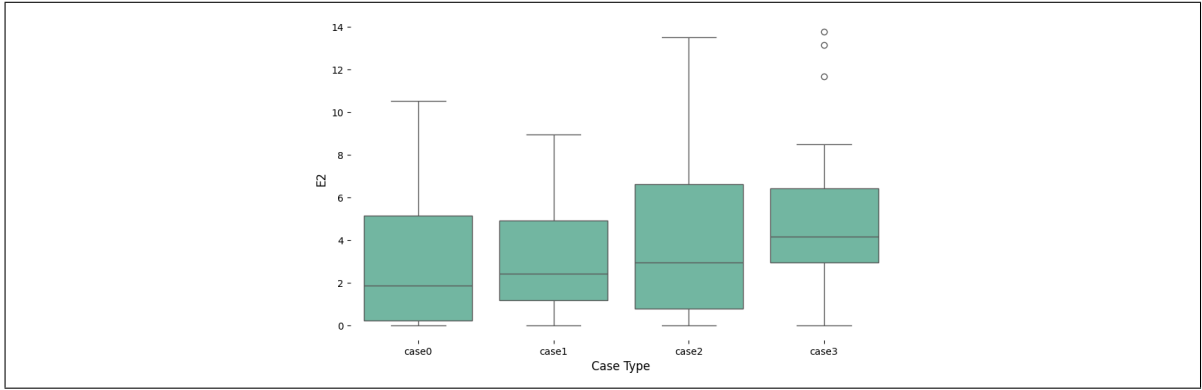


FIGURE 5.29 : Impact of case types on the gap E_2

From the boxplot in Figure 5.29, we observe that Case 3, where no missing operations are allowed on both the first and last machines, shows the largest median value and variability for E_2 . This suggests that the position of missing operations significantly affects the gap E_2 in these instances.

To confirm the statistical significance of these observations, we performed an ANOVA test. The results, with $F = 4.02$ and $p = 0.0084$, indicate that the configuration of missing operations has a statistically significant effect on E_2 , confirming that the case types have an impact on E_2 .

5.7 CONCLUSION

In this study, we have shown that in certain configurations, non-permutation schedules can yield significantly better results than permutation schedules for both makespan and total completion time criteria, with differences reaching up to 20% for makespan and 13% for total completion time. Nonetheless, in instances where this gap is minimal or negligible, permutation schedules remain preferable due to the considerably lower computational cost compared to non-permutation schedules.

To understand why the gap between permutation and non-permutation schedules is sometimes large and other times negligible, we identified four indicators. The first three indicators assess the percentage and variability of missing operations, while the fourth indicator counts the number of job pairs that can potentially be interchanged in their processing sequence across machines. The statistical tests confirmed the significance of these indicators, except for IND_3 , which was not statistically significant. In particular, IND_4 proved to be a key factor in understanding the gap between permutation and non-permutation schedules. When IND_4 is classified as "Very High" or "High," the gap is typically significant. Conversely, for classifications of "Very Low" or "Low," the gap tends to be minimal or negligible. Furthermore, our analysis on the additional structural elements of the instances, such as processing time intervals, number of machines and tasks, and missing operation configurations, suggests that for cases with a higher number of machines, non-permutation schedules are generally more beneficial, as the gap between scheduling approaches increases with machine count.

These findings can be valuable for future research, particularly in the development of decision-support methods based on the identified indicators, especially IND_4 . Such a method could assist in determining when it is preferable to use non-permutation schedules despite the higher computational cost they entail, thereby providing practical guidance for scheduling decisions in flow shop environments.

ACKNOWLEDGMENTS

This research is partially financed by the Natural Sciences and Engineering Research Council of Canada (NSERC).

DATA AVAILABILITY STATEMENT

The data that support the findings of this study are available from the corresponding author, R. Ouchene, upon request.

BIBLIOGRAPHY

- BENAVIDES, A. J., & RITT, M. (2018c). Fast heuristics for minimizing the makespan in non-permutation flow shops. *Computers & OR*, 100, 230-243.
- BRUM, A., RUIZ, R., & RITT, M. (2022c). Automatic generation of iterated greedy algorithms for the non-permutation flow shop scheduling problem with total completion time minimization. *Computers & Industrial Engineering*, 163, 107843.
- CHANG, P.-C., HUANG, W.-H., WU, J.-L., & CHENG, T. (2013). A block mining and re-combination enhanced genetic algorithm for the permutation flowshop scheduling problem. *International Journal of Production Economics*, 141(1), 45-55.
- CHENG, J., KISE, H., & MATSUMOTO, H. (1997). A branch-and-bound algorithm with fuzzy inference for a permutation flowshop scheduling problem. *European Journal of Operational Research*, 96(3), 578-590.
- EKŞİOĞLU, B., EKŞİOĞLU, S. D., & JAIN, P. (2008). A tabu search algorithm for the flowshop scheduling problem with changing neighborhoods. *Computers & Industrial Engineering*, 54(1), 1-11.
- FRENCH, S. (1982b). *Sequencing and Scheduling : An Introduction to the Mathematics of the Job-Shop*. Ellis Horwood Ltd.
- GAREY, M. R., JOHNSON, D. S., & SETHI, R. (1976b). The Complexity of Flowshop and Jobshop Scheduling. *Mathematics of Operations Research*, 1(2), 117-129.
- GLASS, C. A., GUPTA, J. N., & POTTS, C. N. (1999a). Two-machine no-wait flow shop scheduling with missing operations. *Mathematics of Operations Research*, 24(4), 911-924.
- HENNEBERG, M., & NEUFELD, J. (2016c). A constructive algorithm and a simulated annealing approach for solving flowshop problems with missing operations. *International Journal of Production Research*, 54(12), 3534-3550.
- IGNALL, E., & SCHRAGE, L. (1965a). Application of the Branch and Bound Technique to Some Flow-Shop Scheduling Problems. *Operations Research*, 13(3), 400-412.
- JOHNSON, S. M. (1954c). Optimal two-and three-stage production schedules with setup times included. *Naval Research Logistics Quarterly*, 1(1), 61-68.
- LADHARI, T., & HAOUARI, M. (2005b). A computational study of the permutation flow shop problem based on a tight lower bound. *Computers & OR*, 32(7), 1831-1847.
- LEISTEN, R., & KOLBE, M. (1998c). A note on scheduling jobs with missing operations in permutation flow shops. *International Journal of Production Research*, 36(9), 2627-2630.
- LIAO, C.-J., LIAO, L., & TSENG, C. (2006a). A performance evaluation of permutation vs. non-permutation schedules in a flowshop. *International Journal of Production Research*, 44, 4297-4309.

- NAWAZ, M., ENSCORE JR, E. E., & HAM, I. (1983b). A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. *Omega*, 11(1), 91-95.
- OUCHENE, R., REBAINE, D., & BAPTISTE, P. (2023c). Ambiguity of the definition of permutation flow shops in the presence of missing operations. *Computers & Industrial Engineering*, 182, 109387.
- POTTS, C. N., SHMOYS, D. B., & WILLIAMSON, D. P. (1991d). Permutation vs. non-permutation flow shop schedules. *Operations Research Letters*, 10(5), 281-284.
- PUGAZHENDHI, S., THIAGARAJAN, S., RAJENDRAN, C., & ANANTHARAMAN, N. (2003c). Performance enhancement by using non-permutation schedules in flowline-based manufacturing systems. *Computers & Industrial Engineering*, 44(1), 133-157.
- RAMEZANIAN, R., SAIDI-MEHRABAD, M., & RAHMANI, D. (2011b). Flow shop scheduling problem with missing operations : Genetic algorithm and tabu search. *International Journal of Applied Operational Research*, 1, 21-30.
- RITT, M., & ROSSIT, D. A. (2024c). Effective heuristics for permutation and non-permutation flow shop scheduling with missing operations. *Computers & Operations Research*, 170, 106742.
- ROSSIT, D. A., TOHMÉ, F., & FRUTOS, M. (2018c). The non-permutation flow-shop scheduling problem : a literature review. *Omega*, 77, 143-153.
- SADJADI, S., ARYANEZHAAD, M., & ZIAEE, M. (2008c). The general flowshop scheduling problem : mathematical models. *Journal of Applied Sciences*, 8(17), 3032-3037.
- SAHNI, S., & CHO, Y. (1979a). Complexity of scheduling shops with no wait in process. *Mathematics of Operations Research*, 4(4), 448-457.
- VASILJEVIC, D., & DANILOVIC, M. (2015). Handling ties in heuristics for the permutation flow shop scheduling problem. *Journal of Manufacturing Systems*, 35, 1-9.

5.8 APPENDIX : THE NO WAIT CASE

For the sake of clarity, we first recall the definition of a no-wait flow shop.

Définition 5.8.1. *A no-wait flow shop problem is a flow shop where each job is executed with no waiting time between two successive operations of the same job, i.e. each operation has to be immediately followed by its next operation*

Théorème 5.8.1. *Let us assume, in a valid no-wait schedule, any two jobs i_1 and i_2 , and two machines $(h, k) \in E_{i_1, i_2}$, where job i_1 precedes job i_2 on machine h , and job i_2 precedes job i_1*

on machine k . Then, we have :

$$p_{i_1,h} + \sum_{j=h}^k p_{i_2,j} \leq \sum_{j=h}^{k-1} p_{i_1,j}.$$

Proof. Let $T_{i_1,j}$ and $C_{i_1,j}$, as well as $T_{i_2,j}$ and $C_{i_2,j}$, be the start and completion times of jobs i_1 and i_2 on machine j , respectively, in a valid no-wait schedule. With the no-wait constraint, we have

$$C_{i_2,k} = T_{i_2,h} + \sum_{j=h}^k p_{i_2,j},$$

and

$$T_{i_1,k} = T_{i_1,h} + \sum_{j=h}^{k-1} p_{i_1,j}.$$

On the one hand, since job i_1 precedes job i_2 on machine h , we get

$$T_{i_1,h} + p_{i_1,h} \leq T_{i_2,h}.$$

On the other hand, assuming job i_2 precedes job i_1 on machine k , we have

$$C_{i_2,k} = T_{i_2,h} + \sum_{j=h}^k p_{i_2,j} \leq T_{i_1,k}.$$

Substituting these into the respective equations, we derive

$$T_{i_1,h} + p_{i_1,h} + \sum_{j=h}^k p_{i_2,j} \leq T_{i_1,h} + \sum_{j=h}^{k-1} p_{i_1,j}.$$

Therefore, as illustrated in Figure 5.30, we obtain

$$p_{i_1,h} + \sum_{j=h}^k p_{i_2,j} \leq \sum_{j=h}^{k-1} p_{i_1,j}.$$

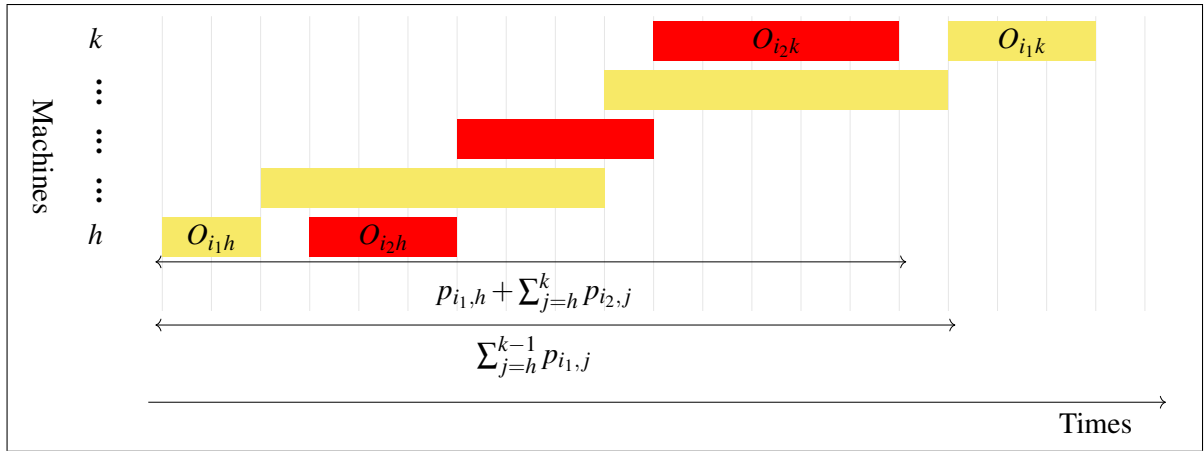


FIGURE 5.30 : Representation of the necessary condition on job peers

CHAPITRE VI
THE IMPACT OF SERVICE CONSTRAINTS ON THE
MAKESPAN IN A JOB SHOP PROBLEM

Ce chapitre est basé sur un manuscrit soumis à la revue *Annals of Operations Research* le 24 mars 2025.

Ouchene, R., Baptiste, P. & Rebaine, D., *The impact of service constraints on the makespan in a job shop problem. Annals of Operations Research*, soumis le **24 Mars 2025**.

ABSTRACT

In job shop scheduling problems, the majority of research has been centered around the makespan criterion, which relates to the efficiency of resource allocation. However, in the service sector, customer experience must also be taken into account. This experience can be compromised when some customers overtake others, which constitutes a violation of the FIFO (First In, First Out) rule, or when there are unnecessary delays, even if resources are available for processing (a violation of the No-delay constraint). This paper uses the Mixed Integer Programming (MIP) approach to study the impact of considering FIFO together with No-delay constraints in job shop models within the service sector. Three new models are developed *i.e.*, the first with the FIFO constraint, the second with the No- delay constraint, and the third with both constraints considered together. The experimental results we conducted show that incorporating these constraints individually has a limited impact on the makespan. while their combination leads to significant differences. In addition, we propose an approach to minimize the number of constraint fifo violations and the number of maximum delay while maintaining an optimal makespan, offering a compromise between improving customer experience and efficient resource management.

keywords : Job Shop ; Makespan ; FIFO ; No-delay.

6.1 INTRODUCTION

The traditional job shop scheduling problem (JSP) involves the processing of $J = \{1, \dots, n\}$, a set of n of jobs, across $M = \{1, \dots, m\}$, a set of m machines. Each job comprises a vector of operations $(O_{i,j_1}, \dots, O_{i,j_k})$ to be processed in a specific order on different machines. Each operation O_{ij} is processed on machine j for a time length p_{ij} . The sequence $j_1, j_2, \dots, j_k$, the order of machines in the vector of operations, denotes the processing route each job must follow through the machines (MANNE, [1960c](#)).

Schedules in JSP are classified into three categories : semi-active, active, and non-delay schedules. In a semi-active schedule, no operation can start earlier without changing the processing order of a machine. In an active schedule, no operation can start earlier without delaying another operation. Finally, in a non-delay schedule, no machine is left idle while an

operation is waiting for processing. Active schedules are known to be the smallest dominant set of schedules (FRENCH, [1982c](#)).

The JSP was originally designed for manufacturing and production systems (CAI et al., [2023](#); A. K. GUPTA et SIVAKUMAR, [2006](#); M. GUPTA et al., [2002](#)), primarily to optimize machine utilization. However, in recent years, the job shop model has found extensive applications in the service sector, particularly in healthcare, where it is used to organize patient care pathways. In this context, each patient represents a 'job' moving through a series of stations or medical services, which function as machines in the traditional job shop model (LIANG et al., [2015c](#); VOGL et al., [2019c](#))

In the environment of services, the minimization of the waiting time is crucial AGNIHOTRI et al., [2024b](#), and this becomes even more important when resources are available but not used. The No-delay constraint aims to eliminate unnecessary waiting time by ensuring available resources are promptly used, thereby reducing inefficiencies and enhancing the client experience. Furthermore, the First-In-First-Out (FIFO) discipline is the default rule used in many service operations (MARYNISSEN et DEMEULEMEESTER, [2019c](#)). It ensures clients are treated in the order of their arrival, which is often perceived as fair and helps reduce the anxiety and frustration associated with waiting.

Incorporating these constraints (FIFO, No-Delay) into the job shop model in a service environment can significantly enhance patient satisfaction and overall well-being. However, in a job shop aiming to minimize the makespan, certain operations may be granted higher priority than others. The two ways to prioritize an operation are either to push it to the front of a queue (violation of FIFO), or, if FIFO is imposed, to block non-priority operations in upstream queues (Violation of No-Delay). Consequently, enforcing one constraint can lead to

potential violations of the other, and imposing both simultaneously may substantially increase the makespan. This is what is studied in this work.

To the best of our knowledge, there are currently no mathematical formulations in the scheduling job shop literature that capture the constraints of No-delay and FIFO in the scheduling literature. The contribution of this article is to propose three new models that explicitly incorporate these constraints : FIFO, No-delay, and FIFO with No-delay together. We also provide an in-depth analysis of the impact of these constraints on the makespan value of a schedule.

This work is oriented towards applications that occur in service environments, where jobs represent clients. Similar to how patients arrive for appointments in a service setting, jobs have entry dates. These entry dates are not predetermined and must be established by the different proposed models. The FIFO and the No-delay constraints, taken together, take effect once the jobs (clients) have entered the system *i.e.*, after their entry dates.

This paper is organized as follows. Section 2 presents a review of the literature in the field of job shops and multi-appointment scheduling problems, providing insights into the development and application of scheduling models in various service sectors, with a focus on healthcare. In Section 3, we develop the mathematical modeling of key constraints, specifically FIFO and No-delay. Section 4 analyzes the experimental studies conducted, highlighting the findings. Finally, Section 5 concludes the paper with a summary of the findings and proposals for future research.

6.2 LITERATURE REVIEW

The literature is rich with various solving methods proposed to tackle this highly combinatorial problem. Among the exact approaches used to solve the job shop problem, mixed

integer linear programming (MILP) is frequently applied. Three Mixed Integer Programming (MIP) formulations are commonly used for solving the JSP problems *viz.* the time-indexed formulation BOWMAN, 1959b, the rank-based formulation WAGNER, 1959b, and the disjunctive formulation MANNE, 1960c. A theoretical comparison between these formulations asserts that the disjunctive formulation is the most effective, while the time-indexed formulation performs the worst C.-H. PAN, 1997b. This conclusion was experimentally confirmed in KU et BECK, 2016b, which compared the performance of various MIP models for the classical job shop problem, demonstrating the disjunctive model is the most effective in minimizing the makespan. Let us mention other exact methods have also been applied to the job shop problem, such as Lagrangian relaxation and branch-and-cut methods. For example, Lagrangian relaxation has been used to minimize total delay YAN et al., 2018b, while a branch-and-cut algorithm based on the disjunctive formulation has also been used to address the problem APPELGATE et COOK, 1991d.

However, exact methods can struggle with the high computational complexity of large-scale instances. As a result, researchers tend to favor the development of approximate methods to achieve satisfactory solutions. Among heuristic methods, there are mainly those based on priority dispatching rules, such as the shortest processing time rule, the first in, first out rule, and the Most Work Remaining CHIANG et FU, 2007b; HAUPT, 1989c; NARASSIMA et al., 2022b. With regard to the meta-heuristic approach, we may cite Tabu Search DELL'AMICO et TRUBIAN, 1993b, Simulated Annealing KRISHNA et al., 1995b, and Evolutionary Algorithms SALIDO et al., 2016b, to name a few. However, in recent years, hybrid algorithms ALKHATEEB et al., 2022b; FONTES et al., 2023b have been introduced and have gained popularity.

Recently, there has been a growing interest in addressing multi-appointment scheduling problems in hospitals; for a comprehensive review, see e.g. MARYNISSEN et DEMEULEMEESTER, 2019c. The principal techniques used for appointment scheduling prominently include heu-

ristic and meta-heuristic methods (BURDETT et KOZAN, [2018b](#), KAZEMIAN et al., [2017b](#), REZAEIAHARI et KHASAWNEH, [2017b](#)), stochastic optimization (LEEFTINK et al., [2019b](#), ZHOU et YUE, [2019b](#)), column generation and Lagrangian relaxation techniques (XIAO et al., [2022b](#), DIAMANT et al., [2018b](#)), and mathematical programming (CONFORTI et al., [2011b](#), KORTBEEK et al., [2017](#), LODI et TUBERTINI, [2016](#)).

Multi-appointment scheduling problems have been interpreted in some cases as shop scheduling problems with resource constraints. For instance, the flow shop model is utilized in LIANG et al., [2015c](#), in which the authors introduce a two-stage mathematical programming model aimed at improving appointment scheduling in oncology clinics, focusing on the balanced utilization of resources and the minimization of patient waiting times. In VOGL et al., [2019c](#), the complex job shop scheduling problem was addressed with additional constraints for scheduling radiotherapy appointments, seeking to minimize the operational time of the particle beam the bottleneck resource while minimizing penalties from the time window constraint violations. An integer mathematical model and meta-heuristic strategies that combine a genetic algorithm with iterative local search are proposed as solving approaches.

For multi-appointment scheduling problems, the objectives studied in the literature vary, ranging from a single to multiple objectives. For works focusing on single-objective optimization, we refer to SAADANI et al., [2014b](#), who discuss the scheduling policy of a set of patient operations on some resources of the Medical Technical Platform and operating rooms. This work develops an MILP model that aims to minimize the duration of stay of each patient. CONFORTI et al., [2011b](#) presents an optimal decision-making model for supporting hospital healthcare managers, with an objective function that focuses on maximizing the number of admitted patients during the planning week. For multi-objectives, ALA et al., [2022b](#) discusses models that enhance fairness by minimizing average admission times, especially for patients with urgent clinical needs, whereas the second objective reduces patient dissatisfaction from

prolonged hospital stays. The authors present strategies such as Fuzzy Ant Lion Optimization (FALO) and Non-dominated Sorting Genetic Algorithm II (NSGA-II). In CASTRO et PETROVIC, 2012, the primary focus is on minimizing the weighted number of patients who exceed waiting time targets. The second objective involves minimizing the maximum lateness, and the third objective is the minimization of the sum of weighted lateness for a radiotherapy pre-treatment scheduling problem.

6.3 MIXED INTEGER PROGRAMMING APPROACH

In this section, we present mathematical formulations for different job shop scheduling models. Figure 6.1 illustrates the connections among the eight job shop scheduling models examined in this study. These models incorporate constraints such as No-Delay and FIFO (First In, First Out), while aiming to minimize three key objectives : makespan, delay, and FIFO violations. The diagram demonstrates how each model interacts with the others in light of these constraints and objectives.

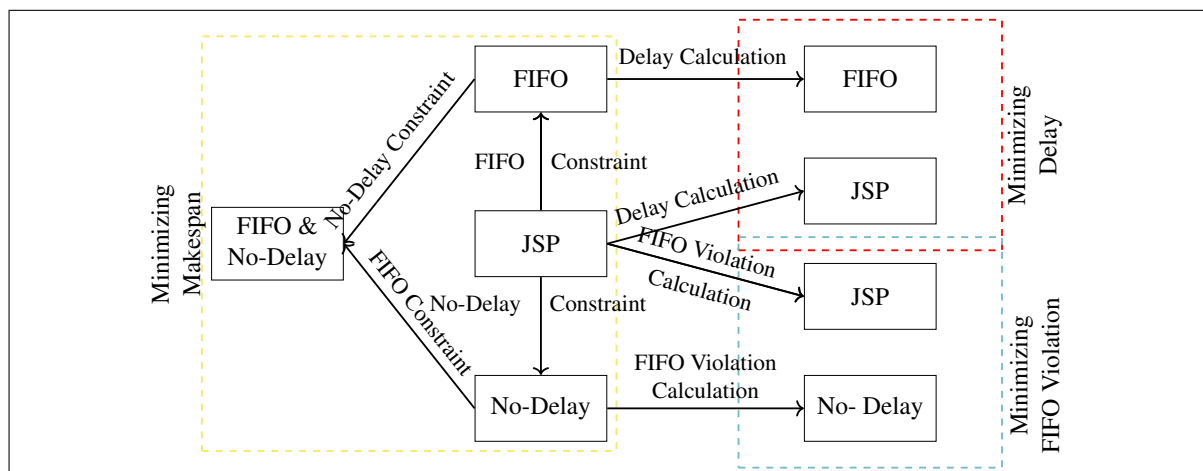


FIGURE 6.1 : A visual representation of job shop scheduling models with FIFO and No-Delay constraints, aimed at minimizing makespan, delay, and FIFO violations

6.3.1 CLASSICAL DISJUNCTIVE JOB SHOP MODEL

The traditional job shop scheduling Problem (JSP) involves a number of jobs that need to be processed through a set of available machines. Each job consists of a number of operations, each having a specific processing time. The usual assumptions apply as follows :

- Adjacent operations within the same job must satisfy the priority constraint.
- Any job can only be processed by one machine at time.
- Any machine can only process one operation at a time.
- Preemption is not allowed.
- Each operation has a fixed and known processing time.
- The transportation time during job processing is ignored or included in the processing time.

The traditional job shop scheduling problem can be defined using a disjunctive graph representation. Here, each job consists of a sequence of operations processed on specified machines in a particular order. The goal is to minimize the makespan. The mathematical formulation is as follows :

Parameters

- i : Index for jobs, ranging from 1 to n where n is the number of jobs.
- j : Index for machines, ranging from 1 to m where m is the number of machines.
- k : k -th operation of a job.
- p_{ij} : Processing time of job i on machine j .
- m_{ik} : Machine on which the k -th operation of job i is processed.
- L : Large value.

Decision variables

- S_{ij} : Start time of job i on machine j . This represents the time at which job i begins processing on machine j .
- C_i : Completion time of job i . This is the time at which job i finishes all its operations.
- $C_{\max}$: Completion time of the last job in the schedule (also known as the makespan).
- $B_{i_1, i_2, j}$: Binary variable of value 1 if job i_1 precedes job i_2 on machine j , to ensure proper sequencing of jobs on each machine.

The following is the mathematical disjunctive model of job shop as presented in HAUPT, 1989c :

$$M_1 \left\{ \begin{array}{ll} \min & C_{\max} \quad (1) \\ C_i \geq & S_{ij} + p_{ij}, \quad \forall i \in J, \forall j \in M, \quad (2) \\ C_{\max} \geq & C_i, \quad \forall i \in J, \quad (3) \\ S_{ij} \geq & 0, \quad \forall i \in J, \forall j \in M, \quad (4) \\ S_{i, m_{ik}} + p_{i, m_{ik}} \leq & S_{i, m_{i(k+1)}}, \quad \forall i \in N, \forall k < m, \quad (5) \\ S_{i_1, j} + p_{i_1, j} \leq & S_{i_2, j} + L(1 - B_{i_1 i_2 j}), \quad \forall i_1, i_2 \in N, i_1 \neq i_2, \forall j \in M, \quad (6) \\ B_{i_1 i_2 j} + B_{i_2 i_1 j} = & 1, \quad \forall i_1, i_2 \in N, i_1 \neq i_2, \forall j \in M. \quad (7) \end{array} \right.$$

Where,

- (1) $C_{\max}$, the makespan, is the objective to minimize.
- (2) The completion time of each job i is not earlier than the end time of its operations.
- (3) $C_{\max}$ is at least as long as the completion time of any job.
- (4) The start times of the jobs are non-negative.
- (5) The start time of operation $(k+1)$ of job i on machine $m_{i(k+1)}$ has to be processed after the completion of operation k of job i on machine m_{ik} .

- (6) If job i_1 precedes job i_2 on machine j , then $C_{i_1} \leq S_{i_2}$.
- (7) On any machine, either job i_1 precedes job i_2 or vice versa, ensuring a clear operation order.

The number of variables and constraints in the above mixed-integer programming model are $(1+n)nm$ and $n(2n+1)m$, respectively. In the following analysis, the optimal makespan for the classical job shop is denoted by $C_{\max}^{\text{JSP}}$.

6.3.2 JOB SHOP WITH THE FIFO CONSTRAINT

The FIFO model enforces the First In, First Out discipline, where jobs are processed upon the order of their arrival to the machines. This constraint is essential in service environments to ensure fairness and reduce waiting anxiety. In addition to its practical importance, the application of FIFO in the context of a job shop can also be viewed from a theoretical perspective. Indeed, the concept of FIFO is fundamental in the ordering of operations within a permutation flow shop system (PFSP), where, according to Pinedo : *all queues are assumed to operate under the FIFO rule, i.e., a job cannot overtake another one while in the queue* PINEDO, 2002d. This suggests the incorporation of the FIFO constraint could also serve as a theoretical definition of permutation in the job shop problem, inspired by the definition of Pinedo introduced in the flow shop model.

To illustrate this idea of a job shop with the FIFO constraint, we consider the following two scheduling solutions of an instance with 3 jobs and 3 machines. The processing times of job 1, job 2, and job 3 are $(3, 1, 2)$, $(2, 2, 1)$, and $(3, 1, 7)$, respectively.

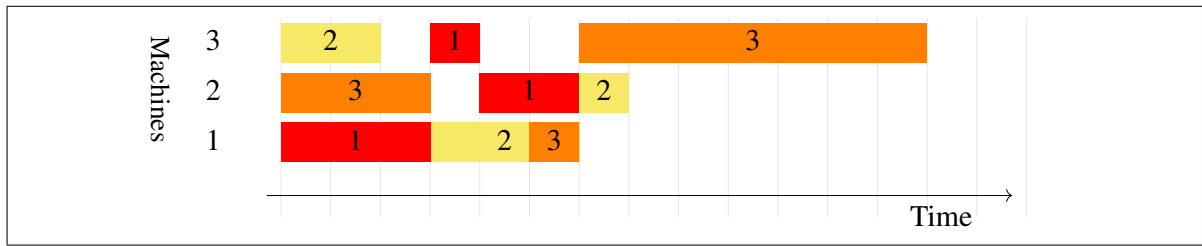


FIGURE 6.2 : Scheduling with respect to FIFO in a job shop problem

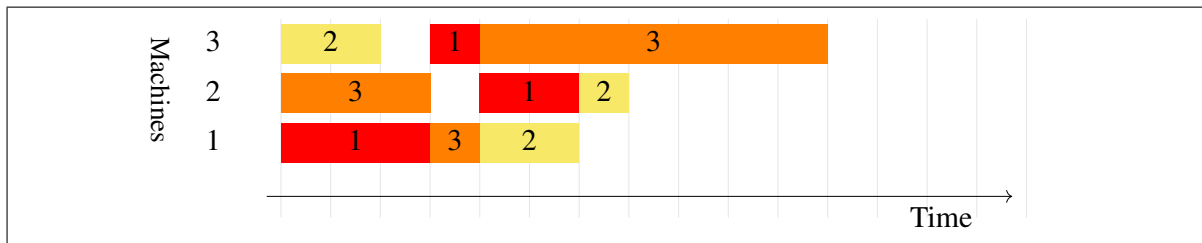


FIGURE 6.3 : Scheduling without the FIFO constraint in a job shop problem

Figure D.4 represents a job shop system respecting the FIFO constraint, where the first job in the machine's queue is always executed first. This is in accordance with the FIFO rule. On the other hand, Figure D.5 shows a scenario where the scheduling solution does not respect the FIFO rule. Indeed, job 2 finishes before job 3 and thus is the first in the waiting list for machine 1. However, despite being the first in the queue, job 2 is executed after job 3, violating the FIFO rule. This solution is in fact optimal.

So far, we defined what a job shop with FIFO constraint implies. From this definition, the following property emerges.

Théorème 6.3.1. *The scheduling solutions, respecting the FIFO constraints, are dominant in a two-machine job shop subject to makespan.*

Proof : The two-machine job shop with the FIFO constraint can be solved to optimality by the JACKSON, 1956 $O(n \log n)$ -time algorithm. Let us recall the optimal solution is produced by

reducing it into two flow shop problems each with two machines. Jobs are categorized into four groups : I_1 for those processed solely on the first machine, I_2 for those on the second machine, $I_{1,2}$ for those that first pass through machine 1 and then machine 2, and $I_{2,1}$ for the reverse route. We first schedule to optimality $I_{1,2}$ and $I_{2,1}$, corresponding to two flow shops, using Johnson rule JOHNSON, 1954d. The final schedule for the corresponding job shop is as follows : start on the first machine with the optimal sequence of $I_{1,2}$, followed by I_1 , and then by the optimal sequence of $I_{2,1}$, whereas the sequence is reversed on the second machine. This approach ensures the first job to arrive at the corresponding machine is the first to be processed. The FIFO rule is thus respected. $\square$

To incorporate the FIFO constraint into a job shop problem, we have adapted the classical disjunctive model. This adaptation requires the introduction of the following parameter :

mpr_{ij} : Machine assigned for the operation immediately preceding the operation on machine j for job i . For the first operation of each job, $\text{mpr}_{i,m_{i1}} = -1$ to indicate no preceding operation.

In addition, we introduced a set of constraints to enforce the FIFO rule, expressed as follows :

$$S_{i_1, \text{mpr}_{i_1 j}} + p_{i_1, \text{mpr}_{i_1 j}} + B_{i_1 i_2 j} L \geq S_{i_2, \text{mpr}_{i_2 j}} + p_{i_2, \text{mpr}_{i_2 j}}, \quad \forall i_1 \neq i_2 \in N, \forall j \in M \quad (8)$$

where $\text{mpr}_{i_1 j}, \text{mpr}_{i_2 j} \neq -1$. This constraint (8) ensures for each pair of jobs, i_1 and i_2 , and for each machine, j , where jobs i_1 and i_2 require a prior operation on another machine, before their processing on machine j , if job i_2 is scheduled before job i_1 on machine j (indicated by $B_{i_1, i_2, j} = 0$), then job i_2 must complete its processing on the preceding machine $\text{mpr}_{i_2 j}$ before

the completion of job i_1 on its preceding machine mpr_{i_1j} .

The mathematical formulation for the FIFO model is therefore as follows :

$$M_2 \left\{ \begin{array}{l} \min \quad C_{\max} \\ \text{subject to} \\ (2), \quad (3), \quad (4), \quad (5), \quad (6), \text{ and } \quad (7), \\ S_{i_1, \text{mpr}_{i_1j}} + p_{i_1, \text{mpr}_{i_1j}} + B_{i_1i_2j}L \geq S_{i_2, \text{mpr}_{i_2j}} + p_{i_2, \text{mpr}_{i_2j}}, \\ \forall i_1 \neq i_2 \in J, \forall j \in M, \\ \text{mpr}_{i_1j}, \text{mpr}_{i_2j} \neq -1. \end{array} \right. \quad (1) \quad (8)$$

The number of variables and constraints of the above mixed integer programming model are $(1+n)nm$ and $3mn^2$, respectively.

We denote the resulting optimal makespan Job shop problem under FIFO constraints by $C_{\max}^{\text{FIFO}}$.

6.3.3 JOB SHOP WITH THE NO-DELAY CONSTRAINT

We ensure in this model there are No-Delays between the time a job is ready to be processed and the actual start of processing on any machine. This constraint helps in reducing perceived inefficiencies and enhances resource utilization.

The mathematical formulation of the no-delay job shop problem is as follows. We adhere to the disjunctive job shop model. However, a group of constraints is added to enforce the no-delay condition. To do so, we need the following two conditions :

- Condition 1 denotes the case where the start time of the operation $O_{j,k}, j = m_{i,k}$, must be either equal to the finish time of the preceding operation in the same job $O_{i,k-1}$, or it must be equal to the finish time of the operation from another job that immediately precedes it on the same machine. Utilizing the notation from the disjunctive model, this can be formulated as follows :

$$S_{i,m_{i,k}} = \max(S_{i,m_{i,k-1}} + p_{j,m_{i,k-1}}, S_{h,j} + p_{h,j}),$$

where h denotes the job that immediately precedes job i on machine j , for $k = 2, \dots, n$. This indicates operation k of job i on machine j either follows operation $k - 1$ of the same job or follows the operation of job h on machine j that has just been completed.

- Condition 2, on the other hand, addresses the case where the start time of the operation $O_{i,k}, j = m_{i,k}$, is equal to the finish time of the preceding operation in the same job $O_{i,k-1}$. The figure below illustrates this scenario, showing the inactivity time w_{ij} on machine j before the start of job i .

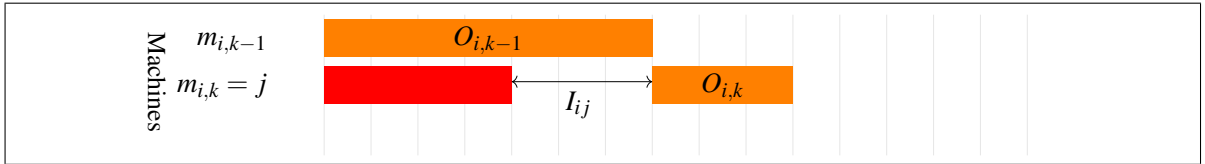


FIGURE 6.4 : Inactivity time I_{ij} on machine j before the start of job i

It is important to ensure no other operations are ready for processing in the intervening time between the end of the preceding operation and the start of operation $O_{i,k}$ on the same machine. This period is represented by the inactivity time I_{ij} on machine j .

To model these constraints linearly, we use parameter mpr_{ij} , introduced in the FIFO model section, and introduce the following binary variables :

$$x_{i_1 i_2 j} = \begin{cases} 1, & \text{if job } i_2 \text{ is executed immediately after job } i_1 \text{ on machine } j, \\ 0, & \text{otherwise.} \end{cases}$$

$x_{i,i,j} = 1$ means operation $O_{i,j}$ immediately follows $O_{i,mpr_{ij}}$.

We therefore derive the following constraints for Condition 1 :

$$\sum_{i_1 \in \text{Jobs}} x_{i_1 i_2 j} = 1, \quad \forall i_2 \in N, \forall j \in M. \quad (9)$$

$$B_{i_1 i_2 j} \geq x_{i_1 i_2 j}, \quad \forall i_1, i_2 \in N, \forall j \in M. \quad (10)$$

$$S_{ij} - (S_{i,mpr_{ij}} + p_{i,mpr_{ij}}) - L \cdot (1 - x_{i,i,j}) \leq 0, \quad \forall i \in N, \forall j \in M, mpr_{ij} \neq -1. \quad (11)$$

$$S_{i_2 j} - (S_{i_1 j} + p_{i_1 j}) + L(B_{i_1 i_2 j} + x_{i_1 i_2 j} - 2) \leq 0, \quad \forall i_1, i_2 \in N, i_1 \neq i_2, \forall j \in M. \quad (12)$$

Constraint (9) ensures each job has a single predecessor for each machine. For constraint (10) an immediate predecessor must be a predecessor. Constraints (11) and (12) ensure, for each operation of a job i on machine j , the start time of the operation $O_{j,k}$ must be either equal to the finish time of the preceding operation in the same job or equal to the finish time of the operation from another job.

For Condition 2, we have

$$S_{i_1,k} - (S_{i_2,mpr_{i_2,k}} + p_{i_2,mpr_{i_2,k}} + L(1 + B_{i_1 i_2 k} - x_{i_1 i_1 k})) \leq 0, \forall i_1 \neq i_2 \in N, \\ \forall k \in M, mpr_{i_2,k} \neq -1. \quad (13)$$

This constraint ensures if the operation of job i_1 on machine k starts immediately after the end of its preceding operation, as indicated by the variable $x_{i_1,i_1,k} = 1$, then all operations that follow it on machine k must be ready to be executed no earlier than the start time of this operation. This ensures there will be no other operations ready to be executed before the operation of job i_1 on machine k .

The mathematical formulation with the No-delay model constraint is

$$M_3 \left\{ \begin{array}{ll} \min & C_{\max} \quad (1) \\ \text{subject to} & \\ & (2), \quad (3), \quad (4), \quad (5), \quad (6), \text{ and } \quad (7), \\ & \sum_{i_1 \in \text{Jobs}} x_{i_1 i_2 j} = 1, \quad \forall i_2 \in J, \forall j \in M, \quad (9) \\ & B_{i_1 i_2 j} \geq x_{i_1 i_2 j}, \quad \forall i_1, i_2 \in J, \forall j \in M, \quad (10) \\ & 0 \geq S_{ij} - (S_{i,mpr_{ij}} + p_{i,mpr_{ij}}) - L(1 - x_{i,i,j}), \quad \forall i \in N, \forall j \in M, mpr_{ij} \neq -1, \quad (11) \\ & 0 \geq S_{i_2 j} - (S_{i_1 j} + p_{i_1 j}) + L(B_{i_1 i_2 j} + x_{i_1 i_2 j} - 2), \quad \forall i_1, i_2 \in J, i_1 \neq i_2, \forall j \in M, \quad (12) \\ & 0 \geq S_{i_1 k} - (S_{i_2, mpr_{i_2 k}} + p_{i_2, mpr_{i_2 k}} + \\ & \quad L(1 + B_{i_1 i_2 k} - x_{i_1 i_1 k})), \quad \forall i_1, i_2 \in J, i_1 \neq i_2, \forall k \in M, \\ & \quad \quad \quad mpr_{i_2 k} \neq -1. \quad (13) \end{array} \right.$$

The number of variables and constraints in the above mixed-integer programming model are $(2n + 1)nm$ and $nm(5n + 1)$, respectively. For the job shop with No-Delay constraints, the optimal makespan is referred to as $C_{\max}^{\text{ND}}$.

6.3.4 JOB SHOP WITH THE FIFO AND NO-DELAY CONSTRAINTS

In this section, we combine the FIFO and No-delay constraints into a single model. In the case of job shop scheduling models in which the FIFO constraint is imposed on the machine queues, the schedules with No-delay are not necessarily dominant. As an illustration, let us consider two scheduling solutions of an instance with 3 jobs and 4 machines. The processing times of job 1, job 2, and job 3 are $(1, 10, 0, 0)$, $(1, 1, 3, 0)$, and $(3, 2, 6, 0)$, respectively.

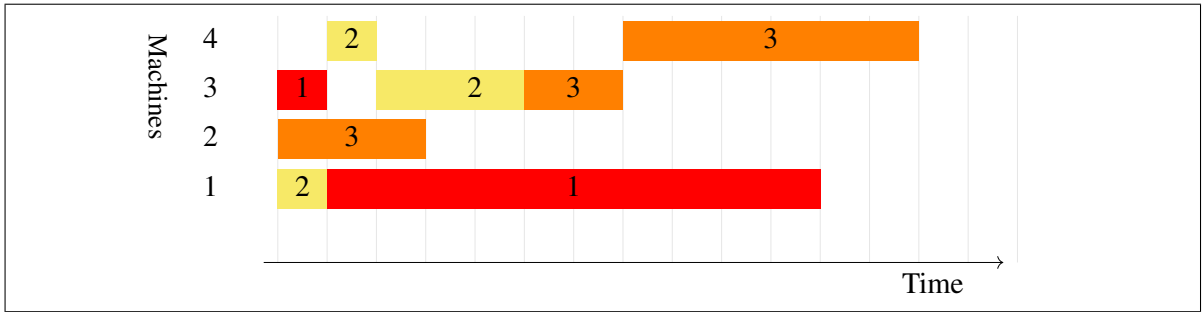


FIGURE 6.5 : Scheduling with Respect to FIFO in a job shop problem without delay

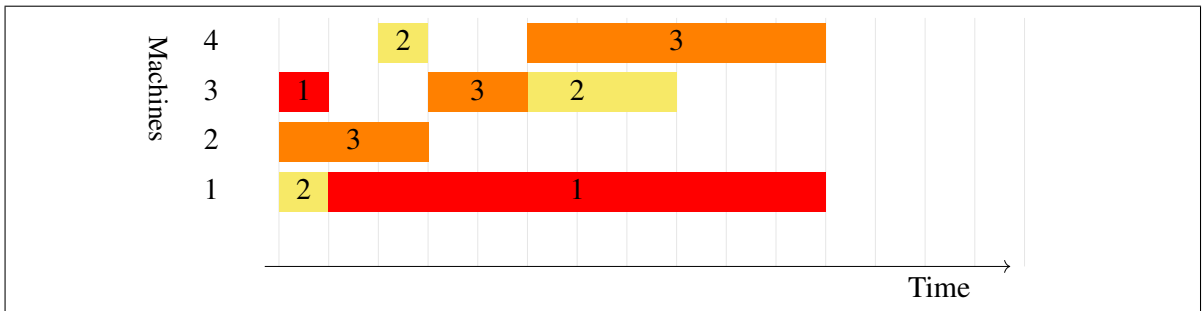


FIGURE 6.6 : Scheduling with respect to FIFO in a job shop problem with delay

To model the FIFO constraint with no-delay in a job shop problem, we add the no-delay constraints to the previously presented FIFO job shop model.

$$M_4 \quad \left\{ \begin{array}{l} \min \quad C_{\max} \\ \text{subject to} \\ (2), \quad (3), \quad (4), \quad (5), \quad (6), \quad (7), \\ (8), \quad (9), \quad (10), \quad (11), \quad (12), \text{ and } \quad (13) \end{array} \right. \quad (1)$$

The number of variables and constraints of the above mixed integer programming model are $(2n+1)nm$ and $6mn^2$, respectively. In the combined FIFO and No-Delay setting, the optimal makespan is indicated by $C_{\max}^{\text{FIFO-ND}}$.

6.3.5 JOB SHOP SCHEDULING WITH A SECONDARY CRITERION

In this section, we introduce two job shop models incorporating a secondary criterion. The first model aims to minimize FIFO violations while ensuring that $C_{\max}$ remains within $C_{\max}^{\text{JSP}}$. The second model focuses on minimizing the maximum delay while maintaining $C_{\max}$ within similar limits.

We begin with the first model, which addresses FIFO violations. This proposed model relaxes strict FIFO constraints by allowing a limited number of operation violations. The objective is to minimize these violations while keeping the makespan within an acceptable range. To achieve this, we define the variable F_{ji} as follows :

$$F_{ji} = \begin{cases} 1, & \text{if the FIFO constraint is violated for operation } O_{ij}, \\ 0, & \text{otherwise.} \end{cases}$$

We introduce the following objective function :

$$\min \sum F_{ji}.$$

Next, we modify constraint (8), which imposes the FIFO passage rule, to make it more flexible :

$$S_{i_1, \text{mpr}_{i_1 j}} + p_{i_1, \text{mpr}_{i_1 j}} + (F_{ji_1} + B_{i_1 i_2 j})L \geq S_{i_2, \text{mpr}_{i_2 j}} + p_{i_2, \text{mpr}_{i_2 j}},$$

$$\forall i_1 \neq i_2 \in J, \quad \forall j \in M, \quad \text{mpr}_{i_1 j}, \text{mpr}_{i_2 j} \neq -1.$$

In this formulation, the constraint becomes inactive if $F_{ji} = 1$.

The mathematical formulation for the Flexible FIFO JSP model is as follows :

$$M_5 \left\{ \begin{array}{l} \min \sum F_{ji} \\ \text{subject to :} \\ (2), (3), (4), (5), (6), \text{ and } (7), \\ S_{i_1, \text{mpr}_{i_1 j}} + p_{i_1, \text{mpr}_{i_1 j}} + (F_{ji_1} + B_{i_1 i_2 j})L \geq S_{i_2, \text{mpr}_{i_2 j}} + p_{i_2, \text{mpr}_{i_2 j}}, \\ \forall i_1 \neq i_2 \in J, \forall j \in M, \\ \text{mpr}_{i_1 j}, \text{mpr}_{i_2 j} \neq -1, \\ C_{\max} \leq C_{\max}^{\text{JSP}}. \end{array} \right.$$

We now present a model aimed at minimizing the maximum delay while ensuring the value of the makespan remains within $C_{\max}^{\text{JSP}}$. The variable $D_{\max}$ is introduced to represent the maximum voluntary delay between operations in the schedule. By minimizing $D_{\max}$, the model seeks to control voluntary idleness while maintaining the makespan within its limits. This approach relaxes the No-Delay constraints introduced in 6.3.3 by allowing a minimal level of operation delays. The mathematical

model is therefore as follows :

$$M_6 \left\{ \begin{array}{l} \min \quad D_{\max} \\ \text{subject to} \\ (2), \quad (3), \quad (4), \quad (5), \quad (6), \quad (7), \text{ and } \quad (15) \\ D_{\max} \geq S_{ij} - (S_{i,\text{mpr}_{ij}} + p_{i,\text{mpr}_{ij}}) - L(1 - x_{i,i,j}), \quad \forall i \in J, \forall j \in M, \text{mpr}_{ij} \neq -1, \quad (16) \\ D_{\max} \geq S_{i_2j} - (S_{i_1j} + p_{i_1j}) + M(B_{i_1i_2j} + x_{i_1i_2j} - 2), \quad \forall i_1 \neq i_2 \in J, \forall j \in M, \quad (17) \\ D_{\max} \geq S_{i_1k} - (S_{i_2,\text{mpr}_{i_2k}} + p_{i_2,\text{mpr}_{i_2k}} + \\ \quad M(1 + B_{i_1i_2k} - x_{i_1i_1k})), \quad \forall i_1 \neq i_2 \in J, \forall k \in M, \\ \quad \text{mpr}_{i_2k} \neq -1, \quad (18) \\ C_{\max} \leq C_{\max}^{\text{JSP}}. \quad (15) \end{array} \right.$$

We are now ready to prove model M_6 minimizes the maximum delay.

Théorème 6.3.2. *Model M_6 minimizes the maximum delay in the job shop problem.*

Proof. Without loss of generality, let us consider machine j and two consecutive operations O_{ji_1} and O_{ji_2} . Let the idle time of this machine, between the end of operation O_{ji_1} and the start of operation O_{ji_2} , be denoted by $I_{i_1i_2j} = S_{i_2j} - C_{i_1j}$.

The delay caused during this idle period is calculated as the minimum between this idle period $I_{i_1i_2j}$ and the maximum of the availability of the tasks that succeed i_1 on machine j , minus the end time of the idle period, which is the start time of task i_2 on machine j , denoted S_{i_2j} , that is

$$\min(I_{i_1i_2j}, \max_{\forall h \in S}(S_{ji_2} - C_{\text{mpr}_{hj}})),$$

where $C_{\text{mpr}_{hj}}$ is the completion time of the previous operation O_{jh} for job h , with $h \in S$, S being the set of tasks executed after i_1 on machine j .

In order to minimize the maximum delay, we must minimize $D_{\max}$ defined as follows :

$$D_{\max} = \max_{\forall i_1, i_2 \in J, \forall j \in M} \left(\min(I_{i_1 i_2 j}, \max_{\forall h \in S} (S_{i_2 j} - C_{\text{mpr}_{jh} h})) \right).$$

In the minimization problem with respect to $D_{\max}$, the previous equality is equivalent to

$$D_{\max} \geq \min(I_{i_1 i_2 j}, \max_{\forall h \in S} (S_{i_2 j} - C_{\text{mpr}_{jh} h})) \quad \forall i_1, i_2 \in J, \forall h \in S, \forall j \in M.$$

Now, let us show constraints (16), (17), and (18), along with their interaction with other constraints in the model, allow us to linearize the minimization of $D_{\max}$. From constraint (9), we can distinguish two cases as follows.

- **Case 1.** $x_{i_2 i_2 j} = 1$: This implies, in the mathematical Model M_6 , constraints (16) and (18) are active, leading to the following :

$$D_{\max} \geq S_{i_2 j} - (S_{i_2 \text{mpr}_{i_2 j}} + p_{i_2 \text{mpr}_{i_2 j}}).$$

In addition, for every successor of i_2 (where $B_{h, i_2, j} = 0$), we have

$$D_{\max} \geq S_{i_2 j} - (S_{h \text{mpr}_{hk}} + p_{h \text{mpr}_{hj}}) \quad \text{with} \quad \text{mpr}_{hj} \neq -1.$$

- **Case 2.** $x_{i_2 i_2 j} = 0$: In this case, constraint (17) is active, while constraints (16) and (18) are not. For the immediate predecessor, we have $x_{i_1 i_2 j} = 1$ and $B_{i_1 i_2 j} = 1$. We therefore get

$$D_{\max} \geq S_{i_2 j} - (S_{i_1 j} + p_{i_1 j}).$$

Given we are dealing with a minimization problem, we have $x_{i_1 i_2 j} = 1$ if, and only if, the idle time $I_{i_1 i_2 j}$ satisfies the following conditions :

For the immediate successor i_2 of i_1 , we have

$$I_{i_1 i_2 j} = S_{i_2 j} - (S_{i_1 j} + p_{i_1 j}) \geq S_{i_2 j} - (S_{j \text{mpr}_{i_2 j}} + p_{\text{mpr}_{i_2 j}}).$$

For any other successors of i_1 , we have

$$I_{i_1 i_2 j} = S_{i_2 j} - (S_{i_1 j} + p_{i_1 j}) \geq S_{i_2 j} - (S_{h \text{mpr}_{i_2 j}} + p_{\text{mpr}_{i_2 j}}),$$

where h represents another successor of i_1 . Otherwise, $x_{i_2 i_2 j} = 0$ if either

$$I_{i_1 i_2 j} = S_{i_2 j} - (S_{i_1 j} + p_{i_1 j}) < S_{i_2 j} - (S_{\text{mpr}_{i_2 j}} + p_{\text{mpr}_{i_2 j}}),$$

or there exists a successor h of i_1 such that

$$I_{i_1 i_2 j} = S_{i_2 j} - (S_{i_1 j} + p_{i_1 j}) < S_{i_2 j} - (S_{h \text{mpr}_{i_2 j}} + p_{h \text{mpr}_{i_2 j}}).$$

We therefore have

$$D_{\max} \geq \min(I_{i_1 i_2 j}, \max_{\forall h \in S} (S_{i_2, h} - C_{\text{mpr}_{i_2 j} h})) \quad \forall i_1, i_2 \in J, \forall h \in S, \forall j \in M.$$

The possible configurations are as pictured by Figure 6.7 and 6.8.

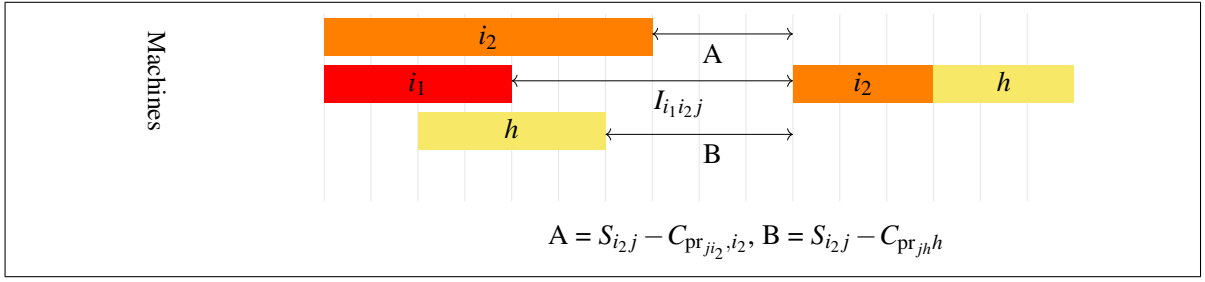


FIGURE 6.7 : Configuration for $x_{i_2 i_2 j} = 1$

Figure 6.7 displays the situation where $x_{i_2 i_2 j} = 1$. In this case, we have

$$I_{i_1 i_2 j} > S_{i_2 j} - C_{mpr_{i_2 j}, i_2} \quad \text{and} \quad I_{i_1 i_2 j} \geq S_{i_2 j} - C_{mpr_{hj}, h}.$$

This indicates the delay during this period of machine inactivity is

$$D_{\max} \geq \max(S_{i_2 j} - C_{mpr_{hj}, h}, S_{i_2 j} - C_{mpr_{i_2 j}, i_2}).$$

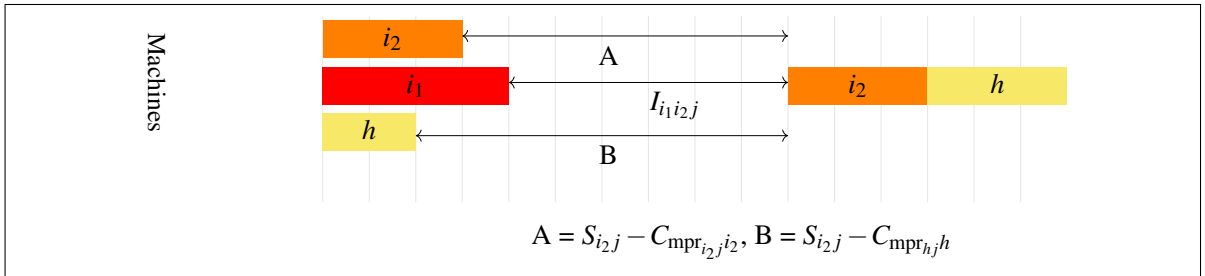


FIGURE 6.8 : Configuration for $x_{i_2 i_2 j} = 0$

Figure 6.8 displays the case where $x_{i_2 i_2 j} = 0$ and the idle time $I_{i_1 i_2 j}$ is smaller than either A or B , *i.e.*,

$$I_{i_1 i_2 j} \leq S_{i_2 j} - C_{mpr_{i_2 j}, i_2} \quad \text{or} \quad I_{i_1 i_2 j} \leq S_{i_2 j} - C_{mpr_{hj}, h}.$$

This indicates the delay during this period of machine inactivity is

$$D_{\max} \geq I_{i_1 i_2 j}.$$

■

6.3.6 JOB SHOP SCHEDULING WITH CONSTRAINTS AND A SECONDARY CRITERION

In this section, we introduce two job shop models with constraints (FIFO, No-Delay) and a secondary criterion, *viz.*

- Job shop with FIFO constraints and minimization of the delays.
- Job shop with No-Delay constraints and minimization of FIFO violations.

We begin with the first model, which aims to minimize the maximum delay while ensuring $C_{\max}$ remains within acceptable bounds in a FIFO job shop setting.

The proposed model relaxes the strict No-delay constraints by permitting a limited number of operation violations, providing greater flexibility in scheduling while still adhering to FIFO constraints where possible. The mathematical formulation of the model is as follows :

$$M_7 \quad \left\{ \begin{array}{l} \min \quad D_{\max} \\ \text{subject to} \\ (2), \quad (3), \quad (4), \quad (5), \quad (6), \quad (7), \text{ and } \quad (8) \\ C_{\max} \leq C_{\max}^{\text{FIFO}} \end{array} \right. \quad (15')$$

Next, we turn to the second model, which focuses on minimizing FIFO violations while ensuring the value of the makespan remains within acceptable bounds in a No-Delay job shop setting. The proposed model relaxes the strict FIFO constraints by permitting a limited number of operation violations. The mathematical formulation for the Flexible FIFO and No-delay JSP model is as follows :

$$M_7 \left\{ \begin{array}{l} \min \quad \sum F_{ji} \\ \text{subject to :} \\ (2), \quad (3), \quad (4), \quad (5), \quad (6), \quad (9), \quad (10), \quad (11), \quad (12), \quad (13), \quad (14), \\ C_{\max} \leq C_{\max}^{\text{ND}} \end{array} \right. \quad (15'')$$

6.4 COMPUTATIONAL STUDY

We conducted the experiments using IBM CPLEX 20.1 solver on 11 Dell PowerEdge R740 machines, each equipped with 512GB of memory and Intel Xeon Gold 6258R CPUs operating at 2.70 GHz. This computational setup ensured efficient resolution of large-scale instances. The models were evaluated using a range of classical instances from the literature, with job configurations in the set {6, 10, 15} and machine configurations in {5,6, 10, 15}. A comparison of the optimal results produced by our models was conducted with respect to benchmark instances from classical job shop scheduling problems ADAMS et al., [1988a](#); APPLEGATE et COOK, [1991d](#); FISHER et THOMPSON, [1963a](#); LAWRENCE, [1984b](#); E. D. TAILLARD, [1993a](#).

6.4.1 METHODOLOGY

We first introduce the main notations used to assess the impact of FIFO and No-Delay constraints. We define the following key performance indicators :

- **FIFO Indicator** : Measures the percentage of FIFO violations relative to the total number of operations :

$$\text{FIFO Indicator} = \frac{\text{Number of Violations}}{n \times m} \times 100.$$

where n is the number of jobs and m is the number of machines.

- **Delay Impact Indicator (Ind2)** : Evaluates the relative delay compared to the average processing time :

$$\text{Ind2} = \frac{\text{Delay}}{\text{Average Processing Time}} \times 100.$$

This indicator provides insight into the level of voluntary idleness in the schedule.

- **GAP Analysis** : Quantifies the difference in makespan between models :

- **FIFO vs Classical JSP** :

$$\text{GAP (\%)} = \left(\frac{C_{\max}^{\text{FIFO}} - C_{\max}^{\text{JSP}}}{C_{\max}^{\text{JSP}}} \right) \times 100.$$

- **No-Delay vs Classical JSP** :

$$\text{GAP (\%)} = \left(\frac{C_{\max}^{\text{ND}} - C_{\max}^{\text{JSP}}}{C_{\max}^{\text{JSP}}} \right) \times 100.$$

- **No-Delay FIFO vs Classical JSP** :

$$\text{GAP (\%)} = \left(\frac{C_{\max}^{\text{FIFO-ND}} - C_{\max}^{\text{JSP}}}{C_{\max}^{\text{JSP}}} \right) \times 100.$$

The methodology is implemented in three steps :

- Calculate the optimal makespan for the JSP model under each constraint (FIFO No-Delay, No-Delay, and FIFO) and compare the results to the classical JSP.
- Minimize a secondary criterion (FIFO violations, delays) within the JSP model while fixing the makespan to the optimal value $C_{\max}^{\text{JSP}}$. The minimized values of the secondary criterion are then

compared to the FIFO violations and delays calculated for ten different solutions obtained from the classical JSP, each optimized for makespan minimization.

- This step is divided into two parts : first, comparing the minimum FIFO violations between the classical JSP and the JSP with No-Delay constraints, and second, comparing the minimum delay between the classical JSP and the JSP with FIFO constraints.

Figure 6.9 illustrates the models and indicators used in each step.

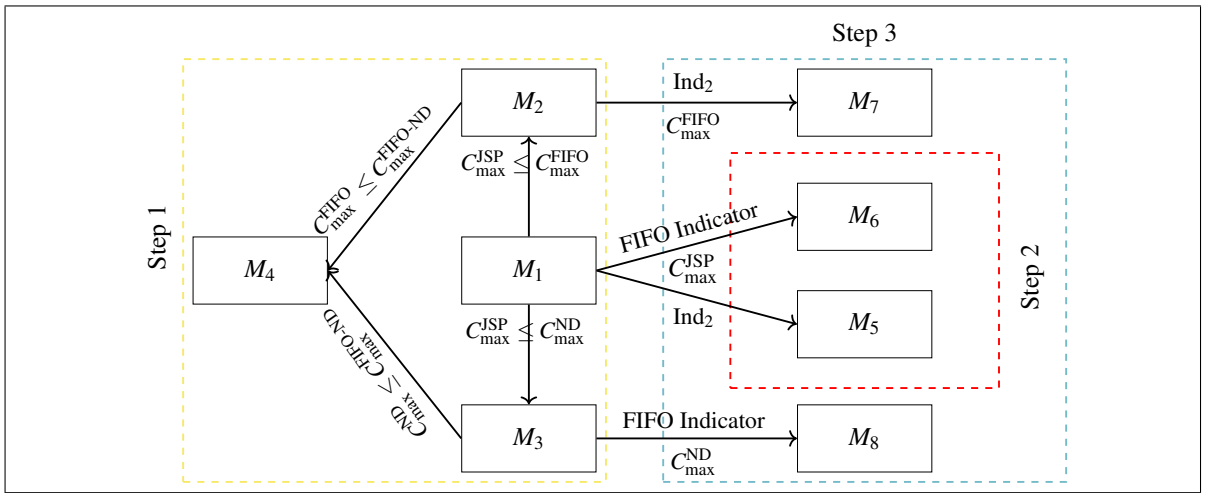


FIGURE 6.9 : Illustration of the models, constraints, and indicators used across the three methodological steps.

To obtain the optimal solution for the different objectives (makespan, delays, FIFO violations), we first solve the mathematical models introduced in Section 3. For instances where the optimal solution cannot be obtained in a reasonable time, we apply the ε -constraint method, primarily to the No-Delay and No-Delay-FIFO job shop models, which has proven more effective than a direct approach. This method focuses on minimizing the maximum voluntary idleness ($D_{\max}$) while systematically exploring trade-offs with the makespan. By iteratively adjusting the makespan bound until $D_{\max} = 0$, the algorithm determines an optimal makespan corresponding to zero delays. The detailed pseudo-code of this method is presented in Algorithm 1.

Algorithm 1 Incremental Adjustment of $C_{\max}$

```
1: Input : Initial lower bound for  $C_{\max}$ , maximum allowed  $C_{\max}$ , step size  $\Delta$ , stopping
   criterion (delay = 0)
2: Output : Pareto front of  $C_{\max}$  vs. delay
3:
4: Initialize  $C_{\max}$  = initial lower bound
5: Initialize ParetoFront =  $\emptyset$ 
6: While delay > 0 and  $C_{\max} \leq$  maximum allowed  $C_{\max}$ 
7: Solve JSP with the current  $C_{\max}$  using the  $\varepsilon$ -constraint method
8: Calculate the corresponding delay for the current solution
9: Add the pair ( $C_{\max}$ , delay) to ParetoFront
10: IF (delay = 0)
11: Break
12: EndIf
13: Increment  $C_{\max}$  by  $\Delta$ 
14: EndWhile
15: Return ParetoFront
```

6.4.2 RESULTS

This section presents the results of our computational experiments, highlighting the impact of various constraints(FIFO, No-Delay, and FIFO No-Delay) on makespan, delay, and FIFO violations. The aim is to provide a comprehensive evaluation of the proposed models under these constraints and to analyze their trade-offs compared to the classical Job Shop Problem (JSP) model.

JOB SHOP MODEL WITH CONSTRAINTS

This section presents the impact of different constraints (FIFO, No-delay, FIFO with No-delay) on the makespan, denoted hereafter $C_{\max}$, compared to the classical JSP model.

Table 6.1 summarizes the results.

TABLE 6.1 : Summarized Table

Statistic	FIFO	FIFO with No-delay	No-delay
Number of instances solved to optimality	38	29	30
Average GAP (%)	1.19	2.28	0.64
Maximum GAP (%)	3.51	5.226	2.26
Minimum GAP (%)	0	0	0
Standard Deviation (%)	1	1.66	0.71

Figure 6.10 visually illustrates the differences in GAP values for FIFO, FIFO with No-delay, and No-delay constraints

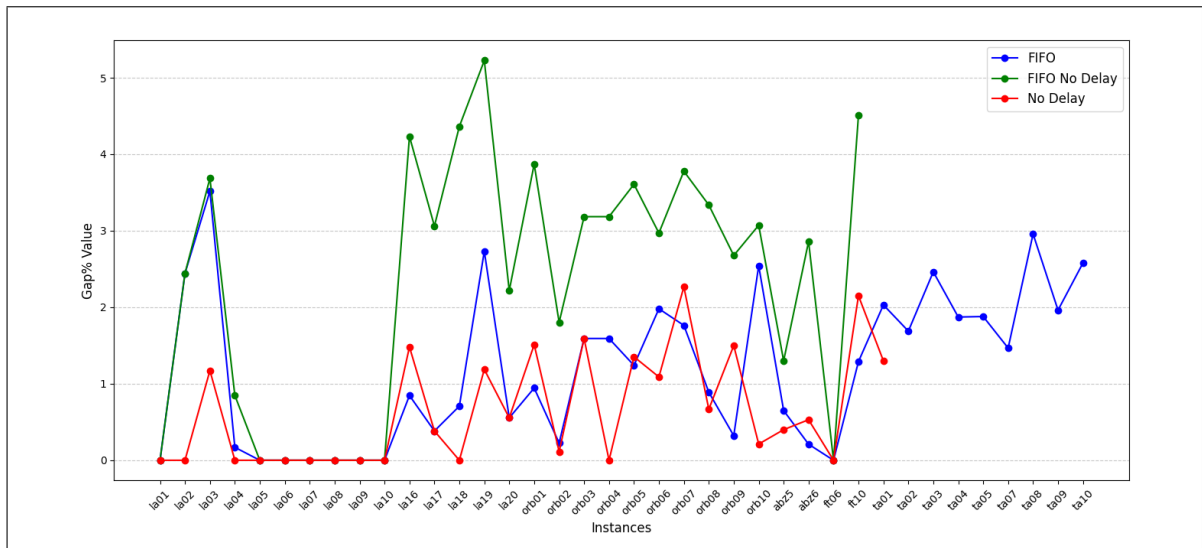


FIGURE 6.10 : Comparison of GAP values for FIFO, FIFO with No-delay, and No-delay constraints

The analysis of the tested instances reveals that imposing the FIFO constraint has a limited impact on the makespan. The average gap observed is 1.19%, with a maximum gap of 3.51% and a minimum gap of 0%. These results suggest that, in most cases, the makespan remains close to the optimal value produced by the classical Job Shop Problem (JSP) model. Similarly, applying the No-delay constraint alone results in a minimal average gap of 0.64%, with a maximum gap of 2.26% and a minimum gap

of 0%. However, when both the No-delay and FIFO constraints are applied together, the makespan gap significantly increases to an average of 2.28%, which is more than three times the gap observed with the No-delay constraint alone and twice the gap observed with the FIFO constraint alone. This suggests that combining No-delay and FIFO constraints can lead to a considerable increase in makespan.

JOB SHOP MODEL WITH A SECONDARY CRITERION

In this section, we consider a secondary criterion such as the minimization of the number of FIFO violations and the minimization of the delays. To compare the results, we generated 10 optimal solutions for the classical JSP model.

The delay and number of FIFO violations were calculated for these solutions and compared to those obtained by solving M_5 (minimizing FIFO violations) and M_6 (minimizing delays), using the optimal makespan from the classical JSP model ($C_{\max}^{\text{JSP}}$) as a lower bound.

The results are pictured in Figure 6.11 and Figure 6.12. These results show the following :

- Minimizing delays (M_6) is achieved by imposing the makespan bound determined by the optimal solution of the classical JSP as a constraint. On average, this results in significantly reduced delays (from 189.66% to 17.11%), while maintaining the optimal makespan.
- Minimizing FIFO violations (M_5) follows the same approach, using the optimal makespan from the classical JSP as a bound. On average, this reduces the number of violations from 11.18% to 2.86%, providing a more feasible solution in practice.

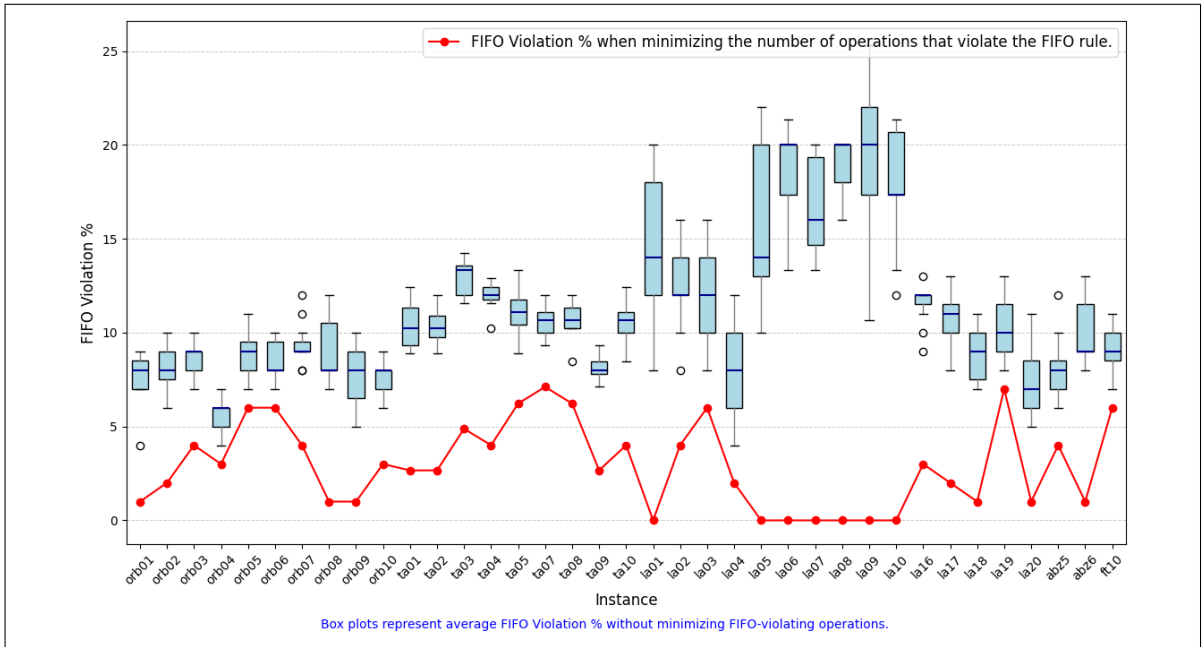


FIGURE 6.11 : Minimum FIFO violations (while preserving $C_{\max}^{\text{JSP}}$) vs. average violations observed over 10 $J//C_{\max}$ optimizations.

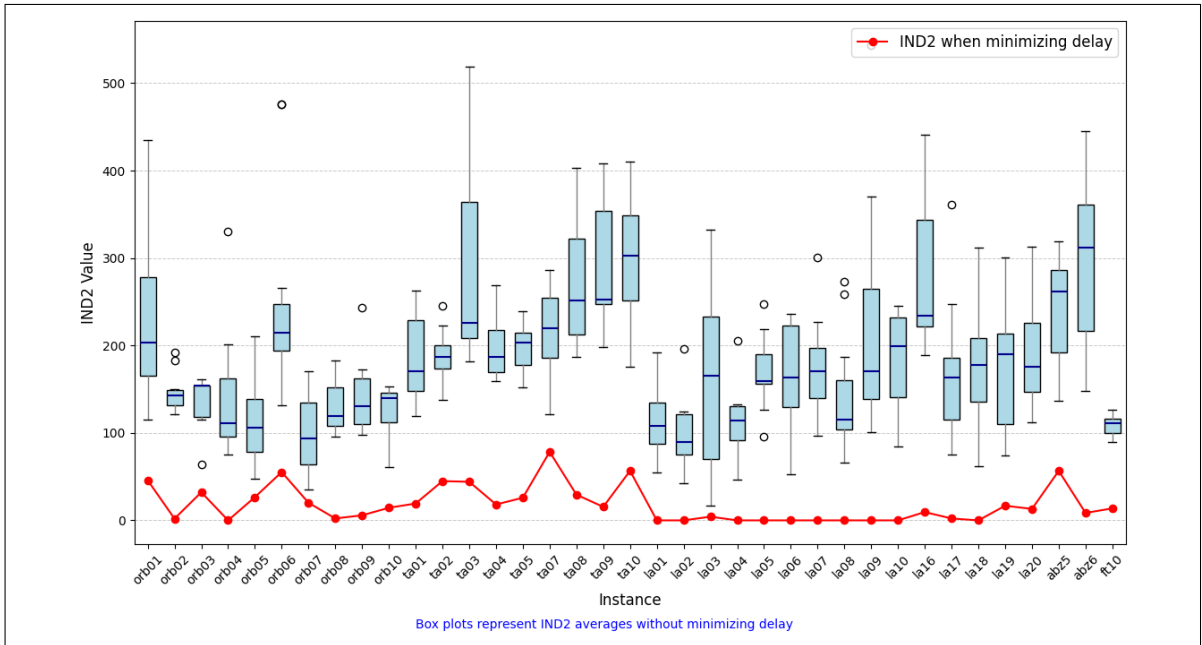


FIGURE 6.12 : Minimum delay (while preserving $C_{\max}^{\text{JSP}}$) vs. average delay over 10 $J//C_{\max}$ optimizations.

JOB SHOP MODEL WITH CONSTRAINTS AND A SECONDARY CRITERION

In this section, we consider two job shop models with constraints (FIFO, No-delay) with a secondary criterion, *viz.*

- Job shop with FIFO constraints and minimization of delays.
- Job shop with No-Delay constraints and minimization of FIFO violations.

We begin by analyzing the job shop model with FIFO constraints and the secondary criterion to minimize delays. We analyze the minimal delays obtained by minimizing the model M_7 , where $C_{\max}$ is bounded by the optimal solution of $C_{\max}^{\text{FIFO}}$, and compare them to the minimal delays achieved by minimizing M_6 (without the FIFO constraint), where $C_{\max}$ is bounded by the optimal $C_{\max}^{\text{JSP}}$.

The results are illustrated in Figure 6.13, which compares the levels of *Ind2* (delay indicator) between the models with and without the FIFO constraint, highlighting the impact of enforcing FIFO on the perceived delay.

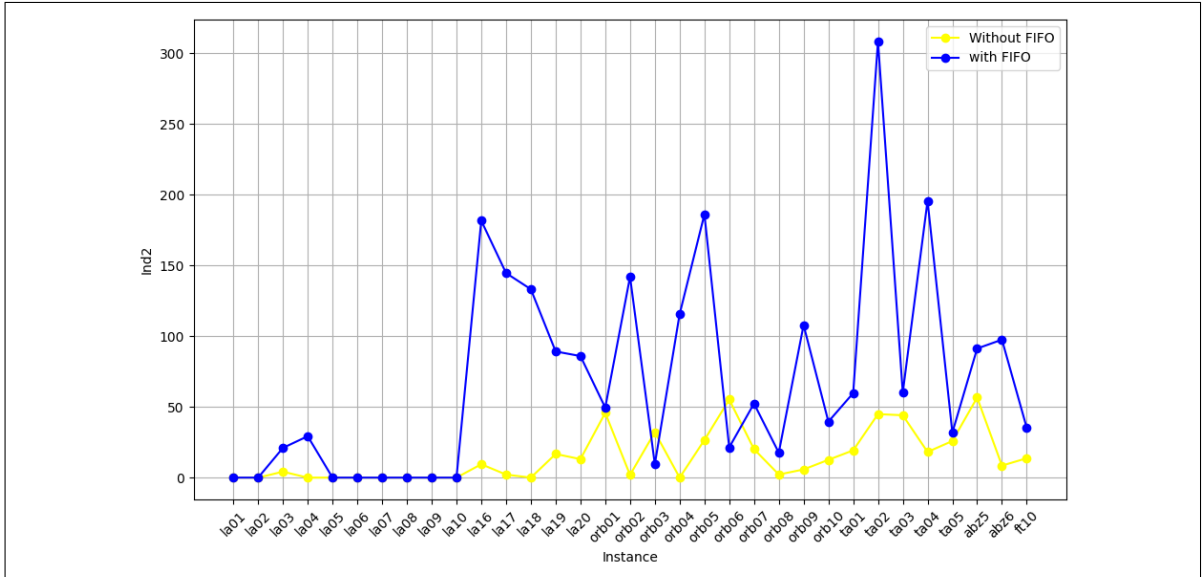


FIGURE 6.13 : Comparison of delays with and without imposing the FIFO constraint

As shown in Figure 6.13, maintaining the optimal $C_{\max}^{\text{FIFO}}$ in the FIFO model necessitates higher delays compared to the JSP model.

Let us now turn to the job shop model with No-delay constraints and the secondary criterion of minimizing FIFO violations. we analyze the minimal number of FIFO violations obtained by minimizing the model M_8 , where $C_{\max}$ is bounded by the optimal solution of $C_{\max}^{\text{ND}}$, and compare them to the minimal number of FIFO violations achieved by minimizing M_5 (without the No-delay constraint), where $C_{\max}$ is bounded by the optimal $C_{\max}^{\text{JSP}}$.

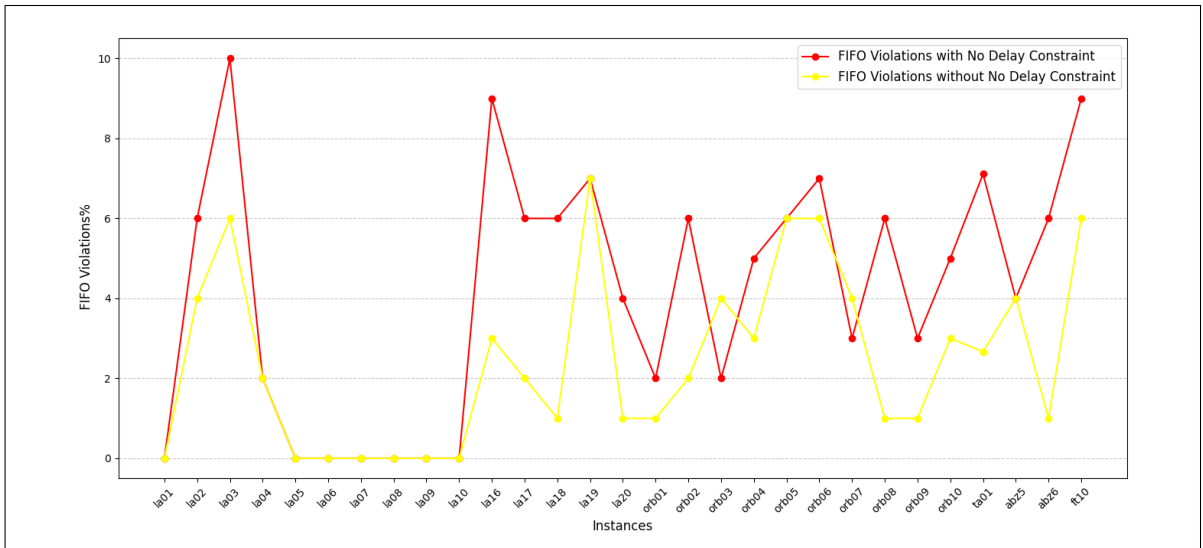


FIGURE 6.14 : Comparison of FIFO violations with and without the No-delay Constraint

As illustrated in Figure 6.14, the introduction of the No-delay constraint significantly increases the number of FIFO violations compared to when it is not imposed.

6.5 CONCLUSION

In this study, we introduced new scheduling models incorporating important constraints in the services environment, *viz.* FIFO, No-delay, and FIFO with No-delay. These constraints are critical in the service environment as they directly impact customer experience.

In the first phase of the analysis, the focus was on the effect of these constraints on the makespan. The findings indicate that individually enforcing FIFO or No-Delay constraints has a relatively minor impact, yielding average gap values of 1.27% and 0.63%, respectively. In contrast, jointly applying both constraints significantly increases the gap to 2.32%. The second phase of the study considered secondary objectives (the minimization of delays and FIFO violations) while preserving the optimal makespan ($C_{\max}^{\text{JSP}}$). Ten optimal solutions were generated from the classical JSP model, and their delays and FIFO violations were compared with those produced by independently minimizing each criterion. The results reveal substantial improvements : minimizing delays reduced the average delay from 189.66% to 17.11%, while minimizing FIFO violations decreased the violation rate from 11.18% to 2.86%. In the third phase, we examined job shop models that integrate FIFO or No-Delay constraints with secondary objectives. Under FIFO constraints with delay minimization, maintaining $C_{\max}^{\text{FIFO}}$ resulted in an average delay of 69.89%, compared to 17.11% in the classical JSP model. Similarly, introducing No-Delay constraints while minimizing FIFO violations, and maintaining $C_{\max}^{\text{ND}}$, increased the average violation FIFO rate to 4.66%, compared to 2.86% in the model without No-Delay constraints.

Overall, strictly enforcing FIFO constraints raises the makespan and exacerbates delays. Similarly, imposing No-Delay constraints increases the frequency of FIFO violations, and simultaneously enforcing both constraints leads to a more pronounced increase in makespan. Future research of interest that addresses appointment scheduling problems in services, particularly those using mixed integer programming (MIP) approaches with respect to the makespan as the primary objective, should consider including the minimization of the delay and the reduction of the FIFO violations in their formulations. Solver-generated solutions may sometimes result in excessive delays and violations, which may have a great impact on the customer satisfaction. By integrating these elements into the future models, it will be possible to ensure the proposed solutions not only optimize the makespan, but also maintain acceptable levels of customer service quality. While most prior studies on heuristics and meta-heuristics have focused on the No-Delay scheduling, this research emphasizes the importance of voluntary idleness in developing effective FIFO-based approaches. To address larger-scale instances, future research should explore heuristic strategies that incorporate voluntary idleness, particularly for FIFO-based scheduling

in service environments.

Acknowledgments : This research was partially funded by the Natural Sciences and Engineering Research Council of Canada (NSERC).

DATA AVAILABILITY STATEMENT

The data that support the findings of this study are available from the corresponding author, R. Ouchene, upon request.

BIBLIOGRAPHY

- ADAMS, J., BALAS, E., & ZAWACK, D. (1988a). The shifting bottleneck procedure for job shop scheduling. *Management Science*, 34(3), 391-401.
- AGNIHOTRI, S., CAPPANERA, P., NONATO, M., & VISINTIN, F. (2024b). Appointment scheduling in surgery pre-admission testing clinics. *Omega*, 123, 102994.
- ALA, A., SIMIC, V., PAMUCAR, D., & TIRKOLAEI, E. B. (2022b). Appointment scheduling problem under fairness policy in healthcare services : fuzzy ant lion optimizer. *Expert systems with applications*, 207, 117949.
- ALKHATEEB, F., ABED-ALGUNI, B. H., & AL-ROUSAN, M. H. (2022b). Discrete hybrid cuckoo search and simulated annealing algorithm for solving the job shop scheduling problem. *Journal of Supercomputing*, 78(4), 4799-4826.
- APPLEGATE, D., & COOK, W. (1991d). A computational study of the job-shop scheduling problem. *ORSA Journal on computing*, 3(2), 149-156.
- BOWMAN, E. H. (1959b). The schedule-sequencing problem. *Operations research*, 7(5), 621-624.
- BURDETT, R. L., & KOZAN, E. (2018b). An integrated approach for scheduling health care activities in a hospital. *European Journal of Operational Research*, 264(2), 756-773.
- CAI, L., LI, W., LUO, Y., & HE, L. (2023). Real-time scheduling simulation optimisation of job shop in a production-logistics collaborative environment. *International Journal of Production Research*, 61(5), 1373-1393.
- CASTRO, E., & PETROVIC, S. (2012). Combined mathematical programming and heuristics for a radiotherapy pre-treatment scheduling problem. *Journal of Scheduling*, 15, 333-346.
- CHIANG, T.-C., & FU, L.-C. (2007b). Using dispatching rules for job shop scheduling with due date-based objectives. *International Journal of Production Research*, 45(14), 3245-3262.
- CONFORTI, D., GUERRIERO, F., GUIDO, R., CERINIC, M. M., & CONFORTI, M. L. (2011b). An optimal decision making model for supporting week hospital management. *Health care management science*, 14, 74-88.
- DELL'AMICO, M., & TRUBIAN, M. (1993b). Applying tabu search to the job-shop scheduling problem. *Annals of Operations Research*, 41(3), 231-252.
- DIAMANT, A., MILNER, J., & QUERESHY, F. (2018b). Dynamic patient scheduling for multi-appointment health care programs. *Production and Operations Management*, 27(1), 58-79.
- FISHER, H., & THOMPSON, G. L. (1963a). Probabilistic Learning Combinations of Local Job-Shop Scheduling Rules. In *Industrial Scheduling*. Prentice-Hall.
- FONTES, D. B., HOMAYOUNI, S. M., & GONÇALVES, J. F. (2023b). A hybrid particle swarm optimization and simulated annealing algorithm for the job shop scheduling problem with transport resources. *European Journal of Operational Research*, 306(3), 1140-1157.

- FRENCH, S. (1982c). *Sequencing and Scheduling : An Introduction to the Mathematics of the Job-Shop*. Ellis Horwood Ltd.
- GUPTA, A. K., & SIVAKUMAR, A. I. (2006). Job shop scheduling techniques in semiconductor manufacturing. *The International Journal of Advanced Manufacturing Technology*, 27, 1163-1169.
- GUPTA, M., KO, H.-J., & MIN, H. (2002). TOC-based performance measures and five focusing steps in a job-shop manufacturing environment. *International Journal of Production Research*, 40(4), 907-930.
- HAUPT, R. (1989c). A survey of priority rule-based scheduling. *Operations-Research-Spektrum*, 11(1), 3-16.
- JACKSON, J. (1956). An extension of Johnson's results on job lot scheduling. *Naval Research Logistic Quarterly*, 3, 201-203.
- JOHNSON, S. M. (1954d). Optimal two-and three-stage production schedules with setup times included. *textitNaval research logistics quarterly*, 1(1), 61-68.
- KAZEMIAN, P., SIR, M. Y., VAN OYEN, M. P., LOVELY, J. K., LARSON, D. W., & PASUPATHY, K. S. (2017b). Coordinating clinic and surgery appointments to meet access service levels for elective surgery. *Journal of Biomedical Informatics*, 66, 105-115.
- KORTBEEK, N., van der VELDE, M., & LITVAK, N. (2017). Organizing multidisciplinary care for children with neuromuscular diseases at the Academic Medical Center, Amsterdam, *Health Systems*, 6, 209-225.
- KRISHNA, K., GANESHAN, K., & RAM, D. J. (1995b). Distributed simulated annealing algorithms for job shop scheduling. *IEEE Transactions on Systems, Man, and Cybernetics*, 25(7), 1102-1109.
- KU, W.-Y., & BECK, J. C. (2016b). Mixed integer programming models for job shop scheduling : A computational analysis. *Computers & Operations Research*, 73, 165-173.
- LAWRENCE, S. (1984b). *Resource Constrained Project Scheduling : An Experimental Investigation of Heuristic Scheduling Techniques (Supplement)* (rapp. tech.). Carnegie-Mellon University.
- LEEFTINK, A., VliegEN, I., & HANS, E. W. (2019b). Stochastic integer programming for multi-disciplinary outpatient clinic planning. *Health Care Management Science*, 22, 53-67.
- LIANG, B., TURKCAN, A., CEYHAN, M. E., & STUART, K. (2015c). Improvement of chemotherapy patient flow and scheduling in an outpatient oncology clinic. *International Journal of Production Research*, 53(24), 7177-7190.
- LODI, A., & TUBERTINI, P. (2016). Pre-operative Activities and Operating Theater Planning in Emilia-Romagna, Italy. In K. FUJISAWA, Y. SHINANO & H. WAKI (Éd.), *Optimization in the Real World*. Springer.
- MANNE, A. S. (1960c). On the job-shop scheduling problem. *Operations research*, 8(2), 219-223.
- NARASSIMA, M., VASUDEVAN, S. K., & ANBUUDAYASANKAR, S. (2022b). Development of a priority dispatching rule and novel production scheduling algorithm for a job shop. *International Journal of Computer Aided Engineering and Technology*, 16(4), 417-449.

- PAN, C.-H. (1997b). A study of integer programming formulations for scheduling problems. *International Journal of Systems Science*, 28(1), 33-41.
- PINEDO, M. (2002d). *Scheduling : Theory, Algorithms and Systems*, Springer, New York, p. 15.
- REZAEIAHARI, M., & KHASAWNEH, M. T. (2017b). An optimization model for scheduling patients in destination medical centers. *Operations Research for Health Care*, 15, 68-81.
- SAADANI, N., BAHROUN, Z., & BOURAS, A. (2014b). A Linear Mathematical Model for Patients' Activities Scheduling on Hospital Resources. *Proceedings of the 2014 International Conference on Control, Decision and Information Technologies (CoDIT)*, 74-80.
- SALIDO, M. A., ESCAMILLA, J., GIRET, A., & BARBER, F. (2016b). A genetic algorithm for energy-efficiency in job-shop scheduling. *The International Journal of Advanced Manufacturing Technology*, 85, 1303-1314.
- TAILLARD, E. D. (1993a). Benchmarks for basic scheduling problems. *European Journal of Operational Research*, 64(2), 278-285.
- VOGL, P., BRAUNE, R., & DOERNER, K. F. (2019c). Scheduling recurring radiotherapy appointments in an ion beam facility : considering optional activities and time window constraints. *Journal of Scheduling*, 22(2), 137-154.
- WAGNER, H. M. (1959b). An integer linear-programming model for machine scheduling. *Naval research logistics quarterly*, 6(2), 131-140.
- XIAO, L., ZHEN, L., LAPORTE, G., BALDACCI, R., & WANG, C. (2022b). A column generation-based heuristic for a rehabilitation patient scheduling and routing problem. *Computers & Operations Research*, 148, 105970.
- YAN, B., BRAGIN, M. A., & LUH, P. B. (2018b). Novel formulation and resolution of job-shop scheduling problems. *IEEE Robotics and Automation Letters*, 3(4), 3387-3393.
- ZHOU, S., & YUE, Q. (2019b). Appointment scheduling for multi-stage sequential service systems with stochastic service durations. *Computers & Operations Research*, 112, 104757.

CONCLUSION

Tout au long de ce travail, nous avons mis en évidence l'importance de la notion de permutation et de ses propriétés, ainsi que les implications qui en découlent dans différents environnements d'ordonnancement d'atelier. Dans un premier temps, nous avons montré que la présence d'opérations manquantes au sein d'un flow shop suscite une réflexion nouvelle sur la définition même de la permutation : les différentes interprétations existantes (fondées sur la transitivité, la discipline FIFO) ne sont plus forcément équivalentes. Cette remise en question permet d'établir un cadre formel visant à clarifier les conditions d'équivalence ou de différenciation entre les diverses définitions de permutation.

Ensuite, nous avons étudié le compromis entre la complexité de résolution et la performance des solutions dans le contexte des flow shops avec opérations manquantes. Nous avons constaté que, dans ce cadre, l'écart entre les approches à permutation et non-permutation, que ce soit en termes de makespan ou de moyenne des temps de fin de traitement est nettement plus important que dans le cas où aucune opération manquante n'est présente. Par ailleurs, cet écart varie d'une instance à l'autre : il peut être négligeable pour certaines configurations, alors qu'il se révèle très significatif pour d'autres. Afin de détecter ces situations, nous avons identifié plusieurs indicateurs structurels d'instance. L'objectif était de déterminer dans quelles conditions adapter une approche non-permutation, malgré sa complexité accrue, offre un avantage réel, et quand, au contraire, l'approche à permutation demeure préférable en raison de son coût de calcul moindre et d'un écart de performance potentiellement négligeable. Notre étude a ainsi mis en lumière un nouvel indicateur, noté IND_4 , particulièrement pertinent pour anticiper ces situations. Les tests statistiques ont confirmé une forte corrélation entre cet indicateur et l'écart de performance, attestant de sa valeur pour orienter le choix entre les deux approches.

Enfin, nous avons élargi notre étude de la permutation à la problématique du job shop. En nous appuyant sur la définition de permutation proposée par PINEDO, 2002c, reposant sur le principe FIFO, nous avons démontré l'existence d'un lien étroit entre l'imposition de cette règle et la génération de délais. La modélisation MIP que nous avons élaborée a permis de mettre en lumière l'impact respectif des contraintes FIFO et no-delay sur la performance globale du système. En effet, si l'application stricte

de la règle FIFO tend à accroître les délais, l'imposition de la contrainte no-delay peut, quant à elle, conduire à des violations de la discipline FIFO. L'application individuelle de ces contraintes a un impact limité sur le makespan, tandis que leur application conjointe engendre une augmentation considérable de ce dernier. Ces résultats soulignent l'importance de considérer l'ensemble des implications associées aux méthodes imposant FIFO ou no-delay, notamment dans les applications liées aux services, où des violations de FIFO ou la génération de délais peuvent significativement altérer l'expérience client.

Dans cette étude, nous avons examiné l'impact de la permutation dans le cadre de l'ordonnancement d'atelier. Les perspectives de recherche qui en découlent sont multiples :

- Nous avons étudié l'impact et les propriétés de différentes définitions de permutation dans un flow shop présentant des opérations manquantes, avec pour objectif principal la minimisation du makespan. Pour de futures recherches, il serait intéressant d'élargir ce cadre à d'autres critères (tels que la moyenne des dates de fin ou des objectifs basés sur le retard) afin d'évaluer précisément l'influence de chaque définition de permutation.
- Les indicateurs développés ici, notamment IND_4 , se sont révélés particulièrement pertinents pour aider à choisir entre les approches à permutation des approches non-permutation. ce qui suggère la mise en place d'un outil d'aide à la décision. Celui-ci permettrait de sélectionner la stratégie d'ordonnancement la plus adaptée à l'architecture de chaque instance.
- À partir de la définition de PINEDO, 2002c (FIFO), nous avons élargi notre analyse à des problèmes de type job shop. De futures études pourront approfondir les propriétés associées à cette contrainte FIFO et explorer d'autres applications, notamment dans des environnements d'atelier comme le flow shop flexible.
- La majorité des heuristiques et métaheuristiques actuelles travaillent en mode « no-delay », Or, notre étude a montré que, dans un environnement job shop soumis à une contrainte stricte de FIFO, le respect de cette discipline peut nécessiter l'introduction de délais volontaires. Nos résultats soulignent ainsi la nécessité de concevoir des algorithmes intégrant des délais volontaires pour gérer efficacement la contrainte FIFO.

BIBLIOGRAPHIE GÉNÉRALE

- ALA, A., SIMIC, V., PAMUCAR, D., & TIRKOLAEI, E. B. (2022a). Appointment scheduling problem under fairness policy in healthcare services : fuzzy ant lion optimizer. *Expert systems with applications*, 207, 117949.
- ALKHATEEB, F., ABED-ALGUNI, B. H., & AL-ROUSAN, M. H. (2022a). Discrete hybrid cuckoo search and simulated annealing algorithm for solving the job shop scheduling problem. *Journal of Supercomputing*, 78(4), 4799-4826.
- APPLEGATE, D., & COOK, W. (1991c). A computational study of the job-shop scheduling problem. *ORSA Journal on computing*, 3(2), 149-156.
- BAI, D., & REN, T. (2013). New approximation algorithms for flow shop total completion time problem. *Engineering Optimization*, 45(9), 1091-1105.
- BELLMAN, R. (1966). *Dynamic programming. science*, 153(3731), 34-37.
- BENAVIDES, A. J., & RITT, M. (2018a). Fast heuristics for minimizing the makespan in non-permutation flow shops. *Computers & OR*, 100, 230-243.
- BLAZEWICZ, J., LENSTRA, J. K., & KAN, A. R. (1983). Scheduling subject to resource constraints : classification and complexity. *Discrete applied mathematics*, 5(1), 11-24.
- BOWMAN, E. H. (1959a). The schedule-sequencing problem. *Operations research*, 7(5), 621-624.
- BRUM, A., RUIZ, R., & RITT, M. (2022a). Automatic generation of iterated greedy algorithms for the non-permutation flow shop scheduling problem with total completion time minimization. *Computers & Industrial Engineering*, 163, 107843.
- BURDETT, R. L., & KOZAN, E. (2018a). An integrated approach for scheduling health care activities in a hospital. *European Journal of Operational Research*, 264(2), 756-773.
- CHEN, C.-L., NEPPALLI, R. V., & ALJABER, N. (1996). Genetic algorithms applied to the continuous flow shop problem. *Computers & Industrial Engineering*, 30(4), 919-929.
- CHIANG, T.-C., & FU, L.-C. (2007a). Using dispatching rules for job shop scheduling with due date-based objectives. *International Journal of Production Research*, 45(14), 3245-3262.
- CORWIN, B. D., & ESOGBUE, A. O. (1974b). Two machine flow shop scheduling problems with sequence dependent setup times : A dynamic programming approach. *Naval Research Logistics Quarterly*, 21(3), 515-524.
- DANTZIG, G. B., & WOLFE, P. (1960). Decomposition principle for linear programs. *Operations research*, 8(1), 101-111.
- DELLA CROCE, F., NARAYAN, V., & TADEI, R. (1996). The two-machine total completion time flow shop problem. *European Journal of Operational Research*, 90(2), 227-237.

- DELL'AMICO, M., & TRUBIAN, M. (1993a). Applying tabu search to the job-shop scheduling problem. *Annals of Operations Research*, 41(3), 231-252.
- DIAMANT, A., MILNER, J., & QUERESHY, F. (2018a). Dynamic patient scheduling for multi-appointment health care programs. *Production and Operations Management*, 27(1), 58-79.
- DONG, X., HUANG, H., & CHEN, P. (2008). An improved NEH-based heuristic for the permutation flowshop problem. *Computers & Operations Research*, 35(12), 3962-3968.
- DUDEK, R. A., PANWALKAR, S., & SMITH, M. (1992). The lessons of flowshop scheduling research. *Operations Research*, 40(1), 7-13.
- ETILER, O., TOKLU, B., ATAK, M., & WILSON, J. (2004). A genetic algorithm for flow shop scheduling problems. *Journal of the Operational Research Society*, 55(8), 830-835.
- FONTES, D. B., HOMAYOUNI, S. M., & GONÇALVES, J. F. (2023a). A hybrid particle swarm optimization and simulated annealing algorithm for the job shop scheduling problem with transport resources. *European Journal of Operational Research*, 306(3), 1140-1157.
- FRENCH, S. (1982a). *Sequencing and Scheduling : An Introduction to the Mathematics of the Job-Shop*. Ellis Horwood Ltd.
- FU, Y., DING, J., WANG, H., & WANG, J. (2018b). Two-objective stochastic flow-shop scheduling with deteriorating and learning effect in Industry 4.0-based manufacturing system. *Applied Soft Computing*, 68, 847-855.
- GAREY, M. R., JOHNSON, D. S., & SETHI, R. (1976a). The Complexity of Flowshop and Jobshop Scheduling. *Mathematics of Operations Research*, 1(2), 117-129.
- GLASS, C. A., & POTTS, C. N. (1996). A comparison of local search methods for flow shop scheduling. *Annals of Operations Research*, 63, 489-509.
- GMYS, J., MEZMAZ, M., MELAB, N., & TUYTTENS, D. (2020). A computationally efficient Branch-and-Bound algorithm for the permutation flow-shop scheduling problem. *European Journal of Operational Research*, 284(3), 814-833.
- GRABOWSKI, J. (1980). On two-machine scheduling with release and due dates to minimize maximum lateness. *Opsearch*, 17, 133-154.
- HAUPT, R. (1989b). A survey of priority rule-based scheduling. *Operations-Research-Spektrum*, 11(1), 3-16.
- HENNEBERG, M., & NEUFELD, J. (2016a). A constructive algorithm and a simulated annealing approach for solving flowshop problems with missing operations. *International Journal of Production Research*, 54(12), 3534-3550.
- IGNALL, E., & SCHRAGE, L. (1965b). Application of the branch and bound technique to some flow-shop scheduling problems. *Operations research*, 13(3), 400-412.

- JIANG, X., TIAN, Z., LIU, W., SUO, Y., CHEN, K., XU, X., & LI, Z. (2022b). Energy-efficient scheduling of flexible job shops with complex processes : A case study for the aerospace industry complex components in China. *Journal of Industrial Information Integration*, 27, 100293.
- JOHNSON, S. M. (1954a). Optimal two-and three-stage production schedules with setup times included. *Naval research logistics quarterly*, 1(1), 61-68.
- KAZEMIAN, P., SIR, M. Y., VAN OYEN, M. P., LOVELY, J. K., LARSON, D. W., & PASUPATHY, K. S. (2017a). Coordinating clinic and surgery appointments to meet access service levels for elective surgery. *Journal of Biomedical Informatics*, 66, 105-115.
- KRISHNA, K., GANESHAN, K., & RAM, D. J. (1995a). Distributed simulated annealing algorithms for job shop scheduling. *IEEE Transactions on Systems, Man, and Cybernetics*, 25(7), 1102-1109.
- KU, W.-Y., & BECK, J. C. (2016a). Mixed integer programming models for job shop scheduling : A computational analysis. *Computers & Operations Research*, 73, 165-173.
- LADHARI, T., & HAOUARI, M. (2005a). A computational study of the permutation flow shop problem based on a tight lower bound. *Computers & Operations Research*, 32(7), 1831-1847.
- LAHA, D., & CHAKRAVORTY, A. (2011). A new heuristic for minimizing total completion time objective in permutation flow shop scheduling. *The International Journal of Advanced Manufacturing Technology*, 53, 1189-1197.
- LEEFTINK, A., VLIEN, I., & HANS, E. W. (2019a). Stochastic integer programming for multi-disciplinary outpatient clinic planning. *Health Care Management Science*, 22, 53-67.
- LEISTEN, R., & KOLBE, M. (1998a). A note on scheduling jobs with missing operations in permutation flow shops. *International Journal of Production Research*, 36(9), 2627-2630.
- LIANG, B., TURKCAN, A., CEYHAN, M. E., & STUART, K. (2015b). Improvement of chemotherapy patient flow and scheduling in an outpatient oncology clinic. *International Journal of Production Research*, 53(24), 7177-7190.
- LIAO, C., LIAO, L., & TSENG, C. (2006a). A performance evaluation of permutation vs. non-permutation schedules in a flowshop. *International journal of production research*, 44(20), 4297-4309.
- LIN, S.-W., & YING, K.-C. (2009). Applying a hybrid simulated annealing and tabu search approach to non-permutation flowshop scheduling problems. *International Journal of Production Research*, 47(5), 1411-1424.
- LIU, B., WANG, L., & JIN, Y.-H. (2007). An effective PSO-based memetic algorithm for flow shop scheduling. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 37(1), 18-27.
- LIU, W., JIN, Y., & PRICE, M. (2017). A new improved NEH heuristic for permutation flowshop scheduling problems. *International Journal of Production Economics*, 193, 21-30.
- LOMNICKI, Z. (1965). A "branch-and-bound" algorithm for the exact solution of the three-machine scheduling problem. *Journal of the operational research society*, 16(1), 89-100.

- MANNE, A. S. (1960b). On the job-shop scheduling problem. *Operations research*, 8(2), 219-223.
- MARICHELVAM, M., & GEETHA, M. (2016). A hybrid discrete firefly algorithm to solve flow shop scheduling problems to minimise total flow time. *International Journal of Bio-Inspired Computation*, 8(5), 318-325.
- MARYNISSEN, J., & DEMEULEMEESTER, E. (2019b). Literature review on multi-appointment scheduling problems in hospitals. *European Journal of Operational Research*, 272(2), 407-419.
- MOU, J., DUAN, P., GAO, L., LIU, X., & LI, J. (2022). An effective hybrid collaborative algorithm for energy-efficient distributed permutation flow-shop inverse scheduling. *Future Generation Computer Systems*, 128, 521-537.
- MURATA, T., ISHIBUCHI, H., & TANAKA, H. (1996). Genetic algorithms for flowshop scheduling problems. *Computers & Industrial Engineering*, 30(4), 1061-1071.
- NAMIKI, K., KOGA, H., AIDA, S., & HONDA, N. (1979a). AN APPROACH TO THE PRODUCTION LINE OF AUTOMOBILES BY MAN-COMPUTER SYSTEM. In *Case Studies in Automation Related to Humanization of Work* (p. 97-106). Elsevier.
- NARASSIMA, M., VASUDEVAN, S. K., & ANBUUDAYASANKAR, S. (2022a). Development of a priority dispatching rule and novel production scheduling algorithm for a job shop. *International Journal of Computer Aided Engineering and Technology*, 16(4), 417-449.
- NAWAZ, M., ENSCORE JR, E. E., & HAM, I. (1983a). A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. *Omega*, 11(1), 91-95.
- OSMAN, I. H. (1995). Heuristics for the generalised assignment problem : simulated annealing and tabu search approaches. *Operations-Research-Spektrum*, 17, 211-225.
- OUCHENE, R., REBAINE, D., & BAPTISTE, P. (2023b). Ambiguity of the definition of permutation flow shops in the presence of missing operations. *Computers & Industrial Engineering*, 182, 109387.
- PINEDO, M. (2002c). *Scheduling : Theory, Algorithms and Systems*, Springer, New York, p. 15.
- POTTS, C. N., SHMOYS, D. B., & WILLIAMSON, D. P. (1991b). Permutation vs. non-permutation flow shop schedules. *Operations research letters*, 10(5), 281-284.
- PUGAZHENDHI, S., THIAGARAJAN, S., RAJENDRAN, C., & ANANTHARAMAN, N. (2003a). Performance enhancement by using non-permutation schedules in flowline-based manufacturing systems. *Computers & Industrial Engineering*, 44(1), 133-157.
- RAMEZANIAN, R., SAIDI-MEHRABAD, M., & RAHMANI, D. (2011a). Flow shop scheduling problem with missing operations : Genetic algorithm and tabu search. *International Journal of Applied Operational Research*, 1, 21-30.
- REZAEIAHARI, M., & KHASAWNEH, M. T. (2017a). An optimization model for scheduling patients in destination medical centers. *Operations Research for Health Care*, 15, 68-81.

- RITT, M., & ROSSIT, D. A. (2024b). Effective heuristics for permutation and non-permutation flow shop scheduling with missing operations. *Computers & Operations Research*, 170, 106742.
- ROSSIT, D. A., TOHMÉ, F., & FRUTOS, M. (2018a). The non-permutation flow-shop scheduling problem : a literature review. *Omega*, 77, 143-153.
- RUIZ, R., & MAROTO, C. (2005a). A comprehensive review and evaluation of permutation flowshop heuristics. *European journal of operational research*, 165(2), 479-494.
- RUIZ-TORRES, A. J., ABLANEDO-ROSAS, J. H., & OTERO, L. D. (2012b). Scheduling with multiple tasks per job—the case of quality control laboratories in the pharmaceutical industry. *International journal of production research*, 50(3), 691-705.
- SAADANI, N., BAHROUN, Z., & BOURAS, A. (2014a). A Linear Mathematical Model for Patients' Activities Scheduling on Hospital Resources. *Proceedings of the 2014 International Conference on Control, Decision and Information Technologies (CoDIT)*, 74-80.
- SADJADI, S., ARYANEZHAAD, M., & ZIAEE, M. (2008a). The general flowshop scheduling problem : mathematical models. *Journal of Applied Sciences*, 8(17), 3032-3037.
- SALIDO, M. A., ESCAMILLA, J., GIRET, A., & BARBER, F. (2016a). A genetic algorithm for energy-efficiency in job-shop scheduling. *The International Journal of Advanced Manufacturing Technology*, 85, 1303-1314.
- SAYADI, M., RAMEZANIAN, R., & GHAFFARI-NASAB, N. (2010). A discrete firefly meta-heuristic with local search for makespan minimization in permutation flow shop scheduling problems. *International Journal of Industrial Engineering Computations*, 1(1), 1-10.
- SERRANO-RUIZ, J. C., MULA, J., & POLER, R. (2022b). Development of a multidimensional conceptual model for job shop smart manufacturing scheduling from the Industry 4.0 perspective. *Journal of Manufacturing Systems*, 63, 185-202.
- SRIDHAR, J., & RAJENDRAN, C. (1993a). Scheduling in a cellular manufacturing system : a simulated annealing approach. *International Journal of Production Research*, 31(12), 2927-2945.
- STAFFORD, E. F. (1988). On the development of a mixed-integer linear programming model for the flowshop sequencing problem. *Journal of the Operational Research Society*, 39(12), 1163-1174.
- TAILLARD, E. (1990). Some efficient heuristic methods for the flow shop sequencing problem. *European journal of Operational research*, 47(1), 65-74.
- VOGL, P., BRAUNE, R., & DOERNER, K. F. (2019b). Scheduling recurring radiotherapy appointments in an ion beam facility : considering optional activities and time window constraints. *Journal of Scheduling*, 22(2), 137-154.
- WAGNER, H. M. (1959a). An integer linear-programming model for machine scheduling. *Naval research logistics quarterly*, 6(2), 131-140.
- WANG, S., & YU, J. (2010b). An effective heuristic for flexible job-shop scheduling problem with maintenance activities. *Computers & Industrial Engineering*, 59(3), 436-447.

- WOLSEY, L. A. (2020). *Integer programming*. John Wiley & Sons.
- XIAO, L., ZHEN, L., LAPORTE, G., BALDACCI, R., & WANG, C. (2022a). A column generation-based heuristic for a rehabilitation patient scheduling and routing problem. *Computers & Operations Research*, 148, 105970.
- YAGMAHAN, B., & YENISEY, M. M. (2010). A multi-objective ant colony system algorithm for flow shop scheduling problem. *Expert Systems with Applications*, 37(2), 1361-1368.
- YAN, B., BRAGIN, M. A., & LUH, P. B. (2018a). Novel formulation and resolution of job-shop scheduling problems. *IEEE Robotics and Automation Letters*, 3(4), 3387-3393.
- YING, K.-C., & LIAO, C.-J. (2004). An ant colony system for permutation flow-shop sequencing. *Computers & Operations Research*, 31(5), 791-801.
- ZHANG, L., YU, Y., LUO, Y., & ZHANG, S. (2020). Improved cuckoo search algorithm and its application to permutation flow shop scheduling problem. *Journal of Algorithms & Computational Technology*, 14, 1748302620962403.
- ZHOU, S., & YUE, Q. (2019a). Appointment scheduling for multi-stage sequential service systems with stochastic service durations. *Computers & Operations Research*, 112, 104757.

APPENDICE A

COMMUNICATION À CIGI-QUALITA-MOSIM 2023

*Ce chapitre est issu de la communication présentée par R. Ouchene, D. Rebaïne et P. Baptiste à la conférence CIGI-Qualita-MOSIM 2023 (Trois-Rivières, 14–16 juin 2023).*⁵

Ouchene, R., Rebaïne, D., & Baptiste, P. (2023). *Impact des opérations manquantes sur le flow shop avec et sans la contrainte de permutation pour le makespan*. In : CIGI-Qualita-MOSIM 2023, Trois-Rivières, Canada, 14–16 juin 2023. DOI : [10.60662/k0f7-b287](https://doi.org/10.60662/k0f7-b287). PDF : [lien direct](#).

RÉSUMÉ

Les problèmes de flow shop sont généralement résolus en ne considérant que les ordonnancements de permutation. Cependant, il est connu que les ordonnancements sans la contrainte de permutation peuvent produire des résultats meilleurs pour divers critères, mais nécessitent toutefois des efforts de calcul supplémentaires. Il est donc important de répondre à la question sur la pertinence des ordonnancements sans la contrainte de permutation. Bien que ce problème ne se pose pas pour le flow shop avec la contrainte no-wait, étant donné qu'un ordonnancement réalisable dans ce cas est de permutation, la question devient néanmoins intéressante lorsqu'il s'agit de résoudre ce problème en présence d'opérations manquantes. L'objectif de cet article est d'identifier des éléments qui pourraient aider à décider de l'intérêt à considérer les ordonnancements sans la contrainte de permutation pour un problème de flow shop avec et sans la contrainte de no-wait en présence d'opérations manquantes pour le critère du makespan.

5. PDF : [lien direct](#). DOI : [10.60662/k0f7-b287](https://doi.org/10.60662/k0f7-b287).

Mots-clés : Flow shop ; permutation ; opérations manquantes ; makespan ; no-wait.

ABSTRACT (EN)

The flow shop problem is usually studied under the assumption of restricting the solution space to a permutation schedules. However, it is well known that no-permutation schedules may generate to better results for various criteria, but at the expense of additional computational efforts. It is therefore of interest to study the relevance of considering schedules without the permutation constraint. Although this problem does not arise for the flow shop with no-wait constraint, given that a feasible schedule, in this case, is a permutation, the question becomes nevertheless of interest when it comes to solve this problem in the presence of missing operations. In this present paper, we aim to identify elements that help to decide in which circumstances it is judicious to consider no-permutation schedules for flow shop problems with and without the no-wait constraint in the presence of missing operations with respect to the makespan criterion.

Flow shop ; permutation ; missing operations ; makespan ; no-wait.

A.1 INTRODUCTION

Dans un système flow shop (FSP), un ensemble de tâches doit être traité sur un ensemble de machines dans un ordre spécifique. Chaque tâche est composée de m opérations. Chaque opération doit être effectuée sur une seule machine, et chaque machine ne peut traiter qu'une seule tâche à la fois. Le flow shop avec le critère du makespan⁶ est résoluble en temps $O(n \log n)$ pour deux machines JOHNSON,

6. Le temps écoulé depuis le début de la première tâche jusqu'à l'achèvement de la dernière tâche.

1954e. Cependant, lorsque le nombre de machines est supérieur à deux, ce problème devient fortement $\mathcal{NP}$ -difficile **an_2-f3**.

Il existe dans la littérature deux grands types de flow shop : le flow shop de permutation (PFSP) et le flow shop sans la contrainte de permutation (NFSP). Dans un NFSP, la séquence d'exécution des tâches peut changer d'une machine à une autre, ce qui conduit à un élargissement considérable de l'espace des solutions réalisables par rapport au flow shop de permutation, où l'ordre des tâches sur chaque machine est figé. Dans les modèles de type NFSP, un effort de calculs supplémentaire peut être requis pour trouver la solution optimale. En effet, il peut être nécessaire d'explorer un plus grand nombre de combinaisons pour trouver la meilleure séquence d'exécution des tâches. Cependant, cet effort peut améliorer la qualité de la solution optimale. Plusieurs auteurs ont souligné l'intérêt des ordonnancements sans la contrainte de permutation dans des environnements de flow shop réels. PUGAZHENDHI et al., 2003d proposent une heuristique simple pour générer des ordonnancements de NFPS à partir d'un ordonnancement de permutation. Ils ont montré que l'utilisation de cette heuristique permet d'obtenir de meilleures solutions, en utilisant le makespan comme critère principal et le temps d'écoulement total comme critère secondaire. L'étude menée par C.-J. LIAO et al., 2006b a comparé les solutions obtenues pour des ordonnancements avec et sans la contrainte de permutation dans un système flow shop en minimisant deux types de critères : les critères basés sur le temps d'achèvement et ceux basés sur les dates d'échéances. Les auteurs ont mis en évidence que le pourcentage d'amélioration est assez faible pour les critères basés sur le temps d'achèvement, mais significatif pour les critères basés sur les dates d'échéances.

Il est connu que l'existence d'opérations manquantes dans un flow shop (FSMOP) a un impact sur la structure des solutions. L'existence d'opérations manquantes peut se produire dans des situations où certaines tâches n'ont pas besoin d'être traitées sur toutes les machines en raison de leur nature ou de leur configuration. Sous cette hypothèse, SRIDHAR et RAJENDRAN, 1993c proposent un algorithme de recuit simulé (SA) dans le but de minimiser la somme des temps d'exécution pour le flow shop avec des opérations manquantes. LEISTEN et KOLBE, 1998d proposent un assouplissement de la

permutation. Ils développent un algorithme qui contient une procédure d'énumération pour générer des permutations partielles SADJADI et al., 2008d et RAMEZANIAN, SAIDI et RAHMANI, 2011 présentent quelques modèles de programmation linéaires mixtes en nombres entiers pour FSMOP. HENNEBERG et NEUFELD, 2016d ont proposé pour les problèmes FSMOP une version améliorée de l'heuristique de PUGAZHENDHI et al., 2003e avec un algorithme de recuit simulé en deux étapes.

Une autre variante de flow largement rencontrée en pratique est le flow shop avec la contrainte no-wait *i.e.* aucun temps d'attente entre deux opérations successives d'un job n'est permis sur les machines. Cela signifie que chaque étape de traitement doit être immédiatement suivie par l'étape suivante pour chaque tâche. Ce problème a attiré l'attention de nombreux chercheurs en raison de sa pertinence pratique. De nombreuses approches exactes ont été proposées pour résoudre ce problème. Nous pourrions citer l'algorithme de Branch and Bound proposé par MADHUSHINI et RAJENDRAN, 2020. SAMARGHANDI et BEHROOZI, 2017 considèrent ce problème avec la contrainte sur les dates d'achèvement. Ils proposent à cet effet plusieurs modèles mathématiques (mixte en nombres entiers et quadratique en nombres entiers). Le problème du no-wait flow shop étant fortement $\mathcal{NP}$ -difficile, même pour trois machines, de nombreux chercheurs ont proposé des approches heuristiques pour résoudre des instances de moyennes et grandes tailles. Parmi ces approches, nous citons l'approche présentée par NAILWAL et al., 2016 et celle de BERTOLISSI, 2000. D'autres auteurs se sont intéressés aux métaheuristiques. Nous trouvons par exemple les travaux de CHAUDHRY et MAHMOOD, 2012 qui ont proposé un algorithme génétique, l'algorithme de colonie de fourmis proposé par Q.-K. PAN et al., 2008 et l'algorithme de recherche coucou, d'inspiration quantique, proposé par ZHU et al., 2018.

Le problème de flow shop no-wait avec opérations manquantes reste peu étudié. Il est montré NP-difficile au sens fort même pour deux machines SAHNI et CHO, 1979b. GLASS et al., 1999b proposent et analysent des heuristiques pour le problème de flow shop no-wait à deux machines auxquelles certaines tâches ne nécessitent le traitement que sur la première machine. Ils ont proposé deux heuristiques : une heuristique simple qui consiste à ajouter des opérations fictives avec des temps de traitement nuls, et une heuristique à deux phases, appelée GGL, qui consiste à résoudre le sous-problème pour les tâches

sans opérations manquantes et à insérer les tâches restantes dans l’ordonnancement. Ces heuristiques ont un temps de calcul de $O(n \log n)$ et un rapport de performance de 2 dans le pire des cas pour la première et $\frac{4}{3}$ pour la seconde.

Le flow shop est traditionnellement associé à la production manufacturière, mais il peut également modéliser des applications de services telles que celles liées aux hôpitaux et aux cliniques médicales. Dans ces applications, les patients peuvent suivre un ordre fixe de traitement, allant de l’admission à la consultation, examens, traitements, etc. Cependant, étant donné que les patients sont aiguillés vers différents départements en fonction de leur état de santé et de leurs besoins, ils peuvent sauter certaines de ces étapes, ce qui peut poser un problème de flow shop avec des opérations manquantes. Dans une telle situation, la notion de permutation prend une dimension plus psychologique, car lorsque ici les jobs représentent des personnes, il faut tenir compte du sentiment de justice et du désir de ne pas être doublés lorsque ces patients sont transférés d’un service à un autre. Afin de maximiser la satisfaction des clients, tout en optimisant l’utilisation des ressources dans un tel problème, la contrainte de no-wait est imposée et le makespan est minimisé. La minimisation du makespan favorise la gestion des ressources, tandis que la contrainte de no-wait favorise la satisfaction des clients. Cette modélisation permet de trouver un équilibre entre ces deux critères afin de répondre aux besoins des deux parties.

Dans cet article, nous nous intéressons à la pertinence des ordonnancements, sans la contrainte de permutation, pour résoudre un problème de flow shop. Pour ce faire, nous déterminons dans ce qui suit, des indicateurs pour décider de l’avantage à utiliser ces ordonnancements, avec ou sans la contrainte de no-wait, par rapport au critère du makespan.

Dans ce qui suit, nous identifions quelques indicateurs répondant à cette question dans le cas d’opérations manquantes. Ensuite, à la Section 3, nous étendons cette étude d’indicateurs à un contexte sans la contrainte de no-wait. Une conclusion est présentée à la Section 4.

A.2 FLOW SHOP AVEC LA CONTRAINTE DE NO-WAIT

Dans le cas classique d'un flow shop avec la contrainte de no-wait, seuls les ordonnancements de permutation sont pris en compte. Cependant, en présence d'opérations manquantes, cette restriction n'est pas toujours justifiée. Cela est illustré par la Figure A.1, où la solution optimale a été obtenue en planifiant l'opération T_1 avant T_2 sur la première machine, mais cela n'est pas le cas sur la troisième machine. Cet exemple illustre bien un cas d'ordonnement sans la contrainte de permutation.

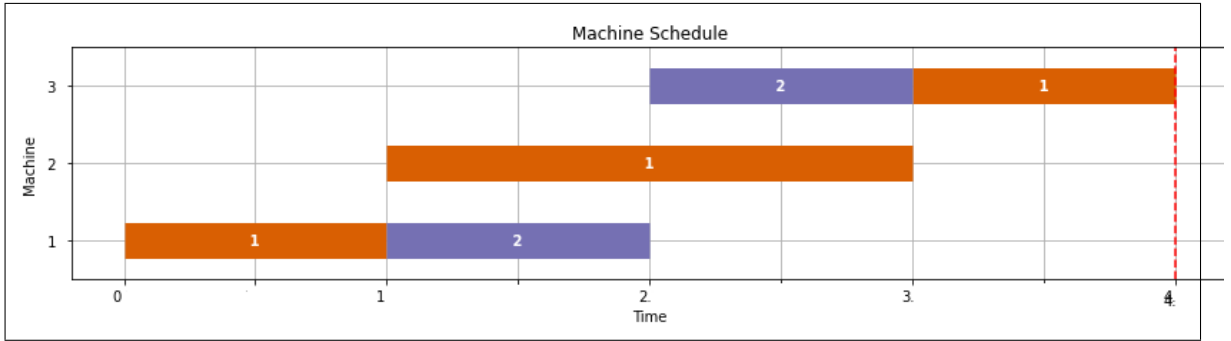


FIGURE A.1 : Exemple d'ordonnement sans la contrainte de permutation pour un flow shop avec la contrainte de no-wait

Dans cette section, nous nous focalisons sur l'analyse de quatre indicateurs sur des instances du problème de flow shop avec la contrainte de no-wait en présence d'opérations manquantes, pour déterminer les situations pour lesquelles il est judicieux d'utiliser les ordonnancements sans la contrainte de permutation, par rapport au critère du makespan.

NOTATIONS

- $T = \{1, \dots, n\}$ désigne l'ensemble des tâches.
- $M = \{M_1, \dots, M_m\}$ désigne l'ensemble des machines.
- Le temps de traitement de la tâche j sur la machine i est noté p_{ij} .
- L'opération de la tâche j sur la machine i est notée O_{ji} .
- E_j est l'ensemble des machines telles que pour $i \in E_j$ on a $p_{ij} \neq 0$.

- $E_{i,j}$ est un ensemble contenant tous les couples de machines (M_h, M_k) tels que $M_h, M_k \in E_i \cap E_j$ et où k est le plus petit entier tel que $M_h, M_k \in E_i \cap E_j$ et $h < k$.

A.2.1 DESCRIPTION DES INDICATEURS

Dans cette section nous identifions les éléments clés pour prédire l'impact des opérations manquantes pour les ordonnancements avec et sans la contrainte de permutation dans un flow shop no-wait. Il est donc important de prendre en compte le nombre total d'opérations manquantes par rapport au nombre total d'opérations. Ce pourcentage peut être analysé à la fois sur l'axe des tâches et des machines. Nous avons alors choisi de considérer les trois indicateurs suivants :

- L'indicateur $IND_1(I)$ s'intéresse au pourcentage global d'opérations manquantes.
- L'indicateur $IND_2(I)$ s'intéresse à la variabilité dans l'ensemble des tâches du nombre d'opérations manquantes. Il mesure cette variabilité en utilisant l'écart type du nombre d'opérations manquantes dans l'ensemble des tâches de l'instance I .
- L'indicateur $IND_3(I)$ s'intéresse à la variabilité dans l'ensemble des machines du nombre d'opérations manquantes. Il mesure cette variabilité en utilisant l'écart type du nombre d'opérations manquantes dans l'ensemble des machines de l'instance I .

Au-delà des indicateurs mentionnés précédemment, nous proposons un indicateur supplémentaire basé sur une condition nécessaire pour les paires de tâches qui peuvent potentiellement s'interchanger dans leur ordre d'exécution sur les machines⁷ dans un ordonnancement de flow shop avec la contrainte no-wait et d'opérations manquantes. Cette condition est exprimée dans la proposition suivante :

7. On dit qu'une paire de tâches (i, j) s'interchange entre-elles si, sur une machine, la tâche i précède la tâche j et, sur une autre, la tâche j précède la tâche i .

Théorème A.2.1. Si pour deux tâches i et j , et deux machines $(M_h, M_k) \in E_{i,j}$, la tâche i précède la tâche j sur la machine M_h et la tâche j précède la tâche i sur la machine M_k , alors

$$p_{h,i} + \sum_{n=h}^k p_{n,j} \leq \sum_{n=h}^{k-1} p_{n,i}.$$

Proof. Soit $T_{h,i}$, $C_{h,i}$ et $T_{h,j}$, $C_{h,j}$ les dates de début et de fin respectivement des tâches i et j dans la machine M_h . Avec la contrainte de no-wait, on a $C_{k,j} = T_{h,j} + \sum_{n=h}^k p_{n,j}$ et $T_{k,i} = T_{h,i} + \sum_{n=h}^{k-1} p_{n,i}$. Nous avons la tâche i précède la tâche j sur la machine M_h ; cela implique $T_{h,i} + p_{h,i} \leq T_{h,j}$. D'autre part, la tâche j précède la tâche i sur la machine M_k . dans ce cas, nous avons

$$C_{k,j} = T_{h,j} + \sum_{n=h}^k p_{n,j} \leq T_{k,i},$$

et

$$T_{h,i} + p_{h,i} + \sum_{n=h}^k p_{n,j} \leq T_{h,i} + \sum_{n=h}^{k-1} p_{n,i}.$$

Par conséquent,

$$p_{h,i} + \sum_{n=h}^k p_{n,j} \leq \sum_{n=h}^{k-1} p_{n,i}.$$

■

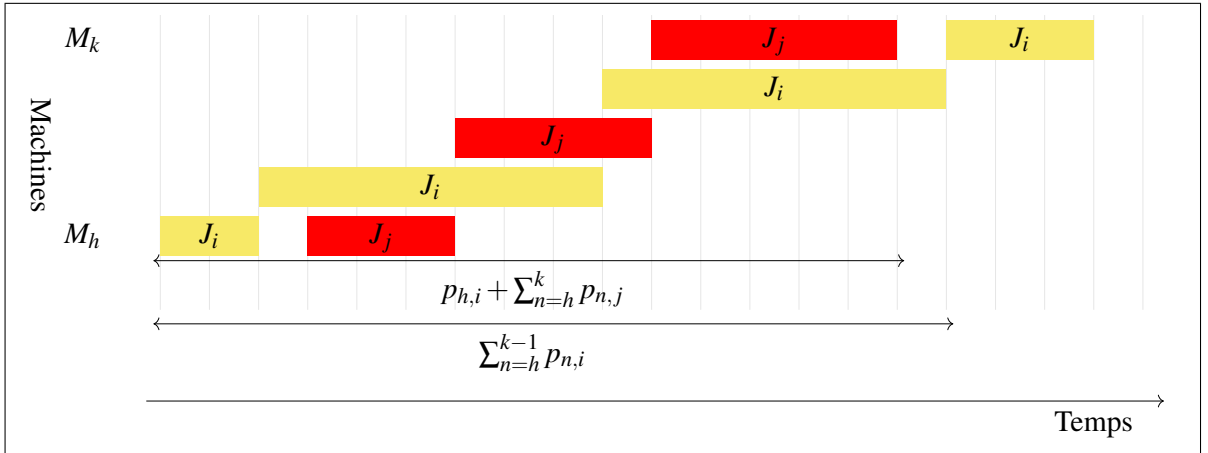


FIGURE A.2 : Représentation de la condition nécessaire sur les paires de tâche.

La Proposition 2.1 exprime une condition nécessaire sur les paires de tâches. Cette condition peut être calculée en utilisant seulement la matrice des temps d'exécution. En parcourant cette matrice, il est possible d'obtenir rapidement pour chaque instance I , le nombre $NB(I)$ qui représente le nombre de couples de tâches qui peuvent potentiellement s'interchanger dans leur ordre d'exécution. À cet effet, nous avons exploité cette information pour décrire notre premier indicateur, noté IND_4 , comme suit :

$$IND_4 = \frac{NB(I)}{A(I)} \times 100,$$

où $A(I)$ représente le nombre total de couples de tâches dans l'instance.

Pour une instance I avec n tâches, $A(I)$ est comme suit :

$$A(I) = \frac{n(n-1)}{2}.$$

A.2.2 TESTS ET DISCUSSIONS

Nous allons maintenant examiner si les 4 indicateurs d'instance peuvent prédire quand il est avantageux d'utiliser des ordonnancements sans la contrainte de permutation pour un problème de flow shop avec la contrainte no-wait en présence d'opérations manquantes. Pour cela, nous avons calculé le pourcentage d'écart de valeurs du makespan obtenus lorsque les tâches sont ordonnancées en respectant la contrainte de permutation et lorsqu'elles ne le sont pas, pour 110 instances. Cet écart est noté Ec . Nous utiliserons ensuite ces résultats pour déterminer si les indicateurs d'instance sont significatifs pour ce pourcentage d'écart. Pour notre étude, nous considérons des instances avec un nombre de machines $m = \{5, 10\}$ et de tâches $n \in \{10, 15, 20, 25, 30\}$ avec un pourcentage d'opérations manquantes de 10, 30, 50 et 70.

Nous avons choisi de diviser nos données d'écart et d'indicateurs IND_2 , IND_3 et IND_4 en 4 catégories équilibrées. Nous avons utilisé une méthode de regroupement en quatre groupes pour étudier le lien entre l'écart et ces indicateurs. Les 4 catégories des indicateurs sont :

- Catégorie 0 : les valeurs de IND_2 , IND_3 et IND_4 sont inférieures ou égales à 13, 9 et 2.5, respectivement.
- Catégorie 1 : les valeurs de IND_2 , IND_3 et IND_4 sont comprises entre 13 et 17, 9 et 12, 2.5 et 5, respectivement.
- Catégorie 2 : les valeurs de IND_2 , IND_3 et IND_4 sont comprises entre 17 et 20, 12 et 17, 5 et 10, respectivement.
- Catégorie 3 : les valeurs de IND_1 sont supérieures à 20, 17 et 10, respectivement.

Ainsi, les catégories 0, 1, 2 et 3 désignent une petite, une moyenne, une grande et une très grande valeur d'indicateurs, respectivement.

Dans cette étude, nous avons utilisé les seuils suivants pour catégoriser les données de l'écart Ec :

- Catégorie 0 : les valeurs de Ec inférieures ou égales à 0.1
- Catégorie 1 : les valeurs de Ec comprises entre 0.1 et 2
- Catégorie 2 : les valeurs de Ec comprises entre 2 et 4
- Catégorie 3 : les valeurs de Ec supérieures à 4.

Ainsi, les catégories 0, 1, 2 et 3 désignent respectivement une petite, une moyenne, une grande et une très grande valeur d'écart.

Pour visualiser les résultats de notre étude, nous avons utilisé des graphiques à barres (voir Figure 3). Le graphique illustre une relation étroite entre l'indicateur IND_4 et l'écart Ec . Lorsque l'un des deux est élevé, l'autre le sera également en général.

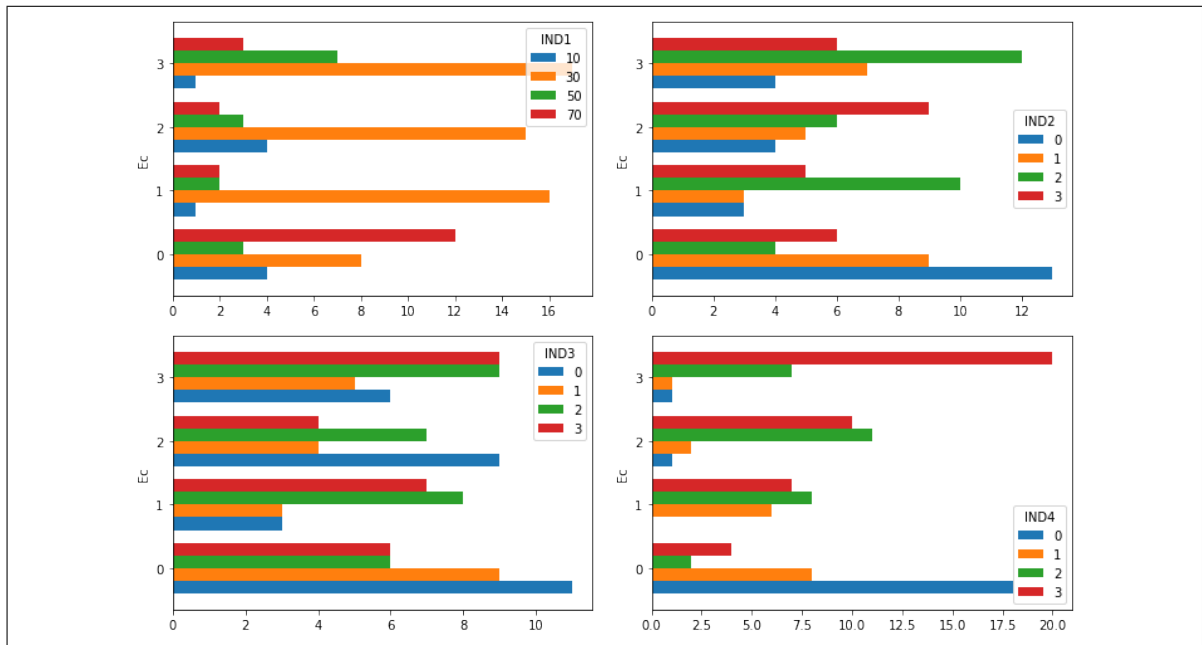


FIGURE A.3 : Graphique à barres pour les catégories de l'indicateur et l'écart.

Après la visualisation des données en utilisant le graphique, une hypothèse pourrait être formulée selon laquelle les indicateurs sont significatifs pour l'écart. Cette hypothèse peut être vérifiée en utilisant des méthodes statistiques comme un test ANOVA (Analyse de la Variance) pour évaluer la significativité de l'effet de l'indicateur sur l'écart ou en utilisant une régression de Spearman pour mesurer la corrélation entre les indicateurs IND_2 , IND_3 et IND_4 avec l'écart Ec . Les résultats de ces tests sont les suivants :

- Le test ANOVA est utilisé pour vérifier si les quatre indicateurs ont un effet significatif sur l'écart Ec en comparant les moyennes de l'écart pour les différentes catégories d'indicateur. Nous avons posé l'hypothèse nulle selon laquelle il n'y a pas de différence significative entre les moyennes de l'écart pour les différentes catégories de d'indicateur et avons utilisé un seuil de signification de 0.005. Les résultats ont montré que la valeur de p^8 était supérieure à notre seuil de signification pour les indicateurs IND_1 , IND_2 et IND_3 avec respectivement $p = 0.13$, $p = 0.23$ et $p = 0.26$. Elle était inférieure au seuil de signification avec ($p = 3.56e^{-06}$) pour IND_4 . Cela signifie que l'indicateur IND_4 a un effet significatif sur l'écart Ec , contrairement aux indicateurs IND_1 , IND_2 et IND_3 .

8. p est la probabilité de l'hypothèse nulle

— Le test de Spearman est utilisé pour vérifier si les indicateurs IND_2 , IND_3 et IND_4 sont corrélés à l'écart Ec . L'hypothèse nulle est qu'il n'y a pas de corrélation entre l'écart et l'indicateur, avec un seuil de signification de 0.005. Les résultats ont montré que la valeur de p était supérieure à notre seuil de signification ($p = 0.14$ pour IND_2 et $p = 0.04$ pour IND_3) et le coefficient de corrélation non linéaire était de 0.19 pour IND_2 et de 0.14 pour IND_3 . Cela signifie qu'il n'y a pas de corrélation significative entre ces indicateurs et l'écart Ec .

Pour IND_4 , les résultats ont montré que la valeur de p était inférieure à notre seuil de signification ($p = 8.93e^{-14}$) et le coefficient de corrélation non linéaire était de 0.64. Cela signifie que l'indicateur IND_4 est corrélée à l'écart Ec .

En résumé, nous avons utilisé des tests statistiques pour évaluer l'effet des quatre indicateurs sur l'écart Ec . Les résultats ont montré l'indicateur IND_4 a un effet significatif sur l'écart Ec (test ANOVA, $p = 3.56e^{-06}$), et qu'ils sont corrélés (test de Spearman, $p = 8.93e^{-14}$), contrairement aux indicateurs IND_1 , IND_2 et IND_3 qui n'ont pas d'effet significatif sur l'écart Ec (test de Spearman, $p = 0.14$ pour IND_2 et $p = 0.04$ pour IND_3 , test ANOVA, $p = 0.13$, $p = 0.23$ et $p = 0.26$ pour IND_1 , IND_2 et IND_3). Ces résultats suggèrent que l'indicateur IND_4 pourrait être efficace pour prédire les variations de l'écart Ec . Ceci pourrait nous aider à déterminer quand il est avantageux d'utiliser des ordonnancements sans la contrainte de permutation pour un problème de flow shop avec la contrainte no-wait en présence d'opérations manquantes.

A.3 FLOW SHOP SANS LA CONTRAINTE DE NO-WAIT

Dans cette section, nous allons utiliser les mêmes instances que précédemment pour étudier l'efficacité de l'indicateur IND_4 lorsque la contrainte de no-wait est relâchée. Nous allons étudier si l'indicateur IND_4 reste efficace pour prédire l'ampleur de l'écart entre les ordonnancements avec et sans la contrainte de permutation dans un problème de flow shop avec des opérations manquantes. Pour cela, nous allons utiliser des tests statistiques similaires à celles utilisées précédemment pour évaluer la

relation entre IND_4 et Ec . Les résultats de cette étude pourront nous indiquer si l'indicateur IND_4 est toujours pertinent pour prédire l'écart entre les deux types d'ordonnancements dans un contexte plus général.

Les résultats de ces tests sont les suivants :

- Le test ANOVA est utilisé pour vérifier si l'indicateur IND_4 a un effet significatif sur l'écart Ec , en comparant les moyennes de l'écart pour les différentes catégories de l'indicateur. Nous avons posé l'hypothèse nulle selon laquelle il n'y a pas de différence significative entre les moyennes de l'écart pour les différentes catégories de l'indicateur IND_4 et avons utilisé un seuil de signification de 0.005. Les résultats ont montré que la valeur de p était inférieure à notre seuil de signification ($p = 0.00014$). Cela signifie que l'indicateur IND_4 a un effet significatif sur l'écart Ec .
- Le test de Spearman est utilisé pour vérifier si l'indicateur IND_4 et l'écart Ec sont corrélés. L'hypothèse nulle est qu'il n'y a pas de corrélation entre ces deux variables, avec un seuil de signification de 0.005. Les résultats ont montré que la valeur de p était inférieure à notre seuil de signification ($p = 0.00024$) et le coefficient de corrélation non linéaire était de 0.34. Cela signifie que la corrélation entre l'indicateur IND_4 et l'écart Ec est faible.

Les résultats des tests statistiques montrent que l'indicateur IND_4 a un effet significatif sur l'écart Ec lorsque la contrainte de no-wait est relâchée. Cependant, la corrélation entre l'indicateur IND_4 et Ec est faible. Cela signifie qu'il pourrait y avoir d'autres facteurs qui influencent l'écart en plus de IND_4 . Afin de poursuivre notre analyse, nous allons maintenant diviser nos données en deux ensembles distincts en fonction du nombre de machines. Le premier ensemble regroupe les instances avec un nombre de machines égal à 5, tandis que le deuxième regroupe les instances avec un nombre de machines égal à 10. Nous utiliserons les mêmes tests statistiques que précédemment pour comparer les résultats obtenus et identifier les similarités et différences entre les deux ensembles. Les résultats de ces tests sont les suivants :

- Premier ensemble : L'indicateur IND_4 n'a pas un effet significatif sur l'écart Ec (test ANOVA, $p = 0.80$), et ils ne sont pas corrélés (test de Spearman, $p = 0.28$ et taux de corrélation égale à 0.13).
- Deuxième ensemble : L'indicateur IND_4 a un effet significatif sur l'écart Ec (test ANOVA, $p = 6.55e^{-05}$), et ils sont corrélés (test de Spearman, $p = 1.20e^{-08}$ et taux de corrélation égale à 0.76).

Les tests statistiques montrent que les résultats obtenus pour les deux ensembles sont différents. Ces résultats indiquent que l'indicateur IND_4 pourrait être plus efficace pour des instances avec un plus grand nombre de machines.

A.4 CONCLUSION

Cette étude a permis d'identifier quatre indicateurs pour décider quand il est avantageux d'utiliser des ordonnancements avec ou sans la contrainte de permutation pour un problème de flow shop avec et sans la contrainte de no-wait en présence d'opérations manquantes pour le critère du makespan.

Les trois premiers indicateurs s'intéressent au pourcentage et la variabilité des opérations manquantes, tandis que le quatrième indicateur se concentre sur les paires de tâches qui peuvent potentiellement s'interchanger dans leur ordre d'exécution sur les machines.

Les résultats des tests statistiques ont montré que le quatrième indicateur est un outil prometteur pour aider à décider quand utiliser des ordonnancements sans la contrainte de permutation pour des problèmes de flow shop avec opérations manquantes. Ces résultats ouvrent des perspectives pour des recherches futures afin d'étudier la performance de cet indicateur pour d'autres critères réguliers et aussi explorer d'autres indicateurs pertinents pour ce problème.

BIBLIOGRAPHY

- BERTOLISSI, E. (2000). Heuristic algorithm for scheduling in the no-wait flow-shop. *Journal of Materials Processing Technology*, 107, 459-465.
- CHAUDHRY, I. A., & MAHMOOD, S. (2012). No-wait flowshop scheduling using genetic algorithm. *Proceedings of the World Congress on Engineering*, 3, 4-6.
- GLASS, C. A., GUPTA, J. N., & POTTS, C. N. (1999b). Two-machine no-wait flow shop scheduling with missing operations. *Mathematics of Operations Research*, 24(4), 911-924.
- HENNEBERG, M., & NEUFELD, J. (2016d). A constructive algorithm and a simulated annealing approach for solving flowshop problems with missing operations. *International Journal of Production Research*, 54(12), 3534-3550.
- JOHNSON, S. M. (1954e). Optimal two-and three-stage production schedules with setup times included. *Naval research logistics quarterly*, 1(1), 61-68.
- LEISTEN, R., & KOLBE, M. (1998d). A note on scheduling jobs with missing operations in permutation flow shops. *International journal of production research*, 36(9), 2627-2630.
- LIAO, C.-J., LIAO, L., & TSENG, C. (2006b). A performance evaluation of permutation vs. non-permutation schedules in a flowshop. *International Journal of Production Research*, 44, 4297-4309.
- MADHUSHINI, N., & RAJENDRAN, C. (2020). Branch-and-bound algorithms for scheduling in an m-machine no-wait flowshop. *Sādhanā*, 45.
- NAILWAL, K., GUPTA, D., & JEET, K. (2016). Heuristics for no-wait flow shop scheduling problem. *International Journal of Industrial Engineering Computations*, 7, 671-680.
- PAN, Q.-K., FATIH TASGETIREN, M., & LIANG, Y.-C. (2008). A discrete particle swarm optimization algorithm for the no-wait flowshop scheduling problem [Part Special Issue : Bio-inspired Methods in Combinatorial Optimization]. *Computers & Operations Research*, 35(9), 2807-2839.
- PUGAZHENDHI, S., THIAGARAJAN, S., RAJENDRAN, C., & ANANTHARAMAN, N. (2003d). Performance enhancement by using non-permutation schedules in flowline-based manufacturing systems. *Computers & Industrial Engineering*, 44(1), 133-157.
- PUGAZHENDHI, S., THIAGARAJAN, S., RAJENDRAN, C., & ANANTHARAMAN, N. (2003e). Performance enhancement by using non-permutation schedules in flowline-based manufacturing systems. *Computers & Industrial Engineering*, 44(1), 133-157.
- RAMEZANIAN, R., SAIDI, M. M., & RAHMANI, D. (2011). Flow shop scheduling problem with missing operations : Genetic algorithm and tabu search.
- SADJADI, S., ARYANEZHAAD, M., & ZIAEE, M. (2008d). The general flowshop scheduling problem : mathematical models. *J. of app. sci*, 8(17), 3032-3037.

- SAHNI, S., & CHO, Y. (1979b). Complexity of scheduling shops with no wait in process. *Mathematics of Operations Research*, 4(4), 448-457.
- SAMARGHANDI, H., & BEHROOZI, M. (2017). On the exact solution of the no-wait flow shop problem with due date constraints. *Comput. Oper. Res.*, 81, 141-159.
- SRIDHAR, J., & RAJENDRAN, C. (1993c). Scheduling in a cellular manufacturing system : a simulated annealing approach. *International Journal of Production Research*, 31(12), 2927-2945.
- ZHU, H., QI, X., CHEN, F., HE, X., CHEN, L., & ZHANG, Z. (2018). Quantum-inspired cuckoo co-search algorithm for no-wait flow shop scheduling. *Applied Intelligence*, 49, 791-803.

APPENDICE B

COMMUNICATION À ROADEF 2024 (AMIENS)

Ce chapitre est issu de la communication présentée par P. Baptiste, R. Ouchène et D. Rebaïne à la conférence ROADEF 2024 (Amiens, 4–7 mars 2024).⁹

Baptiste, P., Ouchène, R., & Rebaïne, D. (2024). *Job-shop de permutation ?* Communication orale, ROADEF 2024, Amiens, France, 4–7 mars 2024. [[abstract](#) / [PDF](#)].

Mots-clés : job-shop, permutation, makespan.

B.1 INTRODUCTION

Dans l’univers de l’ordonnancement, le terme « permutation » est généralement associé au *flow-shop*, défini classiquement comme « le même ordre de passage sur toutes les machines ». Pinedo PINEDO, 2002a propose la définition « FIFO sur chaque machine », équivalente pour le *flow-shop* simple. Réduire l’espace de recherche à un seul ordre simplifie les calculs~; la règle FIFO (First In First Out) est la plus facile à implanter sur le terrain et, dans les services, la plus acceptable (« qui aime se faire doubler? »). Cette définition s’applique à une grande variété de problèmes — *flow-shop* hybride ou flexible, *flow-shop* à opérations manquantes, etc. — et peut même s’étendre au delà du *flow-shop*, notamment au *job-shop*.

B.2 COMPARAISON DES MODÈLES $J \parallel C_{\max}$ ET $J \mid \text{FIFO} \mid C_{\max}$ (JOB-SHOP À PERMUTATION)

La contrainte FIFO a été étudiée comme règle de priorité dans l’ordonnancement temps réel HAUPT, 1989a, mais elle semble absente de la littérature en optimisation mathématique. Nous

9. Abstract/PDF : roadef2024.sciencesconf.org/510065/document. Programme : roa-def2024.sciencesconf.org/program.

cherchons à évaluer la détérioration de $C_{\max}$ induite par l'imposition de FIFO sur toutes les machines dans un job-shop. La méthodologie consiste à prendre un modèle mathématique pour $J||C_{\max}$, y intégrer FIFO, puis comparer les $C_{\max}$ obtenus sur des jeux d'essai de référence et sur des instances aléatoires.

B.2.1 MODÈLE MATHÉMATIQUE

Les opérations sont notées $O_{i,j}$, indexées par le job i et la machine j , avec des durées $p_{i,j}$. On introduit :

- une variable de rang $R_{i,j}$ pour chaque $O_{i,j}$;
- des contraintes d'antériorité entre opérations d'un même job ;
- des contraintes de non-chevauchement pour deux jobs sur une même machine ;
- des contraintes de positivité.

Nous ajoutons une variable $PR_{i,j}$ indiquant la machine exécutant l'opération précédant $O_{i,j}$. Cette variable permet : (i) d'exprimer la date d'arrivée d'une opération dans la file (fin de l'opération précédente) ; (ii) d'interdire qu'une opération démarre avant une autre arrivée plus tôt, ajoutant ainsi N^2M contraintes. Pour la première opération de chaque job, on fixe $PR_{i,j} = -1$ afin d'ignorer FIFO.

B.2.2 RÉSULTATS SUR LES BENCHMARKS

Les modèles $J||C_{\max}$ et $J|FIFO|C_{\max}$ ont été testés sur 40 benchmarks classiques (6×6 , 10×5 , 10×10 , 15×10 , 15×15) provenant de la littérature APPLEGATE et COOK, 1991a; FISHER et THOMPSON, 1963b; LAWRENCE, 1984a; E. D. TAILLARD, 1993b. L'écart moyen des solutions optimales est de 1,433%, avec un maximum de 3,52% (cas 10 jobs, 5 machines).

B.2.3 RÉSULTATS SUR DES INSTANCES ALÉATOIRES

Nous avons généré 100 instances 10×10 avec des durées uniformes ($1 \sim 100$) et un ordre des machines aléatoire. L'écart entre les scénarios « avec FIFO » et « sans FIFO » est faible (moyenne 1,16 ; écart-type 0,96 ; max 3,47 ; min 0), corroborant les benchmarks.

B.3 CONCLUSION

Le très faible écart observé suggère que la contrainte FIFO peut être imposée sans perte significative partout où elle simplifie la mise en œuvre ou améliore l'acceptabilité des solutions. Le modèle présenté ne suppose pas que tous les jobs soient disponibles à $t = 0$; il prévoit une *date de libération* correspondant à l'entrée du job dans la file pour sa première opération, ce qui le rend pertinent pour un système de prise de rendez-vous. Contrairement aux métaheuristiques, ce modèle autorise des temps d'attente : une machine peut différer le traitement du premier job de sa file pour qu'il « perde » une priorité sur une machine ultérieure. Pour des problèmes de grande taille, des métaheuristiques dédiées pourraient être explorées, l'espace de recherche demeurant toutefois plus complexe qu'en flow-shop.

BIBLIOGRAPHY

- APPLEGATE, D., & COOK, W. (1991a). A computational study of job-shop scheduling. *ORSA Journal of Computing*, 3, 149-156.
- FISHER, H., & THOMPSON, G. L. (1963b). Probabilistic learning combinations of local job-shop scheduling rules. In J. MUTH & G. THOMPSON (Éd.), *Industrial Scheduling* (p. 225-251). Prentice Hall.
- HAUPT, R. (1989a). A survey of priority rule-based scheduling. *OR Spektrum*, 11, 3-16.
- LAWRENCE, S. (1984a). *Resource Constrained Project Scheduling : An Experimental Investigation of Heuristic Scheduling Techniques* (rapp. tech.) (Technical Report). Carnegie-Mellon University.
- PINEDO, M. (2002a). *Scheduling : Theory, Algorithms and Systems*. Springer.
- TAILLARD, E. D. (1993b). Benchmarks for basic scheduling problems. *European Journal of Operational Research*, 64, 278-285.

APPENDICE C

COMMUNICATION À ROADEF 2025 (CHAMPS-SUR-MARNE)

*Ce chapitre est issu de la communication présentée par P. Baptiste, R. Ouchène et D. Rebaïne au congrès ROADEF 2025 (Champs-sur-Marne, 26–28 février 2025).*¹⁰

Baptiste, P., Ouchène, R., & Rebaïne, D. (2025). *Flow shop de permutation vs. flow shop de non-permutation*. Communication orale, *ROADEF 2025*, Champs-sur-Marne, France, 26–28 février 2025. [[abstract](#) / [PDF](#)].

C.1 INTRODUCTION

Dans la littérature, sur les problèmes de flow shop, l’accent est principalement mis sur les ordonnancements de permutation FRAMINAN et al., [2004](#), car ils permettent de réduire la complexité de calculs à travers le maintien d’un même ordre d’exécution des tâches sur toutes les machines. Cependant, les ordonnancements de non-permutation, bien que nécessitant un effort supplémentaire de calculs, peuvent améliorer considérablement la qualité des solutions générées C.-J. LIAO et al., [2006c](#). Dans cette étude, nous introduisons un indicateur, **IND**, visant à identifier les situations où l’approche de non-permutation serait plus avantageuse. Cet indicateur permet d’évaluer, à partir des caractéristiques structurelles d’une instance de flow shop avec opérations manquantes, si l’effort supplémentaire de calculs pour les ordonnancements de non-permutation est justifié par une nette amélioration des résultats par rapport au critère du makespan ou celui de la somme des dates de fin.

10. Abstract/PDF : [ROADEF2025_resumes.pdf](#). Site : [roadef2025.org](#).

C.2 ÉTUDE EXPÉRIMENTALE

Nous exposons dans cette section les principaux éléments des simulations que nous avons menées.

C.2.1 DESCRIPTION DE L'INDICATEUR

L'indicateur **IND**, pour une instance donnée, calcule le nombre de paires de tâches (i_1, i_2) sur l'ensemble de machines entre h et k satisfaisant la condition suivante :

$$\sum_{j=h}^k p_{i_2,j} \leq \sum_{j=h+1}^k p_{i_1,j} + \min(p_{i_2,h}, p_{i_2,k}),$$

où $p_{i,j}$ est le temps de traitement de la tâche i sur la machine j . Cet indicateur mesure la proportion de paires de tâches où la contrainte de permutation n'est pas dominante, dans un flow shop avec opérations manquantes, restreint à deux tâches (i_1, i_2) et à l'ensemble de machines entre h et k inclusivement.

C.2.2 MÉTHODOLOGIE

Le solveur IBM Cplex 20.0.1 a été utilisé pour résoudre les modèles du problème d'ordonnement de permutation et de non-permutation dans un flow shop avec opérations manquantes. Un total de 640 instances a été généré, comprenant des configurations variées avec des nombres de jobs (10 et 15), de machines (5 et 10) et des pourcentages d'opérations manquantes (10%, 30%, 50% et 70%). Pour chaque instance, l'indicateur **IND** a été calculé afin d'évaluer son impact sur deux critères de performance : l'écart entre le mode permutation et celui de non-permutation, dans un premier temps, par rapport au critère du makespan (E_1) et, dans un second temps, à celui de la somme des dates de fin (E_2).

C.2.3 RÉSULTATS

L'analyse statistique, que nous avons réalisée, examine la relation entre l'indicateur **IND** et les écarts E_1 et E_2 . Les nuages de points dans les figures ci-dessous illustrent ces relations.

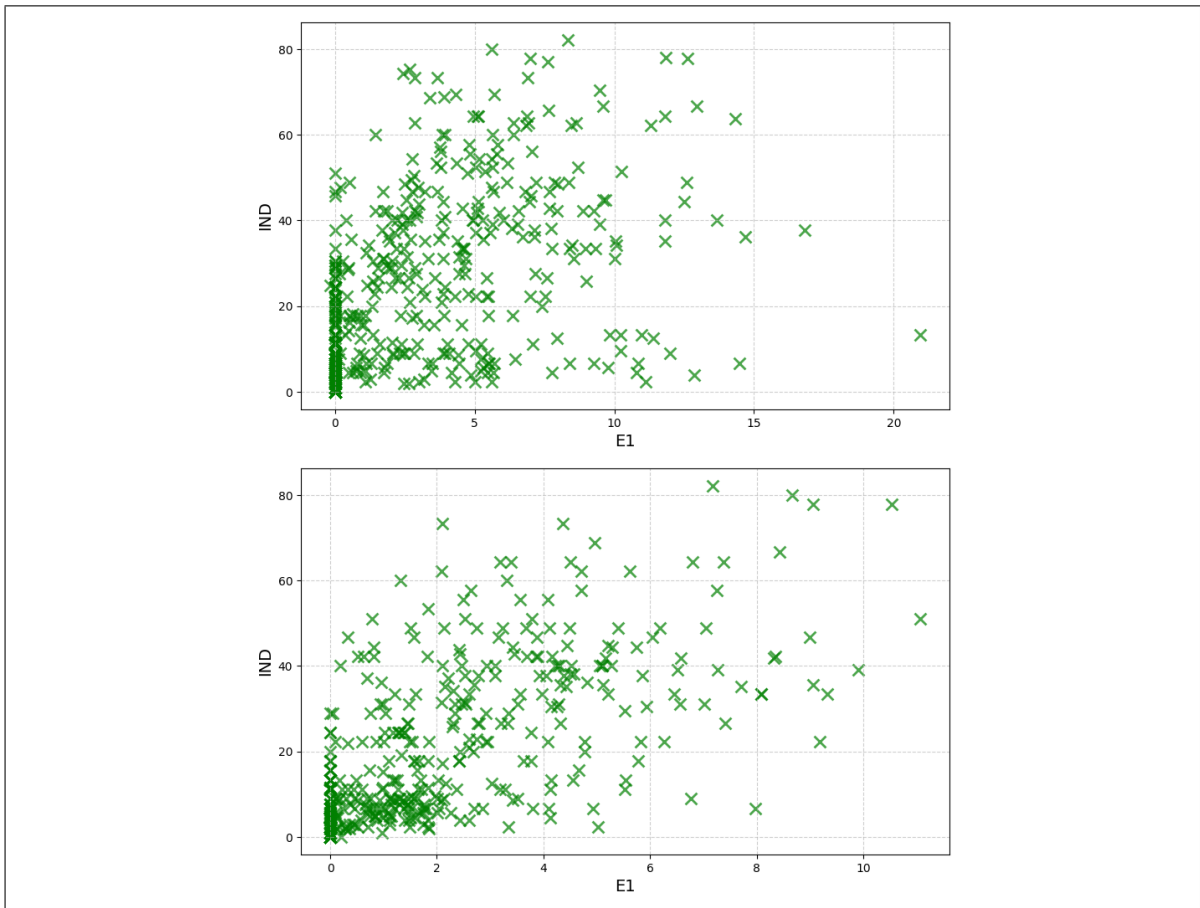


FIGURE C.1 : Relation entre IND et E_1 (gauche) et E_2 (droite)

Un test de corrélation de Spearman révèle une corrélation forte pour E_1 ($r = 0,63$) et très forte pour E_2 ($r = 0,74$). Une analyse de variance (ANOVA), basée sur une classification de **IND** en quartiles, montre des écarts significatifs : $F = 66,61$, $p = 8,44 \times 10^{-37}$ pour E_1 et $F = 124,6$, $p = 2,18 \times 10^{-59}$ pour E_2 . Les boxplots (boîtes à moustaches) ci-dessous illustrent les différences des quartiles de **IND** par rapport à E_1 et E_2 .

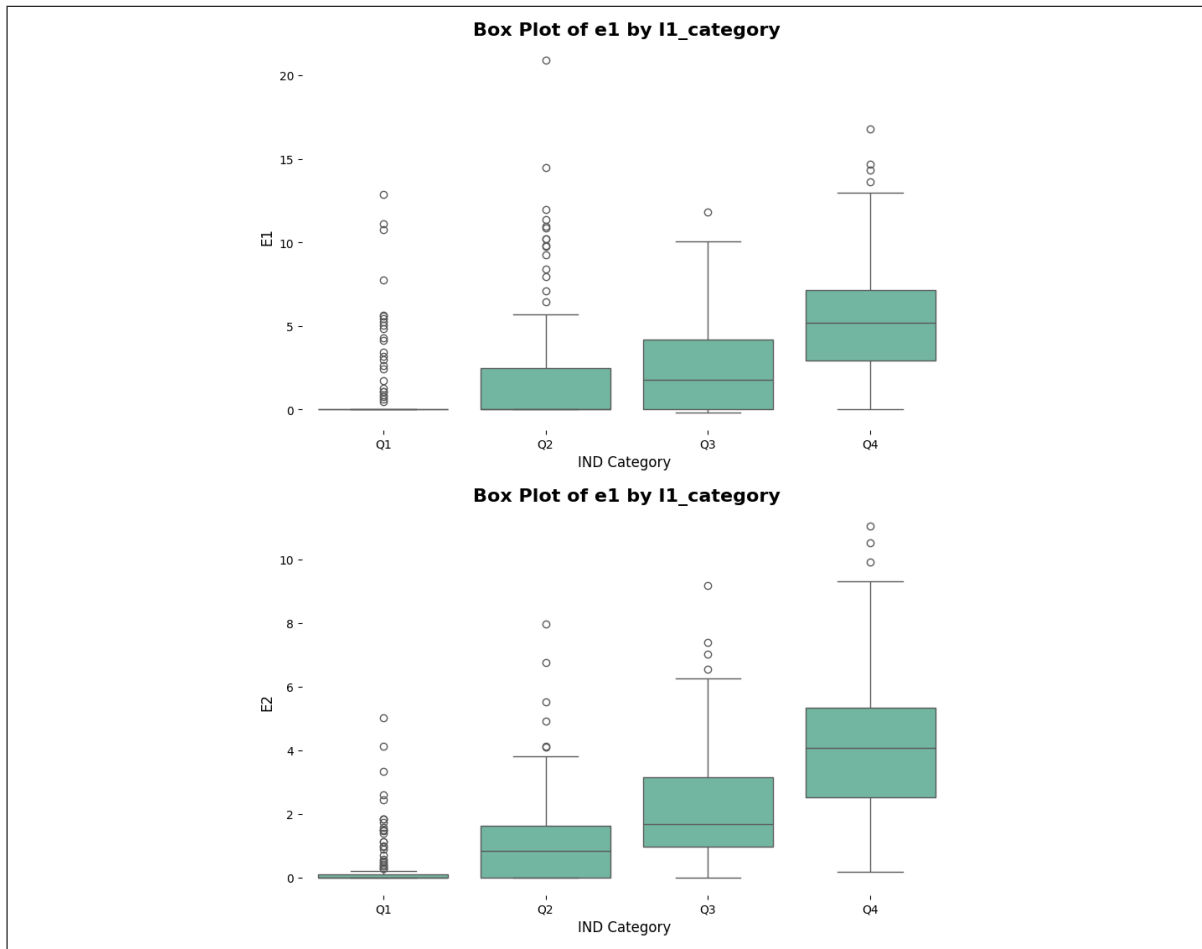


FIGURE C.2 : Boxplots de E_1 (gauche) et E_2 (droite) selon les catégories d'IND

C.3 CONCLUSION

L'indicateur **IND** s'avère intéressant pour prédire l'écart entre les ordonnancements de permutation et de non-permutation par rapport au critère du makespan et à celui de la somme des dates de fin. Pour des valeurs "Élevées" ou "Très Élevées" de **IND**, les ordonnancements de non-permutation sont recommandés, le surcoût de calculs étant compensé par un écart significatif sur la qualité des solutions générées. À l'inverse, des valeurs "Faibles" suggèrent que les permutations suffisent en raison d'un écart minime.

BIBLIOGRAPHY

- FRAMINAN, J. M., GUPTA, J. N., & LEISTEN, R. (2004). A review and classification of heuristics for permutation flow-shop scheduling with makespan objective. *Journal of the Operational Research Society*, 55(12), 1243-1255.
- LIAO, C.-J., LIAO, L., & TSENG, C. (2006c). A performance evaluation of permutation vs. non-permutation schedules in a flowshop. *International Journal of Production Research*, 44, 4297-4309.

APPENDICE D
COMMUNICATION À CODIT 2025 (SPLIT)

Ce chapitre est issu de la communication présentée par P. Baptiste, R. Ouchène et D. Rebaïne à la conférence CoDIT 2025 (Split, 15–18 juillet 2025).¹¹

Ouchene, R., Rebaïne, D. , & Baptiste, P. (2025). *Permutation in Shop Scheduling Problems with FIFO Considerations*. Communication orale, *11th IEEE/IFAC Conference on Control, Decision and Information Technologies (CoDIT 2025)*, Split (Croatie), 15–18 juillet 2025. DOI : *en attente*.

11. Programme officiel : [CoDIT2025-Program.pdf](#).

ABSTRACT

Traditionally, permutation scheduling in Flow Shop problems have been used as maintaining the same task order across all the machines. However, this definition becomes ambiguous when operations are missing OUCHENE et al., 2023a. Alternative definitions, such as the FIFO-based definition introduced by Pinedo PINEDO, 2002b, are more suitable in those contexts. The present study first analyzes Pinedo's definition in Flow Shop problems with missing operations, then extends this approach to Job Shop scheduling, examining its implications with respect to the quality of the solutions.

D.1 INTRODUCTION

Shop scheduling problems occupy a pivotal role in both industrial CORWIN et ESOGBUE, 1974a; DUDEK, PANWALKAR et SMITH, 1992; Y. FU et al., 2018a; JIANG et al., 2022a; NAMIKI et al., 1979b; RUIZ-TORRES et al., 2012a; SERRANO-RUIZ et al., 2022a; WANG et YU, 2010a and service sectors CONFORTI et al., 2011a; LIANG et al., 2015a; SAADANI et al., 2014c; VOGL et al., 2019a. These problems involve scenarios where tasks, composed of multiple operations, must be executed across various machines. In a Flow Shop, each job follows the same predetermined machine sequence. Conversely, in a Job Shop environment, the routing is job-specific : each job may follow a different sequence of machines depending on its individual requirements.

In both industrial and service settings, it is common for certain jobs to skip some machines, leading to the presence of missing operations. For instance, in hospital services, one patient might undergo multiple diagnostic tests, while another diagnosed with a different condition might proceed directly to a specialized service. Similarly, in appointment scheduling systems VOGL et al., 2019a, patients follow personalized paths based on their medical needs, resulting in heterogeneous routing and partially defined job structures.

There are two well-known ways to interpret missing operations. The first, proposed by Potts et al. POTTS et al., 1991a, treats missing operations as tasks with negligible but non-zero duration, requiring jobs to be assigned to every machine. The second interpretation, which we adopt in this work and which aligns with HENNEBERG et NEUFELD, 2016; OUCHENE et al., 2025; RITT et ROSSIT,

[2024a](#); SRIDHAR et RAJENDRAN, [1993d](#), considers a missing operation to be entirely absent from the job's routing.

In classical Flow Shops with missing operations, permutation schedules are traditionally defined as those that preserve a global ordering of jobs across all machines. However, alternative definitions have been proposed. One approach is based on the "no-overtaking" principle : if job *A* precedes job *B* on one machine, it must also precede it on all other machines they both visit. Another interpretation, introduced by Pinedo PINEDO, [2002b](#), defines permutation scheduling according to a FIFO (First-In, First-Out) discipline : a job that finishes earlier on one machine should be scheduled earlier on the next.

In service environments such as healthcare and public administration, FIFO rules play a critical role in managing resources fairly and transparently. By preserving the order of arrival, they help ensure equity among users and contribute to higher levels of satisfaction MARYNISSEN et DEMEULEMEESTER, [2019a](#); VERMEULEN et al., [2007](#). Moreover, minimizing waiting time is a crucial objective in these settings, as it directly affects user experience and operational efficiency (AGNIHOTRI et al., [2024a](#)). However, when jobs follow personalized and incomplete routing paths, as is often the case in service systems, enforcing FIFO discipline across shared machines presents significant modeling and scheduling challenges.

This paper addresses two related but distinct objectives :

- to analyze the theoretical properties of FIFO-based permutation schedules in Flow Shops with missing operations, and
- to extend this FIFO-based permutation concept to Job Shop environments, through mathematical modeling and experimental analysis.

We first explore, in Section D.2, the applicability and implications of Pinedo's FIFO definition in Flow Shops with missing operations. Although limited to a specific machine structure, this analysis offers valuable insights into the behavior of partially routed jobs under FIFO constraints.

In Section D.3, we generalize this perspective to the Job Shop setting, where routing flexibility introduces new modeling complexities. We propose a MIP formulation that explicitly incorporates FIFO constraints between machines. This model allows us to evaluate the impact of FIFO enforcement on performance indicators such as makespan and waiting time.

Finally, Section D.4 summarizes the findings and outlines potential directions for future research.

D.2 PINEDO PERMUTATION FOR FLOW SHOP WITH MISSING OPERATIONS

Permutation flow shop scheduling is traditionally defined by imposing the same task sequence across all machines. However, as highlighted by OUCHENE et al., 2023a, this classical definition becomes ambiguous when considering missing operations. Consequently, two variants arise :

- A **strong permutation** imposes a global ordering of tasks across all the machines, irrespective of missing operations (illustrated in Figure 1, where the global order is 2, 1, 3).

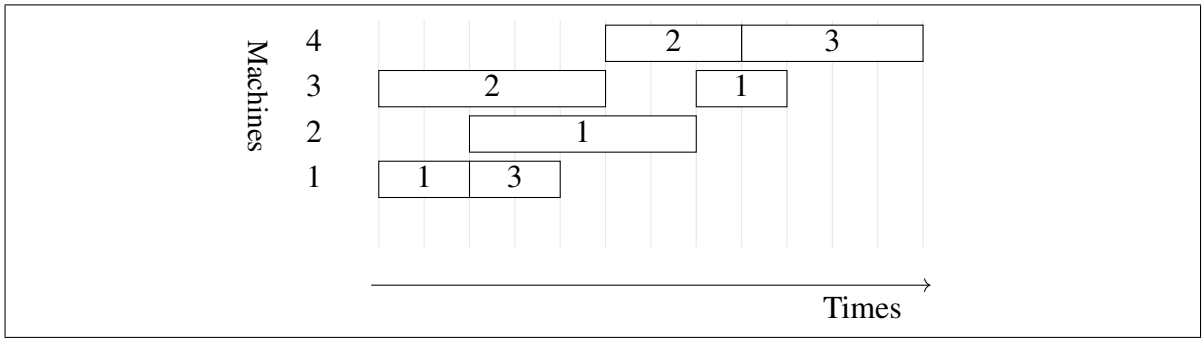


FIGURE D.1 : Illustration of strong permutation scheduling

- A **weak permutation** is defined by a non-overtaking rule : if task i precedes task j on one machine, it must also precede task j on every subsequent machine. Figure 2 illustrates a weak permutation scenario that does not satisfy the strong permutation definition (in this instance, job 1 precedes job 3, job 2 precedes job 1, and job 3 precedes job 2).

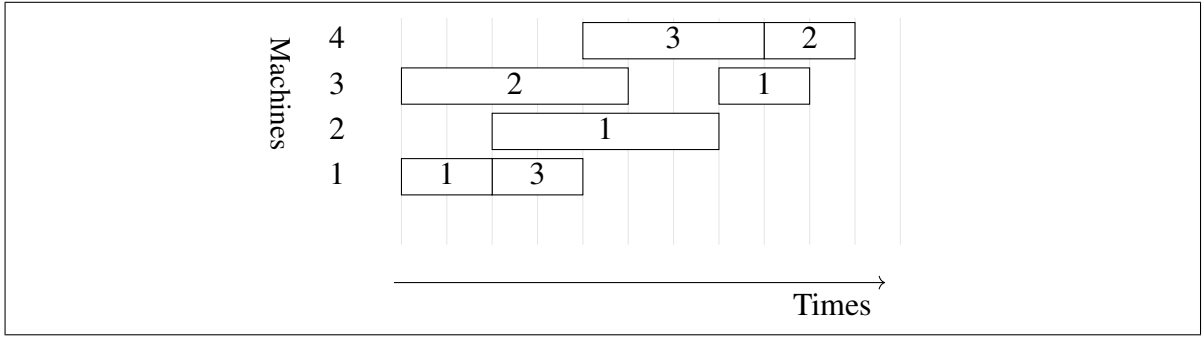


FIGURE D.2 : Illustration of weak permutation scheduling

Moreover, Pinedo (see PINEDO, [2002b](#)) introduces a permutation definition based on a FIFO (First In, First Out) discipline for managing machine queues. This FIFO-based definition coincides with the classical permutation definition in flow shops without missing operations, but it may diverge when certain jobs skip some machines.

In this work, the FIFO rule ensures that if a job finishes its operation earlier on a preceding machine, it must also be executed earlier on the next common machine. This ordering constraint can be formalized as follows. Let J_1 and J_2 be two jobs, and k_1 , k_2 , and h be machines such that :

- J_1 is processed on machine k_1 , then on machine h ,
- J_2 is processed on machine k_2 , then on machine h ,
- h is the next immediate machine for both jobs in their respective routes.

The FIFO rule states that if job J_1 completes on machine k_1 before job J_2 completes on machine k_2 , then job J_1 must be completed before job J_2 on machine h .

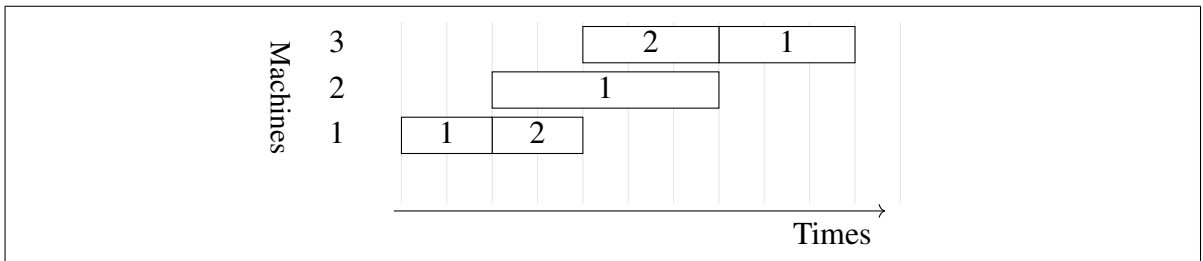


FIGURE D.3 : Pinedo's permutation scheduling for a Flow Shop with missing operations, illustrating a scenario that is neither a weak nor a strong permutation.

Figure 3 illustrates an optimal solution for an instance involving three machines and two jobs, where the objective is to minimize the makespan. Notably, this solution adheres neither to a weak nor a strong permutation, indicating that, under either the weak or strong permutation definitions, permutation scheduling does not dominate¹², even in a three-machine scenario. In the following discussion, we will demonstrate that when considering Pinedo’s definition, permutation becomes dominant for three machines. This highlights the divergence among the various definitions of permutation.

In the following, we illustrate several properties of permutation flow shop scheduling as defined by Pinedo, specifically within contexts where certain operations may be missing.

Théorème D.2.1. *In a permutation flow shop problem with missing operations, scheduling tasks according to a FIFO discipline on the last machine M_m is dominant under the makespan criterion.*

Proof. Consider a flow shop system with missing operations. When focusing specifically on scheduling tasks on the last machine M_m , the problem reduces to a single-machine scheduling problem with release dates, $1 \mid r_i \mid C_{\max}$, where the release date r_i of each task corresponds to its completion time on preceding machines.

The optimal solution for the $1 \mid r_i \mid C_{\max}$ problem involves sequencing tasks in ascending order of their release dates. Thus, scheduling tasks on machine M_m following the FIFO (First In, First Out) discipline does not increase the makespan and is therefore dominant. ■

Corollaire D.2.1.1 (Two-machine case). *In a two-machine permutation flow shop scheduling problem with missing operations, sequencing tasks according to Pinedo’s FIFO discipline on the second machine is dominant under the makespan criterion.*

12. A given permutation definition is said to be dominant if, for any feasible schedule that does not satisfy it, there exists a schedule that satisfies this definition and achieves a makespan that is less than or equal.

Théorème D.2.2. *In a two-machine permutation flow shop problem minimizing makespan, schedules respecting FIFO discipline on the second machine are dominant.*

Proof. Tasks arrive in the queue of the second machine either from machine 1 or directly if machine 2 corresponds to their first operation. To ensure that the queue on the second machine respects the FIFO discipline, we must show that if a task j finishes after a task i on machine 1, then task i must start processing before task j on machine 2.

Consider the first pair of tasks (i, j) such that task i precedes task j on machine 1, but task j precedes task i on machine 2. Suppose that tasks $i_1, \dots, i_k$ lie between i and j on machine 1. To correct this ordering, we remove task i from the queue on machine 1, shift tasks $i_1, \dots, i_k$ and task j forward, and then insert task i immediately after task j . This operation restores the correct order (i, j) on both machines without loss of feasibility or increase in makespan.

Repeating this procedure for all improperly ordered task pairs, we obtain a schedule that respects the FIFO discipline on machine M_2 , thereby proving the dominance of this scheduling approach. ■

Théorème D.2.3. *For a three-machine flow shop problem with missing operations with respect to the makespan criterion, Pinedo permutation schedules, which enforce FIFO discipline on machines 2 and 3, are dominant.*

Proof. In a three-machine flow shop setting, ensuring a FIFO queue discipline on machines 2 and 3 aligns with dominance conditions demonstrated in Theorems D.2.1 and D.2.2. Combining these two results confirms that FIFO-based schedules on M_2 and M_3 are dominant regarding makespan. ■

Remark : In the case without missing operations, Theorem D.2.3 aligns with the classical result presented in FRENCH, [1982a](#), which states that permutation schedules are dominant in three-machine flow shop problems under the makespan criterion.

In this section, we examined the impact and relevance of imposing the permutation constraint defined by Pinedo (based on the FIFO rule) in the general flow shop setting with missing operations. We now extend our analysis to a more general framework : the job shop scheduling problem, where FIFO constraints are integrated into the mathematical model. We propose a mathematical formulation that incorporates these constraints and investigate their impact on the makespan and waiting time.

D.3 ADAPTING PINEDO’S PERMUTATION CONCEPT TO THE JOB SHOP ENVIRONMENT

The objective of this section is to analyze how FIFO constraints can be enforced in job shop scheduling problems, and to evaluate their impact on makespan performance. Although job shops differ structurally from flow shops, the FIFO rule can still be applied to explore its influence on schedule quality. This rule is particularly important in service-oriented contexts and appointment management, where respecting the order of arrival is essential to ensure fairness and customer satisfaction.

The definition introduced by Pinedo for Flow Shop systems offers a compelling theoretical framework that can be tailored to Job Shop contexts. Pinedo defines permutation in Flow Shop scheduling (FIFO permutation) as a system where *all task queues adhere to the FIFO (First In, First Out) discipline*, ensuring that no job can bypass another once it is queued PINEDO, 2002b. Building on this, we suggest that applying this FIFO constraint could provide a theoretical basis for redefining permutation within Job Shop scheduling, drawing inspiration from Pinedo’s original formulation for Flow Shop environments.

To illustrate this, we consider a Job Shop example involving three jobs. Suppose jobs 1, 2, and 3 have processing times of (3, 2, 1), (2, 1, 2), and (1, 3, 3) respectively on three machines. with the machine order for each job being (1, 3, 2) for job 1, (3, 2, 1) for job 2, and (2, 1, 3) for job 3.

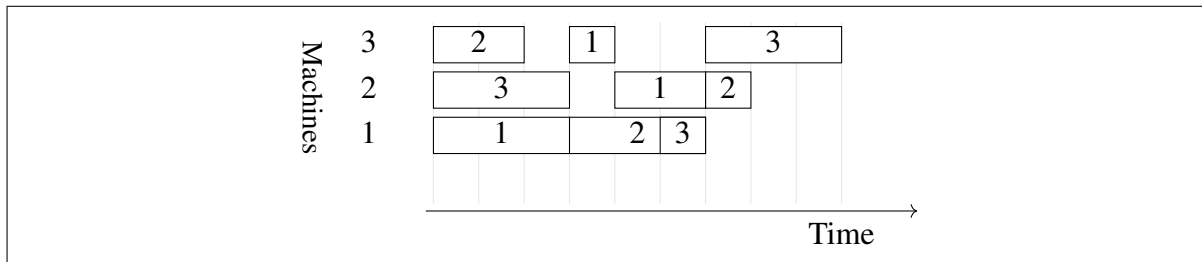


FIGURE D.4 : Scheduling with respect to FIFO in a job shop problem

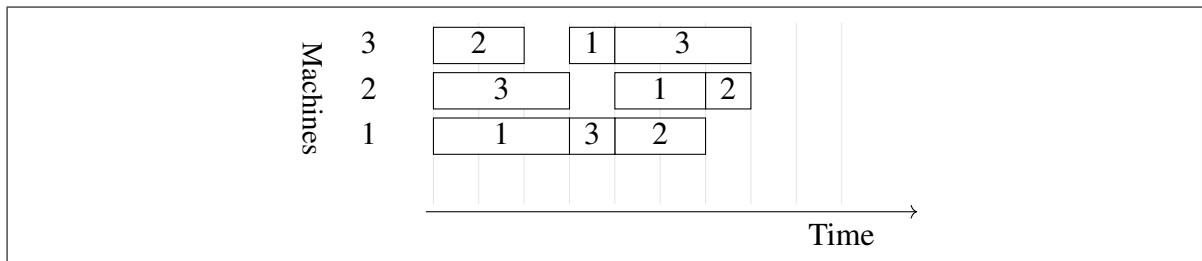


FIGURE D.5 : Scheduling without the FIFO constraint in a job shop problem

The FIFO (First-In, First-Out) rule is one of the most commonly used in service systems. In this context, other performance criteria, such as the minimization of the average waiting time, are also relevant in addition to the makespan. The objective of this section is to study the impact of the FIFO constraint in a Job Shop Scheduling Problem (JSP), focusing on both makespan and waiting time. We first present a mathematical formulation that integrates the FIFO constraint, followed by an extended model aiming to minimize waiting time.

D.3.1 MATHEMATICAL MODEL

Initially, we define the PJSP as a Job Shop model augmented with FIFO constraints. To integrate this FIFO constraint, we adapt the classical disjunctive JSP model presented in MANNE, 1960a by introducing additional parameters, variables, and constraints.

Before presenting the detailed model, we first introduce the essential notation :

Parameters

- i : Index for jobs, $i \in J = \{1, \dots, n\}$, with n representing the total number of jobs.
- j : Index for machines, $j \in M = \{1, \dots, m\}$, with m representing the total number of machines.
- k : Index denoting the k -th operation within job i .
- p_{ij} : Processing time of job i on machine j .
- m_{ik} : Machine assigned to the k -th operation of job i .
- L : A sufficiently large constant (Big-M).

Decision Variables

- S_{ij} : Start time of job i on machine j .
- $a_{i_1, i_2, j}$: Binary sequencing variable; $a_{i_1, i_2, j} = 1$ if job i_1 precedes job i_2 on machine j , and 0 otherwise.

To explicitly integrate the FIFO constraint into the JSP, we introduce an additional parameter : pr_{ij} denoting the machine assigned to the operation immediately preceding the operation scheduled on machine j for job i . For the first operation of job i , $pr_{i, m_{i1}} = -1$. With this parameter, the FIFO constraint is explicitly modeled as follows :

$$S_{i_1, pr_{i_1 j}} + p_{i_1, pr_{i_1 j}} + a_{i_1, i_2, j} L \geq S_{i_2, pr_{i_2 j}} + p_{i_2, pr_{i_2 j}},$$
$$\forall i_1 \neq i_2 \in N, \forall j \in M,$$

where $pr_{i_1 j}, pr_{i_2 j} \neq -1$. This constraint ensures that for any two jobs i_1 and i_2 processed on machine j , if job i_2 precedes job i_1 ($a_{i_1, i_2, j} = 0$), job i_2 must finish its preceding operation before job i_1 completes its own preceding operation.

Next, we extend both the classical JSP and the PJSP models to explicitly account for waiting times between successive operations of a job. While traditional scheduling models primarily focus on optimizing criteria such as makespan, they often overlook the impact of idle periods experienced by individual jobs. In many service-oriented environments, these waiting times can significantly affect

the perceived quality of service. Minimizing waiting time is therefore a crucial objective in such settings, as it directly influences both user satisfaction and the overall operational efficiency of the system AGNIHOTRI et al., 2024a. Incorporating waiting times into the objective function allows for a more balanced evaluation of schedules, especially in contexts where fairness, responsiveness, and client-oriented performance are key concerns. We introduce additional constraints and variables aimed at minimizing the average maximum waiting time. The extended models are referred to as the WJSP, which corresponds to the classical JSP with waiting time minimization, where the objective is to minimize the total waiting time while bounding the makespan by the optimal value of the classical JSP. Similarly, the WPJSP extends the PJSP (JSP with FIFO constraints) by minimizing the total waiting time while bounding the makespan by the optimal value obtained for the PJSP.

Additional Decision Variables

- w_{ij} : Waiting time between operation O_{ij} and the subsequent operation O_{ij+1} of job i .
- y_i : Maximum waiting time observed for job i across all its operations.

Additional Constraints for WJSP and WPJSP

- Waiting time calculation for each job i and each operation j (except the last) :

$$S_{i,O_{ij}} + p_{i,O_{ij}} + w_{i,O_{ij}} = S_{i,O_{ij+1}}, \quad \forall i \in \{1, \dots, n\},$$

$$\forall j \in \{1, \dots, m-1\}$$

- Maximum waiting time constraint per job :

$$y_i \geq w_{ij}, \quad \forall i \in \{1, \dots, n\}, \forall j \in \{1, \dots, m\}$$

The complete mathematical formulation of the WJSP model is presented below. In this formulation, $C_{\max}^{\text{JSP}}$ represents the optimal makespan value computed from the classical Job Shop Scheduling Problem.

$$\begin{aligned}
& \left\{ \begin{array}{ll} \min \frac{1}{n} \sum y_i & (1) \\ C_i \geq S_{ij} + p_{ij} \quad \forall i, j & (2) \\ C_{\max} \geq C_i \quad \forall i & (3) \\ S_{ij} \geq 0 \quad \forall i, j & (4) \\ S_{i,m_{ik}} + p_{i,m_{ik}} + w_{i,m_{ik}} = S_{i,m_{i(k+1)}} \quad \forall i, k < m & (5) \\ S_{i_1,j} + p_{i_1,j} \leq S_{i_2,j} + L(1 - B_{i_1 i_2 j}) \quad \forall i_1 \neq i_2, \forall j & (6) \\ B_{i_1 i_2 j} + B_{i_2 i_1 j} = 1 \quad \forall i_1 \neq i_2, \forall j & (7) \\ y_i \geq w_{ij} \quad \forall i, j & (8) \\ C_{\max} \leq C_{\max}^{\text{JSP}} & (9) \end{array} \right. \\
& \text{WJSP}
\end{aligned}$$

By adding the FIFO constraint to the WJSP model, we obtain the formulation of the WPJSP model, presented below. In this formulation, $C_{\max}^{\text{PJSP}}$ represents the optimal makespan value computed from the classical Job Shop Scheduling Problem with FIFO constraints.

$$\begin{aligned}
& \left\{ \begin{array}{ll} \min \frac{1}{n} \sum y_i & (1) \\ \text{subject to} & \\ (2), (3), (4), (5), (6), (7) \text{ and } (8), & \\ S_{i_1, \text{pr}_{i_1 j}} + p_{i_1, \text{pr}_{i_1 j}} + B_{i_1 i_2 j} L \geq S_{i_2, \text{pr}_{i_2 j}} + p_{i_2, \text{pr}_{i_2 j}}, & (10) \\ \forall i_1 \neq i_2 \in N, \forall j \in M, \text{pr}_{i_1 j}, \text{pr}_{i_2 j} \neq -1. & \\ C_{\max} \leq C_{\max}^{\text{PJSP}} & (11) \end{array} \right. \\
& \text{WPJSP}
\end{aligned}$$

It is worth noting that the constraint on the makespan differs between the two models : in WJSP, the upper bound is the optimal makespan obtained from the classical JSP without FIFO, whereas in WPJSP, the bound corresponds to the optimal makespan under FIFO constraints, with $C_{\max}^{\text{PJSP}} \geq C_{\max}^{\text{JSP}}$.

D.3.2 COMPUTATIONAL STUDY

In this section, we present the experimental study we conducted using the IBM CPLEX 20.1 solver on a cluster of 11 Dell PowerEdge R740 machines, each featuring 512GB of RAM and Intel Xeon Gold 6258R processors clocked at 2.70 GHz.

To evaluate the performance of our approach, we tested the models against a collection of well-established benchmark instances from the literature. The job sizes considered were $\{6, 10, 15\}$, paired

with machine configurations of $\{5, 6, 10, 15\}$. We conducted a comparative analysis by benchmarking our results against optimal solutions derived from standard Job Shop scheduling problem instances ADAMS et al., 1988b; APPELATE et COOK, 1991b; FISHER et THOMPSON, 1963c; LAWRENCE, 1984c; E. D. TAILLARD, 1993c. Only the instances that reached optimality within 3600 seconds were retained for the remainder of the study.

In Table 1, note that PJSP refers to the Optimal Solution of the Permutation Job Shop, while BKSJSP denotes the Best Known Solution of the Job Shop. The GAP metric is computed using the following formula :

$$\text{GAP (\%)} = \left(\frac{\text{BKSJSP} - \text{PJSP}}{\text{BKSJSP}} \right) \times 100.$$

The results reveal that the difference (labeled GAP in the table) between the classical Job Shop and the Permutation Job Shop is minimal, averaging 1.43%. Furthermore, 80% of the instances exhibit a gap below 2%. Figure D.6 shows that in the majority of instances, applying the FIFO constraint (PJSP) leads to a reduction in the average waiting time per job. On average, the improvement is about 19.8% compared to the classical JSP model.

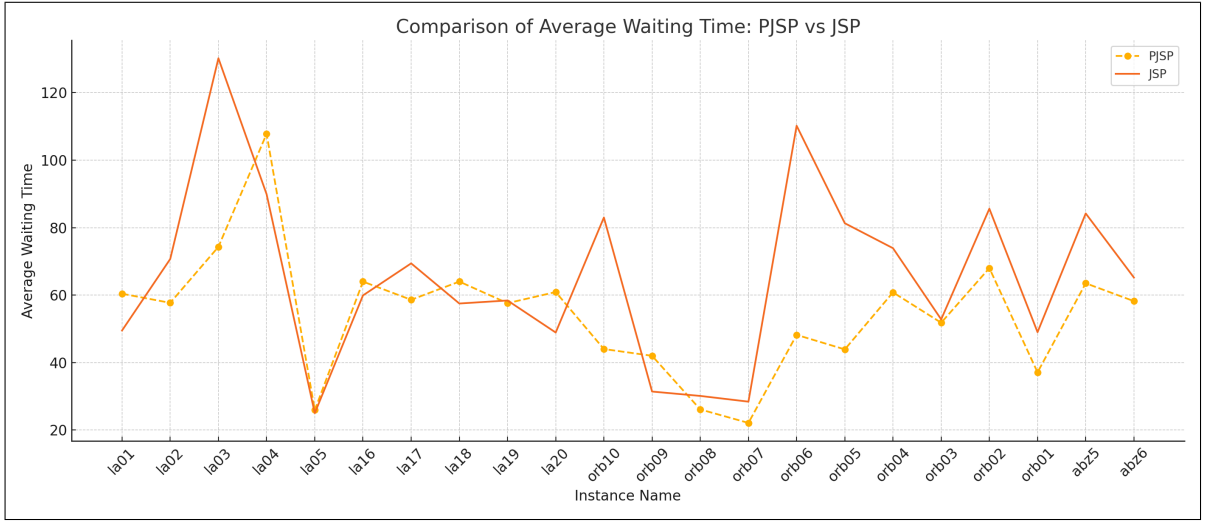


FIGURE D.6 : Comparison of average waiting time for JSP and PJSP

Thus, this section demonstrates that for the tested instances, the impact of FIFO on the makespan is not very significant, and more importantly, enforcing FIFO can significantly reduce job waiting times.

D.4 CONCLUSION

In this study, we analyzed the impact of enforcing FIFO constraints in two scheduling environments : the Flow Shop with missing operations and the Job Shop scheduling problem.

In the Flow Shop context, we examined how different definitions of permutation, such as strong, weak, and FIFO-based, can diverge when operations are missing. This theoretical analysis demonstrated that the choice of permutation definition can significantly affect the structure of feasible schedules and the dominance relationships between permutation and non-permutation solutions.

For the Job Shop problem, we adapted the classical disjunctive model to study how the FIFO discipline influences scheduling efficiency and solution quality. Our experiments on benchmark instances revealed that the PJSP (Permutation Job Shop based on FIFO) model results in only a negligible degradation in makespan. Fixing the makespan to $C_{\max}^{\text{PJSP}}$ in the WJSPmodel (without FIFO constraints) leads to higher average waiting times than in the WPJSP model, where FIFO constraints are enforced and the makespan is set to $C_{\max}^{\text{PJSP}}$. The WPJSP model yields an average reduction of 19.8% in job waiting times,

highlighting the practical benefits of incorporating FIFO constraints, especially in service-oriented applications where waiting time is a key factor in user satisfaction.

Future research could explore how FIFO enforcement affects delays in job shop environments where makespan minimization is the primary objective. In such cases, certain operations may be prioritized and processed earlier in machine queues. Strict FIFO enforcement, however, can introduce additional delays, as lower-priority operations may be forced to wait despite machine availability, solely to preserve queue order. These delays, especially in service-driven contexts, could negatively impact customer satisfaction by keeping resources idle rather than using them efficiently.

The proposed models were tested on instances of small to moderate size, involving up to 15 jobs and 15 machines, which reflect the complexity found in certain service-oriented applications. To generalize the findings and address a broader range of problem sizes, future research could explore alternative exact solution approaches built upon the proposed models, such as Lagrangian relaxation or branch-and-price techniques.

ACKNOWLEDGMENT

This research was partially funded by the Natural Sciences and Engineering Research Council of Canada (NSERC)

TABLE D.1 : Results on the Classical Benchmark Instances

Jobs	Machines	Instances	PJSP	BKSJSP	GAP (%)
Adams, Balas and Zawack ADAMS et al., 1988b					
10	10	abz5	1242	1234	0.65
10	10	abz6	945	943	0.21
Fisher and Thompson FISHER et THOMPSON, 1963c					
6	6	ft06	55	55	0.00
10	10	ft10	942	930	1.29
Lawrence LAWRENCE, 1984c					
10	5	la01	666	666	0.00
10	5	la02	671	655	2.44
10	5	la03	618	597	3.52
10	5	la04	591	590	0.17
10	5	la05	593	593	0.00
10	10	la16	953	945	0.85
10	10	la17	787	784	0.38
10	10	la18	854	848	0.71
10	10	la19	865	842	2.73
10	10	la20	907	902	0.55
15	10	la22	946	927	2.05
15	10	la24	962	935	2.89
15	10	la25	998	977	2.15
15	15	la36	1278	1268	0.79
15	15	la39	1250	1233	1.38
15	15	la40	1236	1222	1.15
Applegate and Cook APPLEGATE et COOK, 1991b					
10	10	orb01	1069	1059	0.94
10	10	orb02	890	888	0.23
10	10	orb03	1021	1005	1.59
10	10	orb04	1021	1005	1.59
10	10	orb05	898	887	1.24
10	10	orb06	1030	1010	1.98
10	10	orb07	404	397	1.76
10	10	orb08	907	899	0.89
10	10	orb09	937	934	0.32
10	10	orb10	968	944	2.54
Taillard E. D. TAILLARD, 1993c					
15	15	ta01	1256	1231	2.03
15	15	ta02	1265	1244	1.69
15	15	ta03	1248	1218	2.46
15	15	ta04	1197	1175	1.87
15	15	ta05	1247	1224	1.88
15	15	ta07	1245	1227	1.47
15	15	ta08	1253	1217	2.96
15	15	ta09	1299	1274	1.96
15	15	ta10	1273	1241	2.58

BIBLIOGRAPHY

- ADAMS, J., BALAS, E., & ZAWACK, D. (1988b). The shifting bottleneck procedure for job shop scheduling. *Management Science*, 34(3), 391-401.
- AGNIHOTRI, S., CAPPANERA, P., NONATO, M., & VISINTIN, F. (2024a). Appointment scheduling in surgery pre-admission testing clinics. *Omega*, 123, 102994.
- APPLEGATE, D., & COOK, W. (1991b). A computational study of job-shop scheduling. *ORSA Journal of Computing*, 3(2), 149-156.
- CONFORTI, D., GUERRIERO, F., GUIDO, R., CERINIC, M. M., & CONFORTI, M. L. (2011a). An optimal decision-making model for supporting week hospital management. *Health Care Management Science*, 14, 74-88.
- CORWIN, B. D., & ESOGBUE, A. O. (1974a). Two-machine flow shop scheduling problems with sequence-dependent setup times : A dynamic programming approach. *Naval Research Logistics Quarterly*, 21(3), 515-524.
- DUDEK, R. A., PANWALKAR, S. S., & SMITH, M. L. (1992). The lessons of flowshop scheduling research. *Operations Research*, 40(1), 7-13.
- FISHER, H., & THOMPSON, G. L. (1963c). Probabilistic learning combinations of local job-shop scheduling rules. In *Industrial Scheduling* (p. 225-251). Prentice-Hall.
- FU, Y., DING, J., WANG, H., & WANG, J. (2018a). Two-objective stochastic flow-shop scheduling with deteriorating and learning effect in Industry 4.0-based manufacturing system. *Applied Soft Computing*, 68, 847-855.
- HENNEBERG, M., & NEUFELD, J. S. (2016). A constructive algorithm and a simulated annealing approach for solving flowshop problems with missing operations. *International Journal of Production Research*, 54(12), 3534-3550.
- JIANG, X., TIAN, Z., LIU, W., SUO, Y., CHEN, K., XU, X., & LI, Z. (2022a). Energy-efficient scheduling of flexible job shops with complex processes : A case study for the aerospace industry complex components in China. *Journal of Industrial Information Integration*, 27, 100293.
- LAWRENCE, S. (1984c). *Resource Constrained Project Scheduling : An Experimental Investigation of Heuristic Scheduling Techniques (Supplement)* (rapp. tech.). Carnegie-Mellon University.
- LIANG, B., TURKCAN, A., CEYHAN, M. E., & STUART, K. (2015a). Improvement of chemotherapy patient flow and scheduling in an outpatient oncology clinic. *International Journal of Production Research*, 53(24), 7177-7190.
- MANNE, A. S. (1960a). On the job-shop scheduling problem. *Operations Research*, 8(2), 219-223.
- MARYNISSEN, J., & DEMEULEMEESTER, E. (2019a). Literature review on multi-appointment scheduling problems in hospitals. *European Journal of Operational Research*, 272(2), 407-419.

- NAMIKI, K., KOGA, H., AIDA, S., & HONDA, N. (1979b). An approach to the production line of automobiles by man-computer system. In *Case Studies in Automation Related to Humanization of Work* (p. 97-106). Elsevier.
- OUCHENE, R., REBAINE, D., & BAPTISTE, P. (2023a). Ambiguity of the definition of permutation flow shops in the presence of missing operations. *Computers & Industrial Engineering*, *182*, 109387.
- OUCHENE, R., REBAINE, D., & BAPTISTE, P. (2025). The impact of missing operations on the flow shop : permutation vs. non permutation modes. *RAIRO - Operations Research*, *59*(3), 1273-1293.
- PINEDO, M. (2002b). *Scheduling : Theory, Algorithms and Systems* (3rd). Springer.
- POTTS, C. N., SHMOYS, D. B., & WILLIAMSON, D. P. (1991a). Permutation vs. non-permutation flow shop schedules. *Operations Research Letters*, *10*(5), 281-284.
- RITT, M., & ROSSIT, D. A. (2024a). Effective heuristics for permutation and non-permutation flow shop scheduling with missing operations. *Computers & Operations Research*, *170*, 106742.
- RUIZ-TORRES, A. J., ABLANEDO-ROSAS, J. H., & OTERO, L. D. (2012a). Scheduling with multiple tasks per job—the case of quality control laboratories in the pharmaceutical industry. *International Journal of Production Research*, *50*(3), 691-705.
- SAADANI, N., BAHROUN, Z., & BOURAS, A. (2014c). A linear mathematical model for patients' activities scheduling on hospital resources. *Proceedings of the 2014 International Conference on Control, Decision and Information Technologies (CoDIT)*, 74-80.
- SERRANO-RUIZ, J. C., MULA, J., & POLER, R. (2022a). Development of a multidimensional conceptual model for job shop smart manufacturing scheduling from the Industry 4.0 perspective. *Journal of Manufacturing Systems*, *63*, 185-202.
- SRIDHAR, J., & RAJENDRAN, C. (1993d). Scheduling in a cellular manufacturing system : a simulated annealing approach. *International Journal of Production Research*, *31*(12), 2927-2945.
- TAILLARD, E. D. (1993c). Benchmarks for basic scheduling problems. *European Journal of Operational Research*, *64*(2), 278-285.
- VERMEULEN, I., BOHTE, S., SOMEFUN, K., & LA POUTRÉ, H. (2007). Multi-agent Pareto appointment exchanging in hospital patient scheduling. *Service Oriented Computing and Applications*, *1*, 185-196.
- VOGL, P., BRAUNE, R., & DOERNER, K. F. (2019a). Scheduling recurring radiotherapy appointments in an ion beam facility : considering optional activities and time window constraints. *Journal of Scheduling*, *22*(2), 137-154.
- WANG, S., & YU, J. (2010a). An effective heuristic for flexible job-shop scheduling problem with maintenance activities. *Computers & Industrial Engineering*, *59*(3), 436-447.