Théorie et applications en ordonnancement : contraintes de ressources et tâches agrégées en catégories

Vassilissa Lehoux

UNIVERSITÉ GRENOBLE 1 - JOSEPH FOURIER

Thèse
pour obtenir le grade de

DOCTEUR DE L'UNIVERSITÉ JOSEPH FOURIER

Discipline : Informatique

présentée et soutenue publiquement
par

Vassilissa LEBACQUE LEHOUX

Théories et applications en ordonnancement : contraintes de ressources et tâches agrégées en catégories

Thèse dirigée par : Gerd FINKE et Nadia BRAUNER

Soutenue le 6 septembre 2007

JURY

Jean-Charles BILLAUT,	Président du jury
Nadia BRAUNER,	Directeur
Gerd FINKE,	Directeur
Aziz MOUKRIM,	Rapporteur
Marino WIDMER,	Rapporteur
Bernard PENZ,	Examinateur

Résumé : Le thème de ce mémoire est l'ordonnancement dans les ateliers de production. L'objectif est d'étudier différents modèles classiques en analysant les liens et différences entre ces modèles et les problèmes pratiques associés. Les méthodes utilisées sont l'analyse de problèmes de nos partenaires industriels, l'étude de la complexité des problèmes ou de la structure des solutions et la proposition de méthodes de résolution exactes ou approchées.

Le premier axe de cette thèse est l'étude de problèmes d'ordonnancement avec contraintes de ressources d'entrée/sortie. Les environnements considérés sont les flowshops robotisés et le nouveau modèle d'indisponibilité des opérateurs.

Le second axe abordé concerne l'ordonnancement avec *high multiplicity* où les pièces sont agrégées en catégories. La description complète d'un ordonnancement (c'est-à-dire les instants de fabrication des tâches) n'est que pseudo-polynomiale de la taille de l'instance.

Theory and application in scheduling : resource constraints and high multiplicity

Abstract : This thesis deals with scheduling problems in manufacturing cells. It aims at studying different classical models, analyzing links and differences between those models and the practical associated problems. The approach is to analyze the problems of our industrial partners, to study the complexity of the problems or the structure of their solution, and to propose exact or approached solution methods.

The first axis of this thesis is the study of scheduling problems with in/out resource constraints. The problems considered are robotic flowshops and the new concept of operator non-availability.

The second axis concerns scheduling problems with high multiplicity, where parts are aggregated into categories. Complete description of a schedule (i.e. the starting times of the operations) is then only pseudo-polynomial of the size of the input.

Mots-Clés : Ordonnancement, ressources d'entrée/sortie, tâches agrégées en catégories, cellules robotisées, planification d'expériences, indisponibilité de l'opérateur, ordonnancement juste-à-temps, tâches couplées.

Keywords : Scheduling, in/out resources, high multiplicity, robotic cells, experiment planing, operator non-availability, just-in-time scheduling, coupled tasks.

Laboratoire G-SCOP, av. Félix Viallet, 38031 Grenoble Cedex.

Remerciements

Je souhaite tout d'abord exprimer toute ma gratitude à Nadia et Gerd qui ont dirigé mes recherches pendant ces trois années. Ils m'ont fait découvrir l'ordonnancement en tant qu'étudiante et m'ont accueilli par la suite dans leur équipe. Leur soutien, leur gentillesse et leurs connaissances m'ont été d'une aide précieuse tout au long de ma thèse et je ne serai les remercier suffisamment pour tout ce qu'ils m'ont apporté.

Tous mes remerciements vont ensuite aux membres de mon jury, à Jean-Charles Billaut qui en fut le président, à Aziz Moukrim et à Marino Widmer qui en furent rapporteurs et à Bernard Penz qui en fut examinateur. Je les remercie pour le temps qu'ils ont consacré à mon travail et les intéressantes remarques qu'ils m'ont faites.

Je tiens aussi à remercier vivement l'ensemble des personnels administratifs et scientifiques des laboratoires Leiniz-IMAG et G-SCOP, et en particulier leurs directeurs Nicolas Balcheff et Yanick Frein pour leur accueil. Il m'est bien sûr impossible d'oublier Henry qui a récupéré les informations contenues sur mon disque dur avant que celui-ci ne rende l'âme, m'évitant ainsi de nombreux problèmes. Mes remerciement les plus sincères vont aussi aux thésards de ces laboratoires pour l'ambiance agréable et le partage de nos expériences qui ont participé au bon déroulement de ma thèse et aux permanents qui ont toujours été prêts à répondre à mes questions.

Je remercie aussi les personnes avec qui j'ai eu la chance de collaborer au cours de ma thèse, Vincent avec qui j'ai partagé mon bureau, Christophe pour ses nombreuses idées, Benoît Celse pour sa confiance, Van Datt qui m'a incluse dans le projet Airbus, Vitaly pour son enthousiasme, Neslihan pour avoir été une étudiante et une amie.

Bien sûr, pour leur soutien et leur amitié, je remercie aussi Marion, Amandine, Cécile, Gaëlle, Emilie-Anne, David, Antoine, Ruben, Victor et Beinjamin avec qui j'ai partagé de nombreux moments pendant ma thèse.

Enfin, toute mon affection et ma tendresse vont à ma famille et belle-famille. Leur présence et leur confort, et en particulier ceux de mon mari, ont eu une place importante dans la réussite de cette thèse.

Table des matières

Introduction

Les ateliers de production sont une source inépuisable de problèmes d'optimisation. On ne compte plus les modèles introduits pour en souligner les nombreuses caractéristiques. À tel point qu'il est parfois difficile de choisir dans cette large étendue de possibilités le plus adéquat pour un problème donné, ou de décider s'il convient d'en créer un nouveau. Ce mémoire est consacré à l'ordonnancement dans les ateliers de production. Il ne s'agit pas bien sûr de faire une étude exhaustive de l'ensemble des modèles qui ont été initiés au cours de ces dernières années, mais pour un certain nombre d'exemples, théoriques et pratiques, d'analyser les liens et les différences entre certains modèles proches, de comparer la complexité des problèmes qui y sont associés, de déterminer dans certains cas des équivalences.

Nous nous attacherons, pour chacun des problèmes que nous abordons, à l'inscrire dans un cadre plus général en soulignant les liens avec d'autres problèmes, à déterminer sa complexité si cela n'a pas été fait et à proposer des méthodes de résolution exactes ou approchées.

Le premier axe de ce mémoire est dédié aux ateliers de production avec ressources dites d'*entrée/sortie* [65]. Avant le début de l'usinage d'une pièce, et/ou après que celui-ci soit terminé, l'intervention d'une machine ou d'un opérateur humain est nécessaire. Cela peut être pour effectuer le montage de la pièce sur la machine ou son démontage, pour préparer la machine à l'usinage d'une pièce différente de la précédente ou encore pour réaliser un certain nombre de tâches qui ne sont pas automatisées. Nous étudions les liens et distinctions entre les modèles classiques de la littérature pour différents types de ressources d'entrée/sortie qui interviennent dans les cellules de production. Lorsque nous nous tournons vers les flowshops robotisés, ce sont des robots qui effectuent ces tâches (Chapitre 2). Lorsque nous proposons un algorithme pour un problème industriel de planification d'expériences introduit par l'Institut Français du Pétrole (Chapitre 3), ce sont des opérateurs humains qui sont sollicités. Dans les problèmes avec ressources d'entrée/sortie classiques, la ressource est toujours supposée disponible. Ce n'est bien entendu pas le cas dans ce problème industriel, ce qui nous fait définir un modèle nouveau d'ordonnancement : l'indisponibilité des opérateurs (Chapitre 4). Pour tous ces problèmes, nous nous appliquons à étudier la complexité et à proposer des heuristiques le cas échéant. Minimiser la durée de la production dans un environnement de type flowshop robotisé à deux machines, lorsque les durées de transport sont indépendantes des pièces, est montré appartenir à $\mathcal{P}$, ainsi que plusieurs problèmes qui lui sont équivalents. Pour répondre à la problématique industrielle de l'Institut Français du Pétrole, nous implémentons un logiciel de planification, utilisé actuellement sur le site de Solaize, qui donne satisfaction à ses utilisateurs. Nous montrons, pour le problème d'indisponibilité des opérateurs, qu'il

n'est pas possible, même pour une unique machine, de déterminer en temps polynomial l'ordonnancement de plus courte durée, mais qu'il est possible d'obtenir des heuristiques polynomiales efficaces à l'aide de listes de priorité.

Le second axe de ce mémoire concerne les problèmes d'ordonnancement dit *high multiplicity*. Dans ces problèmes, les tâches ne sont plus toutes différentes mais regroupées par type, toutes les tâches d'une même catégorie ayant les mêmes caractéristiques. L'étude de la complexité des problèmes pour lesquels les tâches sont agrégées en catégories est rendue difficile par le fait que le nombre total de tâches n'est plus polynomial de la taille de l'instance. Décider si un problème donné appartient ou non à $\mathcal{NP}$ peut dès lors s'avérer ardu. Nous étudions ici deux applications. L'une est l'ordonnancement *juste-à-temps* (Chapitre 5). Pour ce problème, on souhaite produire différents types de tâches sur une ligne flexible. Le but est de répartir la production des différentes catégories de sorte qu'elle soit la plus homogène possible. Cet objectif est difficile à traduire sous la forme d'une fonction objectif concrète car de nombreuses méthodes mesurant la distance entre productions idéale et réelle ont été proposées dans la littérature. Une approche, qui a mené à de nombreuses conjectures, consiste à essayer d'optimiser simultanément plusieurs critères afin de les concilier. Nous montrons que cette optimisation simultanée ne sera pas toujours possible à travers un ensemble de contre-exemples et étudions la structure des solutions du problème. Une heuristique polynomiale est proposée en annexe qui construit des solutions pouvant se représenter de façon compacte. Le second problème *high multiplicity* que nous étudions est le problème des tâches couplées (Chapitre 6). Il s'agit d'ordonnancer des tâches, de façon cyclique ou non, sur une même machine, chaque tâche étant composée de deux opérations séparées par une durée fixée à l'avance. Nous nous intéressons tout d'abord à la version classique du problème, pour laquelle les tâches sont toutes différentes, et montrons l'équivalence de ce problème avec un problème particulier de cellules robotisées. Nous nous tournons ensuite vers le problème pour lequel les tâches sont agrégées en une seule catégorie, c'est-à-dire vers le cas où elles sont toutes identiques. Nous étudions la structure particulière de certaines solutions dans le cas cyclique et en déduisons une conjecture sur la complexité de ce problème.

Chapitre 1

Introduction à l'ordonnancement et à la complexité

Résoudre un problème d'ordonnancement, c'est trouver la "meilleure organisation" pour la réalisation d'un certain nombre de tâches. Cette meilleure organisation dépend à la fois des contraintes (ressources disponibles, précédences entre tâches, etc.) et de l'objectif que l'on s'est fixé : minimiser la durée totale de production (*makespan*) ou le coût pour réaliser toutes les tâches, minimiser le nombre de tâches en retard, etc.

Notre but n'est pas de présenter de façon exhaustive tous les problèmes d'ordonnancement possibles, mais de s'intéresser plus précisément aux problèmes posés par les ateliers de production, puisque c'est dans ce contexte que s'inscrit cette thèse.

Pour une présentation plus complète des problèmes d'ordonnancement et de leur complexité, le lecteur pourra se référer à [65, 15] ou aux sites `http://www.mathematik.uni-osnabrueck.de/` et `http://www.lix.polytechnique.fr/~durr/query/` qui recensent la complexité des problèmes d'ordonnancement classiques.

1.1 Description des problèmes d'ordonnancement et classification $\alpha|\beta|\gamma$

Dans un atelier de production, les tâches représentent souvent des produits à assembler. Pour réaliser les différentes opérations nécessaires pour obtenir le produit fini, on peut travailler sur une ou plusieurs machines, avoir besoin ou non d'un opérateur humain, utiliser des robots pour le transport des pièces...

Les possibilités étant nombreuses, nous utiliserons la classification des problèmes d'ordonnancement de Graham et al. [42] pour introduire les principales composantes des problèmes d'ordonnancement. Cette classification est composée de trois champs notés $\alpha|\beta|\gamma$ qui concernent respectivement l'environnement de production, les caractéristiques des tâches et de cet environnement et l'objectif que l'on s'est fixé.

1.1.1 Ressources humaines et matérielles : le champ α

Le premier champ de la notation $\alpha|\beta|\gamma$ proposée par Graham et al. [42] pour classifier les problèmes d'ordonnancement représente les ressources dont on dispose pour résoudre le problème considéré.

Ces ressources sont principalement de trois types :
- **Les machines**, aussi appelées processeurs, pourront, selon le problème étudié, permettre d'effectuer les mêmes opérations (on parle alors de machines *parallèles*) ou être *dédiées* à une ou plusieurs opérations particulières. Dans le cas de machines parallèles, il est possible de considérer des machines de vitesses de production différentes mais indépendantes de l'opération effectuée (*machines uniformes*) ou des machines de vitesses dépendantes de la tâche effectuée (*unrelated* en anglais). Nous nous intéressons dans cette thèse uniquement aux machines *identiques*, c'est-à-dire dont la vitesse de production est identique quelle que soit la machine et l'opération.
  La *capacité* d'une machine désigne le nombre d'opérations qu'elle peut réaliser simultanément. Si cette capacité n'est pas unitaire, on parlera d'*ordonnancement par batch.*
- **Les robots** sont utilisés principalement pour le transport des pièces entre les machines. Leur *capacité* désigne alors le nombre de pièces qu'ils peuvent transporter simultanément.
- **Les opérateurs** et **serveurs** interviennent pour le pré- ou post-traitement d'une opération. Il s'agit par exemple de préparer une machine ou une pièce à une opération, ou de nettoyer la machine après utilisation.

Chaque tâche correspond à une ou plusieurs opérations. Nous devons connaître l'ordre dans lequel les opérations d'une tâche doivent être réalisées et les ressources qui leur seront nécessaires (si celles-ci ne sont pas identiques pour chaque opération). Cette description de la séquence des opérations d'une tâche et des ressources associées s'appelle *gamme de production.*

Lorsque les gammes de production sont identiques pour toutes les tâches, on dit que l'environnement de production est de type *flowshop.* C'est souvent le cas dans les lignes de production où les machines sont disposées en série et où les pièces passent sur toutes les machines dans l'ordre dans lequel elles sont disposées.

Si l'ordre de passage des opérations sur les machines est différent d'une tâche à l'autre, le problème sera de type *jobshop* et si aucun ordre n'est imposé aux opérations des différentes tâches, on parlera d'*openshop.*

Classiquement, le champ α a deux composantes : $\alpha = \alpha_1\alpha_2$. La composante $\alpha_1 \in \{\varnothing, P, Q, R, F, O, J\}$ désigne le type de machines et de gammes du problème. Les valeurs classiques de α_1 sont les suivantes :

- $\alpha_1 = \varnothing$: une seule machine produit toutes les opérations.
- $\alpha_1 = P$: les machines sont parallèles et identiques.
- $\alpha_1 = Q$: les machines sont parallèles et uniformes. Elles ont des vitesses d'usinage différentes mais indépendantes de la pièce produite. Si on note s_i la vitesse de la machine M_i, alors la durée d'usinage $p_{i,j}$ de la pièce j sur la machine M_i est donnée par $p_{i,j} = p_j/s_i$ (à condition que l'usinage de la pièce j se fasse intégralement sur la machine i).

- $\alpha_1 = R$: les machines sont parallèles et non uniformes (*unrelated* en anglais). La vitesse de chaque machine sera alors différente selon la pièce usinée. Si on usine entièrement la pièce j sur la machine M_i, la durée de l'opération sera $p_{i,j} = p_j/s_{i,j}$ où $s_{i,j}$ est la vitesse d'usinage de j sur la machine M_i.
- $\alpha_1 = F$: les gammes de production sont identiques (l'environnement est de type *flowshop*).
- $\alpha_1 = O$: les gammes de production sont différentes, chaque pièce j est usinée une fois et une seule sur chaque machine mais l'ordre dans lequel doivent s'effectuer les opérations de chaque tâche n'est pas imposé (l'environnement est de type *openshop*).
- $\alpha_1 = J$: les gammes de production sont différentes mais fixées, une pièce peut être usinée aucune ou plus d'une fois sur chaque machine (l'environnement est de type *jobshop*).

auxquelles nous ajoutons
- $\alpha_1 = RC$: une cellule robotisée avec machines dédiées (voir Chapitre 2).

Dans le cadre de l'ordonnancement par batch, nous adoptons les notations de Potts et Kovalyov [74] : $\alpha_1 \in \{\tilde{P}, \tilde{Q}, \tilde{R}, \tilde{F}, \tilde{O}, \tilde{J}\}$ représentera les mêmes problèmes que précédemment, mais pour des machines de capacité supérieure à 1 et $\alpha = \tilde{1}$ représentera le problème à une machine. On ne fait pas apparaître la capacité des machines dans le champ α. Celle-ci sera, comme dans [74], indiquée dans le champ β.

Le champ $\alpha_2 \in \{\varnothing, m\}$ représente le nombre de machines. Si $\alpha_2 = \varnothing$, alors le nombre de machines est une variable du problème. Si $\alpha_2 = m$ le nombre de machines est égal à m.

Nous ajoutons à α deux composantes α_3 et α_4 : $\alpha = \alpha_1\alpha_2, \alpha_3\alpha_4$. Le champ α_3 permet de prendre en compte des ressources supplémentaires : les robots ($\alpha_3 = R$), serveurs ($\alpha_3 = S$), opérateurs ($\alpha_3 = O$), etc., le nombre de ces ressources étant indiqué par le champ α_4.

1.1.2 Caractéristiques des tâches, contraintes du problème, caractéristiques du matériel : le champ β

Le champ β comporte à l'origine 8 entrées : β_1, β_2, β_3, β_4, β_5, β_6, β_7, β_8. Chacune représente une caractéristique que peuvent présenter les tâches du problème posé. Dans le cas où l'une de ces entrées n'apparaît pas ($\beta_i = \varnothing$), les tâches du problème ne présenteront pas la caractéristique correspondante, ou celle-ci sera implicite.

- β_1 représente le mode d'exécution des opérations. Si $\beta_1 = pmtn$ (pour *preemption* en anglais), il est possible d'interrompre l'exécution de toute opération et de la reprendre au même point plus tard et éventuellement sur une autre machine.
- β_2 permet de décrire des ressources additionnelles en précisant le nombre de types de ressources, la quantité disponible de chaque ressource et le minimum de ressources nécessaires pour chaque tâche.
- β_3 représente les contraintes de précédence entre opérations : certaines opérations ne peuvent commencer tant que certaines autres ne sont pas terminées. Ces contraintes de précédence sont modélisées par un graphe orienté acyclique dans lesquels les sommets sont les opérations ou les tâches. Un arc (i, j) lie l'opération i (resp. la tâche i) à l'opération j (resp. la tâche j) si i doit être terminée pour que j puisse commencer. Si

ce graphe est quelconque, on notera $\beta_3 = prec$, mais on considère aussi plusieurs autres types de graphes : $\beta_3 = intree$, $outtree$, $chains$ si respectivement chaque sommet a au plus un successeur, au plus un prédécesseur et au plus un prédécesseur et un successeur.
- $\beta_4 = r_j$ représente les dates de disponibilité des tâches (*ready times*).
- β_5 décrit les durées de production des tâches (*production times*). Si $\beta_5 = \varnothing$, ces durées sont indépendantes les unes des autres, mais on considère souvent pour les problèmes difficiles le cas $\beta_5 = p_{i,j} = p$ (durées d'usinage toutes identiques) ou le cas $p_{i,j} = p_i$ (durées d'usinage indépendantes des pièces).
- Si $\beta_6 = \bar{d}_j$, la tâche j doit être terminée au plus tard à l'instant $\bar{d}_j$ (aucun retard n'est possible par rapport à cette date). On parle de *dead line* en anglais. Si le problème comporte des *dates d'échéance* (*due date* en anglais) plus souples, par rapport auxquelles des retards sont possibles (mais seront en général pénalisés dans fonction objectif), on notera d_j ces instants.
- Si $\beta_7 = n_j \leq k$, le nombre d'opérations qui composent chaque tâche est borné par k.
- L'entrée β_8 exprime l'existence d'une contrainte de *production sans attente* (*no-wait* en anglais). Si β_8 = no-wait, alors les tâches ne peuvent pas attendre après la fin d'une opération entre deux machines successives : l'opération suivante doit commencer immédiatement.

Le champs β est de plus utilisé pour décrire certaines caractéristiques du matériel ou des machines employées. Par exemple, pour les flowshop avec robot (Chapitre 2), on peut indiquer si les durées de transport des pièces sont indépendantes des pièces ($\beta_9 = \delta_{i,j} = \delta_i$) ou si les durées de chargement et déchargement sont considérées comme nulles ($\beta_{10} = \epsilon^l_{i,j} = 0$, $\beta_{11} = \epsilon^u_{i,j} = 0$). De même, pour les problèmes d'ordonnancement par batch, on peut indiquer par $\beta_{12} = b \leq c$ que la capacité des machines est de c.

1.1.3 Objectif : le champ γ

Le champ γ représente l'objectif de l'ordonnancement : obtenir une production de moindre coût ou de durée la plus courte possible, éviter les retards de livraison ou maximiser l'utilisation de certaines ressources. Dans cette thèse, nous nous intéressons plus particulièrement à deux critères :
- Terminer la production le plus tôt possible, c'est-à-dire minimiser la date de fin de la dernière tâche. Ce critère est noté $Cmax$.
- Minimiser la somme des dates d'achèvement des tâches, pour s'assurer d'obtenir des produits finis plus tôt dans le temps sans finir trop tard l'ensemble de la production. Ce critère est noté $\sum C_i$.

1.2 Complexité

Pour mesurer la difficulté d'un problème, on s'attache à estimer le temps nécessaire pour le résoudre. La théorie de la complexité permet de classer les différents problèmes en fonction de l'existence ou de la non existence d'algorithmes de résolution "rapides". On ne présente ici que les principes de base de la théorie de la complexité, mais le lecteur intéressé pourra consulter les ouvrages de Garey et Johnson [68] et Ausiello et al. [9].

1.2.1 Problèmes et évaluation de leurs algorithmes

Un *problème d'optimisation* se présente comme suit : il faut optimiser une certaine fonction de coût f tout en respectant un ensemble de contraintes $\mathcal{C}$. Les solutions *admissibles* du problème sont celles qui respectent toutes les contraintes de $\mathcal{C}$ et on essaie de choisir parmi elles celle qui nous permet d'obtenir la meilleure valeur pour f. Puisque maximiser f est équivalent à minimiser $-f$, nous considérerons par la suite que l'on veut résoudre

$$\begin{array}{l} \min f(s) \\ \text{s.c} \quad s \in \mathcal{C} \end{array}$$

À ce problème d'optimisation est associé un *problème de décision*. Cette fois-ci, la question n'est pas de trouver la solution de $\mathcal{C}$ qui minimise f mais de déterminer s'il existe une solution s de $\mathcal{C}$ telle que $f(s) \leq B$ où B est une borne qui fait partie des données.

De façon plus générale, on appelle *problème de décision* tout problème dont la réponse attendue est "oui" ou "non".

Pour classer les différents problèmes de décision et d'optimisation selon leur difficulté, nous nous intéressons à la vitesse d'exécution de leurs algorithmes.

Pour savoir si un algorithme est rapide ou non, on compte le nombre d'opérations élémentaires effectuées par l'ordinateur lors de son exécution. Ces opérations élémentaires sont par exemple l'addition ou la comparaison de deux nombres, l'affectation d'une valeur à une case mémoire, etc. Ce nombre d'opérations élémentaires est ensuite comparé à la taille mémoire nécessaire pour stocker les paramètres d'entrée de l'algorithme. Par exemple, si un algorithme prend uniquement un entier n en paramètre, on peut considérer que la taille de l'entrée est le nombre de bits nécessaires pour le coder, c'est-à-dire son nombre de chiffres en base 2, $\lfloor \log_2(n) \rfloor + 1$. On ne s'attache pas en théorie de la complexité à connaître précisément le nombre d'opérations qui seront effectuées ou la taille des données. On souhaite en fait obtenir un *ordre de grandeur* du nombre d'opérations quand la taille des données tend vers l'infini, et pas pour chaque instance le nombre exact d'opérations. Ainsi, la taille de n codé en base 2 est de l'ordre de $\ln(n)$, c'est-à-dire qu'il existe une constante strictement positive k et un entier n_0 tels que $\lfloor \log_2(n) \rfloor + 1 \leq k \ln(n)$ pour tout $n \geq n_0$. On note alors $\lfloor \log_2(n) \rfloor + 1 \in O(\ln(n))$ et on dit que la taille de n en base 2 est un *grand O* de $\ln(n)$.

Supposons maintenant que nous avons un algorithme $\mathcal{A}$ qui prend en entrée une donnée de taille N et qui a un temps d'exécution, ou nombre d'opérations effectuées, égal à $T(N)$.

Si $T(N) \in O(P(N))$ où $P(N)$ est un polynôme, on dira que l'algorithme $\mathcal{A}$ est *polynomial* (*polynomial-time algorithm* en anglais). Si ce n'est pas le cas, on dit que $\mathcal{A}$ est un algorithme *exponentiel*.

Dans la mesure où les fonctions exponentielles augmentent bien plus vite que les fonctions polynomiales, il est beaucoup plus intéressant d'obtenir des algorithmes polynomiaux qui pourront être utilisés avec de grandes données alors que les algorithmes exponentiels ne serviront que pour des instances de petite taille.

1.2.2 Les classes $\mathcal{P}$, $\mathcal{NP}$ et Co-$\mathcal{NP}$

La théorie de la complexité s'intéresse aux problèmes de décision et tente pour ceux-ci de répondre à des questions portant sur la qualité des algorithmes que l'on peut espérer pour ces problèmes.

La première question que nous nous posons, au regard de la Section 1.2.1, est la question de l'existence d'un algorithme de résolution polynomial pour le problème de décision considéré. Si un tel algorithme existe, il sera possible d'obtenir la réponse en un temps raisonnable, même si l'instance est de grande taille. On dira dans ce cas que le problème de décision appartient à la classe de problèmes $\mathcal{P}$.

Définition 1 (Classe $\mathcal{P}$) *Un problème Q de décision appartient à la classe $\mathcal{P}$ si et seulement si il possède un algorithme de résolution en temps polynomial.*

La plupart des problèmes d'ordonnancement qui sont abordés dans cette thèse sont des problèmes d'optimisation, auxquels la théorie de la complexité n'est apparemment pas reliée. Pourtant, il existe bien un lien entre un problème de décision et le problème d'optimisation associé. En effet, il est facile de voir que si l'on dispose d'un algorithme de résolution polynomial pour le problème d'optimisation, il suffit de regarder si la valeur optimale proposée pour la fonction objectif est inférieure à B pour obtenir la réponse au problème de décision pour la borne B. Réciproquement, si on dispose d'un algorithme polynomial pour le problème de décision, on peut par une recherche dichotomique trouver la valeur optimale de f en résolvant plusieurs fois le problème de décision. Ainsi, un problème d'optimisation disposera d'un algorithme de résolution polynomial si et seulement si le problème de décision associé est dans la classe $\mathcal{P}$.

Un enjeu important quand on s'intéresse à un problème d'ordonnancement est donc de déterminer si son problème de décision associé est dans $\mathcal{P}$. Pour répondre à cette question, nous introduisons les problèmes de la classe $\mathcal{NP}$.

Pour une instance donnée d'un problème de décision, deux réponses sont possibles : "oui" ou "non". Pour savoir si un problème de décision Q est dans $\mathcal{NP}$, on cherche à savoir s'il existe un *certificat* qui, étant donnée une instance pour laquelle la réponse à Q serait "oui", permettrait de vérifier rapidement que tel est bien le cas.

Définition 2 (Classe $\mathcal{NP}$) *Un problème de décision Q est dans la classe $\mathcal{NP}$ si pour toute instance x dont la réponse est "oui", il existe un certificat de taille polynomiale par rapport à la taille de x et vérifiable en temps polynomial.*

On parle de *certificat concis polynomial pour le "oui"*. Prenons un exemple pour illustrer cette notion de certificat. Nous considérons le problème suivant :

BIPARTITION
INSTANCE : Un ensemble de n entiers $a_1, a_2, \ldots, a_n$
QUESTION : Existe-t-il un sous-ensemble I de $\{1, 2, \ldots, n\}$ tel que $\sum_{i \in I} a_i = \sum_{i \notin I} a_i$?

L'instance $\mathcal{I} = \{a_1, a_2, \ldots, a_n\}$ est composée de n entiers, sa taille est donc de l'ordre de $n \max \ln(a_i)$ si on code les a_i en base 2. On suppose que le problème avec l'instance $\mathcal{I}$

obtient la réponse "oui". Dans ce cas, il existe un sous-ensemble I de $\{1, 2, \ldots, n\}$ tel que $\sum_{i \in I} a_i = \sum_{i \notin I} a_i$. Un certificat qui prouvera que la réponse est bien "oui" pour BIPARTITION sera donc un sous-ensemble I de $\{1, 2, \ldots, n\}$. Il sera bien de taille polynomiale puisqu'il aura au plus $n-1$ éléments. De plus on peut calculer $\sum_{i \in I} a_i$ et $\sum_{i \notin I} a_i$ en temps polynomial puisqu'on effectue n additions, ce qui nous permettra de vérifier la validité du certificat. On a donc un temps d'exécution polynomial.

On peut ainsi, pour une instance $\mathcal{I}$ telle que la réponse au problème BIPARTITION est "oui", fournir un certificat concis prouvant que la réponse est bien "oui". Le problème BIPARTITION est donc dans $\mathcal{NP}$. Il est important de noter que si on voulait par contre savoir si pour une instance la réponse était "non" pour BIPARTITION, on ne pourrait pas utiliser ce certificat.

Les problèmes disposant d'un certificat concis et vérifiable en temps polynomial pour le "non" appartiennent à une autre classe de complexité, la classe Co-$\mathcal{NP}$.

Définition 3 (Classe Co-$\mathcal{NP}$) *Un problème de décision Q est dans la classe Co-$\mathcal{NP}$ si pour toute instance x dont la réponse est "non", il existe un certificat de taille polynomiale par rapport à la taille de x et vérifiable en temps polynomial.*

On peut remarquer que les problèmes de $\mathcal{P}$ sont à la fois dans la classe $\mathcal{NP}$ et dans la classe Co-$\mathcal{NP}$, puisque la réponse, étant donnée une instance, sera obtenue en temps polynomial. Le certificat est donc dans les deux cas l'instance elle-même.

Il est fréquemment admis, mais ce n'est pas à l'heure actuelle démontré, que ces trois classes $\mathcal{P}$, $\mathcal{NP}$ et Co-$\mathcal{NP}$ sont distinctes. Si c'est bien le cas, il existerait des problèmes dans la classe $\mathcal{NP}$ pour lesquels aucun algorithme exact n'est polynomial. Cette hypothèse a conduit à définir les problèmes $\mathcal{NP}$-Complets, c'est-à-dire les problèmes les plus difficiles de la classe $\mathcal{NP}$. Ils ne peuvent admettre d'algorithmes polynomiaux que si $\mathcal{P} = \mathcal{NP}$. La section suivante explique comment montrer l'appartenance d'un problème à l'ensemble des problèmes $\mathcal{NP}$-Complets.

1.2.3 Problèmes $\mathcal{NP}$-Complet et $\mathcal{NP}$-Difficile au sens fort

Avant de définir un problème $\mathcal{NP}$-Complet, il nous faut introduire la notion de *réduction polynomiale*. On considère deux problèmes de décision Q et R. On dit que Q *se réduit à R en temps polynomial*, et on note $Q \propto R$, s'il existe une fonction calculable en temps polynomial f qui transforme toute instance I_q de Q en une instance $I_r = f(I_q)$ de R telle que la réponse soit "oui" pour l'instance I_q si et seulement si la réponse est "oui" pour l'instance I_r. Cette relation n'est pas une relation d'équivalence car elle n'est pas symétrique : le problème R peut-être strictement plus difficile que le problème Q. C'est le cas quand par exemple Q est une cas particulier de R, c'est-à-dire les solutions de Q doivent vérifier les mêmes contraintes que celles de R ainsi qu'une ou plusieurs contraintes supplémentaires, et les questions des deux problèmes sont identiques. Alors, on peut prendre $I_r = I_q$, et la transformation est bien polynomiale : il s'agit de "recopier" l'instance. On peut remarquer par contre que la relation est transitive : si $Q \propto R$ et $R \propto T$, alors $Q \propto T$ (on applique séquentiellement une transformation puis l'autre, le temps de calcul reste donc polynomial).

Les problèmes $\mathcal{NP}$-Complets sont les problèmes les plus difficiles de la classe $\mathcal{NP}$. Pour montrer qu'un problème Q est $\mathcal{NP}$-Complet, il faut donc montrer qu'il vérifie les deux conditions suivantes :

Définition 4 (Problème $\mathcal{NP}$-Complet) *Un problème de décision Q est $\mathcal{NP}$-Complet si et seulement si*
- *Q appartient à la classe $\mathcal{NP}$;*
- *tout problème R de $\mathcal{NP}$ peut se réduire à Q en temps polynomial, c'est-à-dire que Q est au moins aussi difficile que tout problème R de la classe $\mathcal{NP}$.*

Si un problème vérifie uniquement la deuxième condition, on dit qu'il est $\mathcal{NP}$-Difficile.

Cook a montré en 1971 [34] que tout problème de $\mathcal{NP}$ se réduit au problème *SAT*, qui est ainsi devenu le premier problème $\mathcal{NP}$-Complet connu, puisqu'il appartient à $\mathcal{NP}$.

Grâce à ce théorème, appelé *théorème de Cook*, et à la transitivité de la relation $\propto$, il a été possible de montrer qu'un grand nombre de problèmes sont $\mathcal{NP}$-Difficile. En effet, il n'est pas nécessaire de créer une réduction de tout problème de $\mathcal{NP}$ vers Q pour montrer que Q est $\mathcal{NP}$-Difficile. Il est suffisant de montrer qu'un problème $\mathcal{NP}$-Difficile se réduit à Q et la transitivité de $\propto$ permet de conclure.

> **Remarque.** Supposons que $Q \propto R$ et que la transformation calculable en temps polynomial f_{qr} correspond à cette réduction. Si R admet un algorithme polynomial A_r alors on obtient un algorithme polynomial pour Q en exécutant f_{qr} puis A_r. Ainsi,
>
> $$(Q \propto R \wedge R \in \mathcal{P}) \Rightarrow Q \in \mathcal{P}$$
>
> .

Il est généralement admis que les classes $\mathcal{P}$ et $\mathcal{NP}$ sont différentes et donc que les problèmes $\mathcal{NP}$-Complets n'admettent pas d'algorithmes polynomiaux. La remarque précédente indique que si un seul problème $\mathcal{NP}$-Complet admettait un algorithme polynomial, alors tout problème de $\mathcal{NP}$ se résoudrait en temps polynomial et donc qu'on aurait l'égalité des classes $\mathcal{P}$ et $\mathcal{NP}$. On suppose en général qu'il n'existe pas d'algorithme polynomial pour les problèmes $\mathcal{NP}$-Complets et que donc, le problème d'optimisation associé, s'il existe, n'admet pas non plus d'algorithme polynomial.

Certains problèmes, même s'ils n'admettent pas d'algorithmes exacts polynomiaux, pourront admettre des algorithmes relativement rapides appelés algorithmes *pseudo-polynomiaux*. Si un problème a dans ses données k entiers $n_1, n_2, \ldots, n_k$, ceux-ci sont codés en base 2 dans la mémoire d'ordinateur et leur taille est donc un $O(k \ln(\max n_i))$. Un algorithme pour ce problème sera pseudo-polynomial si le nombre d'opérations effectuées dépend de façon polynomiale des n_i (il sera donc exponentiel) et de la taille du reste de l'instance.

Pour certains problèmes de tels algorithmes pseudo-polynomiaux n'existent pas. On dit que ces problèmes sont *$\mathcal{NP}$-Difficiles au sens fort*.

Pour montrer qu'un problème Q est $\mathcal{NP}$-Difficile au sens fort, on cherche une réduction pseudo-polynomiale entre un problème $\mathcal{NP}$-Difficile au sens fort et Q.

1.2.4 Approximabilité

Lorsque le problème de décision associé à un problème d'optimisation est $\mathcal{NP}$-Difficile, on ne peut pas toujours utiliser un algorithme de résolution exact pour le problème d'optimisation, puisqu'il n'en existe pas de polynomial, sauf si $\mathcal{P} = \mathcal{NP}$. On se tourne alors vers des heuristiques dont on mesure la qualité selon deux critères : la qualité des solutions et la rapidité de l'heuristique.

Pour un problème d'optimisation Q donné, on note
- $\mathcal{I}_Q$ l'ensemble des instances de Q ou $\mathcal{I}$ s'il n'y a pas de confusion possible ;
- $Sol_Q(x)$ ou $Sol(x)$ l'ensemble des solutions admissibles de Q pour l'instance x ;
- $m(x, y)$ la valeur de la fonction objectif de la solution admissible y de l'instance x de Q ;
- $OPT(x)$ la valeur optimale de la fonction objectif pour l'instance x de Q ;
- $|x|$ la taille de l'instance x.

Pour mesurer la qualité des solutions d'une heuristique, on essaie d'obtenir des bornes sur l'écart entre la solution qu'elle fournit et la solution optimale. On parle de *ratio de performance*.

Définition 5 (Ratio de performance d'une instance) *Soient Q un problème d'optimisation, x une instance de Q et y une solution admissible pour x. Le* ratio de performance de y par rapport à x *est*

$$R(x, y) = \max\left\{\frac{OPT(x)}{m(x, y)}, \frac{m(x, y)}{OPT(x)}\right\}$$

Pour garantir la qualité des solutions d'une heuristique, on s'intéresse au ratio de performance de chaque solution proposée par rapport à son instance.

Définition 6 ($r(n)$-approximation) *Soient Q un problème d'optimisation et $(r(n))$ une suite à valeurs dans $[1, +\infty[$. On dit qu'un algorithme A est une* $r(n)$-approximation *si pour toute instance x de Q*
- *A délivre une solution admissible $A(x)$ du problème Q ;*
- *$m(x, A(x)) \leq r(|x|)OPT(x)$ si Q est un problème de minimisation ;*
- *$OPT(x) \leq r(|x|)m(x, A(x))$ si Q est un problème de maximisation.*

ou de façon équivalente :
- *A délivre une solution admissible $A(x)$ du problème Q ;*
- *$R(x, A(x)) \leq r(|x|)$.*

Définition 7 (Performance garantie) *Si un algorithme A est une $r(n)$-approximation pour le problème Q, on dit que $r(n)$ est une* performance garantie *de l'algorithme A.*

On s'attache en général à trouver la plus petite petite performance garantie qu'il est possible d'obtenir pour un algorithme. Il n'y a pas de consensus sur le nom qui pourrait lui être donné et nous adoptons ici la notation suivante

Définition 8 (Ratio de performance d'un algorithme) *Soient Q un problème d'optimisation, et A un algorithme qui pour chaque instance x de Q produit une solution $A(x)$ admissible pour x. Le* ratio de performance de l'algorithme A *est*

$$\sup\{R(x, A(x)) | x \in \mathcal{I}_Q\}$$

Une garantie de performance d'un algorithme A est donc une borne supérieure du ratio de performance de l'algorithme A.

On définit, pour les problèmes d'optimisation, différentes classes qui regroupent ces problèmes en fonction des ratios de performances qu'il est possible d'obtenir pour des algorithmes dont le temps d'exécution est raisonnable. Nous introduisons tout d'abord la classe $\mathcal{NPO}$ (pour $\mathcal{NP}$*-Optimization Problems* en anglais) qui est le pendant de la classe $\mathcal{NP}$ pour les problèmes d'optimisation.

Définition 9 (Classe $\mathcal{NPO}$) *Un problème d'optimisation Q est dans la classe $\mathcal{NPO}$ s'il vérifie les propriétés suivantes :*
- *On peut reconnaître en temps polynomial les instances de Q ;*
- *Si x est une instance de Q alors il existe un polynôme v tel que $\forall y \in Sol(x), \quad |y| \leq v(|x|)$. De plus, étant donnés x une instance de Q et y tel que $|y| \leq v(|x|)$, on peut décider si $y \in Sol(x)$ en temps polynomial ;*
- *Pour tout $x \in \mathcal{I}$ et tout $y \in Sol(x)$, $m(x, y)$ peut être calculée en temps polynomial.*

Les problèmes de cette classe ont l'avantage de posséder des solutions dont la taille est bornée par rapport à la taille de l'instance, ce qui est toujours le cas pour les problèmes de décision, puisque les seules réponses possibles sont "oui" et "non". La classe $\mathcal{NPO}$ est comme son nom l'indique liée à la classe $\mathcal{NP}$, puisque si un problème d'optimisation est dans la classe $\mathcal{NPO}$, alors le problème de décision associé est dans $\mathcal{NP}$ et le résultat suivant s'applique :

Théorème 1 *Si un problème Q d'optimisation est dans $\mathcal{NPO}$ et que le problème de décision associé est $\mathcal{NP}$-Complet, alors Q est $\mathcal{NP}$-Difficile.*

La classe $\mathcal{NPO}$ contient plusieurs sous-classes. Nous présentons ici celles qui seront utilisées dans ce mémoire.

Définition 10 (Classe $\mathcal{APX}$) *Un problème Q de $\mathcal{NPO}$ est dans la classe $\mathcal{APX}$ s'il admet une $O(1)$-approximation.*

Définition 11 (Classe $\mathcal{PTAS}$) *Un problème Q de $\mathcal{NPO}$ est dans la classe $\mathcal{PTAS}$ s'il admet un* schéma d'approximation polynomial*, c'est-à-dire s'il existe une famille d'algorithmes polynomiaux $(\mathcal{A}_\epsilon)_{\epsilon>0}$ telle que pour tout $\epsilon > 0$, $\mathcal{A}_\epsilon$ est une $(1+\epsilon)$-approximation de Q.*

En fait, cette définition ne demande aux algorithmes de la famille $(\mathcal{A}_\epsilon)_{\epsilon>0}$ que d'être polynomiaux par rapport à la taille des instances de Q et pas par rapport à la taille de $1/\epsilon$. Si on choisit ϵ arbitrairement petit, pour obtenir une solution très proche de l'optimal, on risque alors d'être confronté à un temps de calcul très important. La classe $\mathcal{FPTAS}$ est donc introduite :

Définition 12 (Classe $\mathcal{FPTAS}$) *Un problème Q de $\mathcal{NPO}$ est dans la classe $\mathcal{FPTAS}$ s'il admet un* schéma d'approximation pleinement polynomial, *c'est-à-dire s'il existe une famille d'algorithmes $(\mathcal{A}_\epsilon)_{\epsilon>0}$, polynomiaux par rapport à $1/\epsilon$ et par rapport à la taille de l'instance, telle que pour tout $\epsilon > 0$, $\mathcal{A}_\epsilon$ est une $(1+\epsilon)$-approximation de Q.*

On a donc les inclusions suivantes :

$$\mathcal{FPTAS} \subseteq \mathcal{PTAS} \subseteq \mathcal{APX} \subseteq \mathcal{NPO}$$

Pour montrer qu'un problème appartient à une des classes $\mathcal{APX}$, $\mathcal{PTAS}$ et $\mathcal{FPTAS}$, on peut exhiber une famille d'algorithmes avec les propriétés requises. Pour montrer qu'il n'appartient pas à $\mathcal{PTAS}$, on utilisera souvent le théorème suivant que nous appelons *théorème des Écarts* (*Gap Theorem* en anglais) pour faciliter les références futures.

Théorème 2 (Gap Theorem) *Soient Q un problème de minimisation de $\mathcal{NPO}$ et R un problème $\mathcal{NP}$-Difficile. Soient f un fonction calculable en temps polynomial qui transforme les instances de R en instances de Q et c une fonction calculable en temps polynomial qui transforme les instances de R en éléments de $\mathbb{N}$. Soit g une constante positive (le* gap*). Si les deux conditions suivantes sont vérifiées*
– Pour toute instance x de R dont la réponse est "oui", $OPT(f(x)) = c(x)$;
– Pour toute instance x de R dont la réponse est "non", $OPT(f(x)) \geq (1+g)c(x)$.
Alors il est impossible d'approximer Q en temps polynomial avec un ratio $r < 1+g$ sauf si $\mathcal{P} = \mathcal{NP}$.

1.2.5 Ordonnancement online

On parle d'ordonnancement *online* pour décrire des problèmes tels que l'ensemble des tâches et leurs caractéristiques ne sont pas connus à l'avance. Typiquement, les tâches ou les opérations ont des dates de disponibilité et ce n'est qu'à l'apparition de la tâche ou de l'opération dans le système que l'on connaît sa durée d'usinage. On ne sait qu'à la fin de la production quel sera le nombre total de tâches ou d'opérations. Le Chapitre 3 propose un exemple industriel de ce type de problème.

Si on dispose tout de même d'un certain nombre d'informations sur les tâches ou opérations qui vont arriver dans le système, on parle d'algorithme *semi-online* ou *nearly-online*. Par exemple, les tâches peuvent appartenir à un ensemble de types prédéfinis et les valeurs possibles pour leurs durées d'usinage être donc connues à l'avance. On peut encore imaginer que l'on connaisse à chaque instant la date d'arrivée de la prochaine tâche (c'est par exemple le cas au Chapitre 3). Ces informations complémentaires permettent d'améliorer les performances des algorithmes proposés pour ces problèmes.

Les algorithmes classiques utilisés pour résoudre des problèmes online sont présentés dans [65]. Pour mesurer la qualité d'un algorithme online, on le compare à un algorithme *clairvoyant* qui disposera à l'avance de toutes les informations relatives aux tâches, c'est-à-dire les dates de disponibilité et les caractéristiques de toutes les opérations. Un algorithme clairvoyant peut donc calculer la solution optimale. On dit qu'il résout le problème *offline*, c'est-à-dire le même problème pour lequel on disposerait de l'ensemble des informations.

Définition 13 ***(Algorithme c-compétitif)*** *On considère un problème Q d'optimisation online et une constante positive c. On dit qu'un algorithme A est* c-compétitif *si pour toute instance x de Q*
– A délivre une solution admissible $A(x)$ du problème Q ;
– $m(x, A(x)) \leq cOPT(x) + b$.
où $m(x, y)$ est la valeur de la fonction objectif de la solution admissible y de l'instance x, $OPT(x)$ est la valeur optimale de la fonction objectif pour le problème offline et l'instance x et b est une constante.

Définition 14 ***(ratio de compétitivité)*** *Le* ratio de compétitivité *(competitive ratio) c_A de l'algorithme A est la borne inférieure de l'ensemble des constantes c telles que A est c-compétitif.*

Un ratio de compétitivité de c_A ne signifie pas que A soit une c_A approximation. En effet, le ratio de compétitivité se calcule par rapport à la solution du problème offline et pas à la meilleure solution du problème online. Le fait que la meilleure solution offline soit plus simple à aborder participe bien sûr à ce choix du calcul des performances des algorithmes online.

Première partie

Problèmes avec ressources d'entrée/sortie

1 Introduction

Nous nous intéressons dans cette partie à des ateliers de production disposant de ressources dites *d'entrée/sortie* (*I/O ressources*). De telles ressources seront nécessaires pour préparer l'usinage d'une pièce sur une machine ou après que celui-ci soit terminé. Pendant la durée d'usinage, la ressource pourra être utilisée pour d'autres tâches présentes dans le système. Nous clarifions dans cette introduction les différences et liens entre trois ressources d'entrée/sortie : les robots, les serveurs et les opérateurs. En particulier, les serveurs et opérateurs sont présents dans la littérature sous différentes appellations et nous proposons un bref état de l'art permettant d'associer les articles présentés au type de problème effectivement étudié. Nous concentrons notre attention sur les caractéristiques de ces ressources et les hypothèses adoptées dans ce mémoire. Pour une présentation plus complète de l'ordonnancement avec ressources d'entrées/sorties, le lecteur peut se référer à [65].

1.1 Transports par un robot

Dans les environnements de type flowshop, jobshop et openshop, les pièces doivent être déplacées d'une machine à l'autre puisque leurs opérations ne s'effectuent pas toutes sur la même machine. Dans la version classique de ces problèmes, on considère que, dès qu'une opération se termine sur la machine M_i, la pièce qui vient d'être usinée est disponible pour l'opération suivante, même si elle se déroule sur une autre machine M_k. Cette hypothèse ne correspond pas toujours à la réalité puisque les machines dans un atelier peuvent être assez distantes, ou les pièces difficiles à transporter et les durées de transport ne sont alors plus négligeables par rapport aux durées d'usinages. Certain auteurs considèrent donc l'existence de *time-lags*, d'une durée minimum qui correspond au transport et qui séparent deux opérations de la même pièce. Cependant, cette modélisation suppose que suffisamment de ressources de transport soient disponibles à tout instant. Ce n'est pas toujours le cas dans les ateliers de production et nous nous intéresserons plus précisément aux cellules disposant d'un seul robot pour effectuer tous les transports des pièces. Un algorithme qui crée un ordonnancement pour une telle cellule de production doit alors prévoir, en plus des instants d'affectation des opérations aux machines, les mouvements du robot dans la cellule. Lorsque les transports sont effectués par un robot, on parle de flowshop, jobshop ou openshop *robotisés*. Le Chapitre 2 traite d'environnements de type flowshop robotisés et nous nous intéresserons plus particulièrement à ces problèmes par la suite.

Différents types de robots sont envisagés dans la littérature. Nous considérons dans ce mémoire uniquement ceux de capacité unitaire, c'est-à-dire pouvant transporter une seule pièce à la fois, et ne disposant pas de système d'échange de pièces, c'est-à-dire ne pouvant charger une pièce qu'une fois la précédente déchargée. On appelle ce type de robot *single gripper* en anglais.

Pour décrire les mouvements du robot dans un environnement de type flowshop, on utilise souvent le concept d'*activité* [37]. Une *activité* A_i du robot se décompose en trois étapes :
- Le robot vide prend une pièce j de la machine M_i ou d'un de ses buffers ;
- Le robot transporte la pièce j de la machine M_i à la machine M_{i+1} ;

– Le robot charge la pièce j sur la machine M_{i+1} ou sur l'un de ses buffers.
Cet ensemble de trois opérations s'effectue en général d'une seule traite, c'est-à-dire que le robot ne peut pas par exemple s'arrêter au milieu d'un transport pour attendre qu'une machine se libère. Une fois une activité commencée elle est effectuée intégralement.

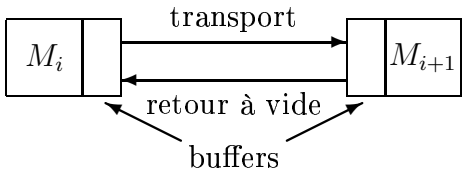


FIG. 1.1 – Déchargement de la machine M_i, transport et chargement sur la machine M_{i+1}

À une séquence d'activités correspond alors une séquence de déplacements du robot, chargé ou vide, puisque le robot, pour réaliser l'activité A_j après l'activité A_i doit se rendre de la machine M_{i+1} à la machine M_j.

Le robot, lors de la production, peut donc être amené à effectuer un trajet entre deux machines non consécutives. Dans un problème de type openshop ou jobshop, il pourra transporter une pièce entre ces deux machines, ou comme dans les problèmes de type flowshop, il pourra s'agir d'un déplacement du robot vide qui va chercher la prochaine pièce qu'il devra transporter. Selon la géométrie de la cellule, on fera des hypothèses différentes sur ces durées de déplacement.

On dira qu'elles sont *additives* si le robot passe pour se rendre de la machine M_i à la machine M_k, par toutes les machines intermédiaires (voir Figure 1.2 et Figure 1.3).

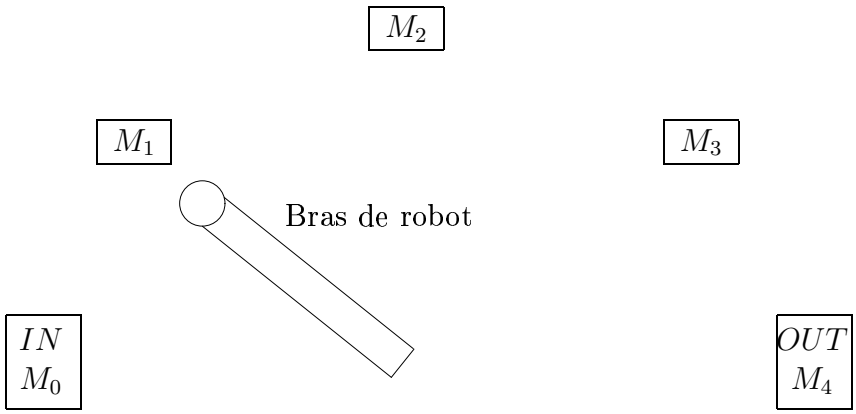


FIG. 1.2 – Cellule de production avec bras de robot

On note δ_i^e le temps mis par le robot vide pour se rendre de la machine M_i à la machine M_{i+1}. À titre d'exemple, la durée totale du déplacement du robot vide entre M_3 et M_1 sera $\delta_1^e + \delta_2^e$ si les durées de déplacement sont additives.

Il est fréquent dans les ateliers de production que les machines aient été placées de façon régulière. Dans ce cas, les durées de transport des pièces et de déplacement à vide du robot entre deux machines consécutives sont identiques quelle que soit la paire de machines considérée : les machines sont *équidistantes*. Les durées de transport des pièces ne dépendent alors que de la pièce transportée.

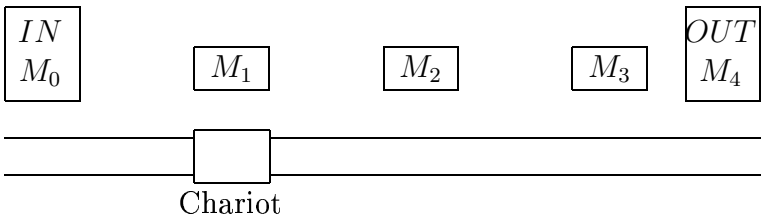


FIG. 1.3 – Cellule de production avec chariot robotisé

Les cellules de production circulaires dans lesquelles le robot peut se rendre directement de la première à la dernière machine sans repasser par aucune autre, ne sont pas étudiées dans ce mémoire. Pour une présentation complète des différentes cellules de production, le lecteur peut se référer à [39].

1.2 Serveurs et opérateurs

Dans cette section, nous clarifions la définition de deux ressources d'entrée/sortie, les serveurs et les opérateurs (voir aussi [13]).

Le chargement et le déchargement d'une pièce peuvent être considérés respectivement comme des setups et des démontages *non séparables* (*non-separable setups and dismountings* en anglais). Lors de setups et de démontages non séparables, la pièce est nécessaire au même titre que la machine, alors que des setups ou démontages séparables peuvent être effectués sur la machine sans que la pièce ne s'y trouve. Ces setups et démontages sont dans les deux cas réalisés par des ouvriers ou des robots.

La complexité des problèmes avec chargement dans un environnement de machines parallèles a été largement étudiée dans la littérature [46, 57, 26, 1, 2, 3]. L'ouvrier est alors appelé *serveur* et la notation $\alpha|\beta|\gamma$ est étendue pour ce problème par Hall et al. [47, 46]. Le champ α est fixé à Pm, Ss pour un problème à m machines parallèles et s serveurs.

Les problèmes de flowshop avec un unique serveur $Fm, S1$ sont donc des cas particuliers des problèmes de flowshop avec robot dans lesquels
- les machines n'ont pas de buffer d'entrée ;
- les durées de transport des pièces sont nulles : la pièce j est disponible à la machine M_{i+1} dès que son usinage est terminé sur la machine M_i ;
- les durées de déchargement sont nulles : on n'a besoin de l'ouvrier que pour effectuer un setup.

Il serait bien entendu envisageable, bien que cela n'ait pas été fait à notre connaissance, d'élargir cette notion de serveur pour qu'il puisse effectuer aussi le déchargement des pièces. Le serveur serait toujours un cas particulier du robot puisqu'il correspondrait alors au cas où
- les machines n'ont pas de buffers d'entrée ni de sortie ;
- les durées de transport des pièces sont nulles : la pièce j est disponible à la machine M_{i+1} dès que son usinage est terminé sur la machine M_i.

Lorsque des setups et des démontages séparables sont effectués par un robot ou un ouvrier,

Cheng et al. [33] appellent cet ouvrier *opérateur* et étudient les flowshops à deux machines et un unique opérateur. Ils fixent le champ α à $F2, O1$. Brucker et al. [30] considèrent le cas où les durées de démontage sont nulles et étudient la complexité de nombreux problèmes de flowshops avec un opérateur. Dans ce papier, ils ne reprennent pas les notations de Cheng et al. [33], mais appellent leur opérateur serveur. Pour un atelier avec machines parallèles, il n'est pas nécessaire d'établir une distinction entre setups séparables et non séparables, puisque les tâches n'ont qu'une seule opération. Cependant, puisque les serveurs ont été explicitement définis comme des dispositifs effectuant le chargement des pièces sur les machines, nous utiliserons les notations de [33] pour ce problème et considérerons par la suite que les serveurs réalisent des setups non séparables.

Le Chapitre 2.2 énonce plusieurs résultats portant sur les flowshops avec robot dont certains peuvent être appliqués aux flowshop avec un unique serveur qui en sont un cas particulier. Dans le Chapitre 3, nous présentons un problème industriel avec ressources d'entrée/sortie. Ces ressources sont des chimistes qui doivent effectuer plusieurs types d'opérations qui relèvent à la fois des serveurs et des opérateurs. Les contraintes particulières liées à certaines opérations nous conduisent de plus à introduire un nouveau concept lié aux absences de ces opérateurs humains qui ne sont pas présents sur le site de production pendant les week-ends, les jours fériés ou les vacances. Ce concept est développé en détail dans le Chapitre 4. Nous parlons d'*indisponibilité d'opérateurs*. Le mot *opérateur* correspond alors à un *opérateur humain* et pas au concept d'opérateur introduit ici.

Chapitre 2

Flowshops robotisés : modèles et problèmes à deux machines

Nous considérons une cellule de production dans laquelle nous souhaitons usiner un ensemble de n pièces $j = 1, 2, \ldots, n$ sur m machines $M_1, M_2, \ldots, M_m$. La production de chaque pièce j est divisée en m opérations qui doivent être produites sans préemption et dans le même ordre sur les m machines. Dans les problèmes de flowshops classiques, dès que la $i^{ème}$ opération de la pièce j est terminée sur la machine M_i, la pièce est disponible et l'usinage pourrait commencer immédiatement sur la machine M_{i+1}. Dans les problèmes de flowshop robotisés, les pièces sont transportées par un robot d'une machine à l'autre. Ce problème fut à l'origine introduit dans [7] et étudié dans [80].

2.1 Cellules robotisées et flowshops avec robot

On trouve dans la littérature, sous la même appellation de *cellules robotisées*, deux sortes de flowshops robotisés. Ces deux configurations d'atelier ont été étudiées de façon quasiment indépendantes dans la littérature malgré leurs nombreuses similarités. Ici, nous faisons clairement apparaître les liens et différences entre les deux modèles.

Dans la première configuration, des stations d'entrée ($IN = M_0$) et de sortie ($OUT = M_{m+1}$) se trouvent à l'entrée et à la sortie de la cellule de production. Elles ont pour rôle de stocker les pièces avant le départ et après la fin de leur production. Dans ces cellules, le robot doit donc, en plus d'assurer le transport des pièces entre les machines, transporter les pièces entre la station d'entrée et la première machine et entre la dernière machine et la station de sortie [36, 22, 38, 17]. Pour cette configuration, on parlera de *cellules robotisées* et la valeur du champs α de la notation $\alpha|\beta|\gamma$ sera RCm (voir Figure 2.1 pour une cellule robotisée à trois machines, $RC3$).

Si la première machine est située immédiatement après la station d'entrée, on peut s'attendre à ce que la durée des transports entre IN et M_1 puisse être considérée comme nulle. On supposera alors que le robot n'est pas nécessaire pour effectuer les transferts entre IN et M_1 (on peut imaginer que IN se comporte comme un tapis roulant à l'entrée

Les résultats de ce chapitre ont fait l'objet d'un article soumis à Computers and OR [60]

de M_1) et que IN est le buffer d'entrée de M_1. Si c'est le cas, et que OUT devient aussi un buffer de M_m et non plus une station, alors on parlera de *flowshops avec robot* (voir Figure 2.2 pour un flowshop avec robot à trois machines). Dans cette configuration, toutes les pièces sont disponibles en M_1 au début de l'ordonnancement et quittent la cellule de production dès que leur usinage est terminé en M_m. Comme dans [50], nous adopterons la valeur $Fm, R1$ pour le champs α de la notation $\alpha|\beta|\gamma$.

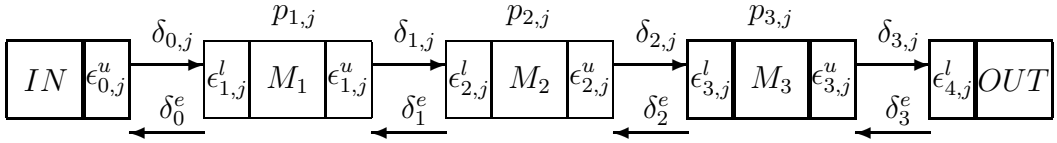


FIG. 2.1 – Notations pour le problème de cellules robotisées à 3 machines, $RC3$

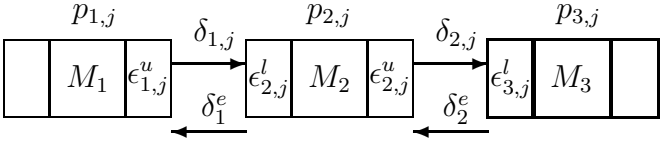


FIG. 2.2 – Notations pour le problème de flowshop avec robot à 3 machines, $F3, R1$

Dans ce chapitre, nous supposerons que le robot et les machines sont de capacités unitaires, c'est-à-dire que le robot ne peut transporter qu'une pièce à la fois et que chaque machine ne réalise l'usinage que d'une pièce à la fois. Nous choisissons de plus de distinguer les buffers d'entrée et de sortie des machines. Les espaces de stockages, s'ils sont placés à l'entrée de la machine, servent à décharger les pièces et les stockent jusqu'à leur usinage. Si des buffers sont placés à la sortie de la machine, ils stockeront des pièces dont l'usinage est terminé avant leur déchargement.

Nous utiliserons, pour les deux types de flowshop robotisés que nous avons introduits, les notations suivantes :
- n, le nombre de pièces ;
- p_{ij}, la durée d'usinage de la pièce j sur la machine M_i ;
- $\delta_{i,j}$, le temps mis par le robot pour transporter la pièce j de la machine M_i à la machine M_{i+1} ;
- δ^e_i, la durée du trajet à vide du robot de la machine M_i à la machine M_{i+1} ou de la machine M_{i+1} à la machine M_i ;
- ϵ^u_{ij}, la durée de déchargement de la pièce j de la machine M_i ;
- ϵ^l_{ij}, la durée de chargement de la pièce j sur la machine M_i.

Dans les deux cas, quel que soit le critère optimisé, il faudra trouver à la fois la séquence des mouvements du robot et l'ordre des pièces qui permettent d'obtenir sa valeur optimale.

Sous les hypothèses que nous considérons, le robot doit être vide avant de pouvoir décharger une pièce d'une machine pour la transporter. De même nous considérons que les machines sont de capacité unitaire et doivent donc être vides pour pouvoir recevoir une nouvelle pièce. Alors, en l'absence d'espaces de stockage (ou *buffers* en anglais) dans la cellule de production, il n'est pas possible pour le robot d'arriver chargé sur une machine déjà occupée par une pièce.

Par la suite, nous nous intéresserons particulièrement à deux cas :

- le cas *sans buffer* (*no-buffer* en anglais), où aucun espace de stockage n'est disponible entre les machines ;
- le cas avec espaces de stockage illimités (*illimited buffers* en anglais), avant et après chaque machine.

Les espaces de stockage que nous considérons, s'ils existent, font partie de la machine, c'est-à-dire que le transfert d'une pièce de la machine à un de ses buffers ou d'un de ses buffers à la machine est instantané et ne nécessite pas le robot. Cette hypothèse est importante comme le montre la Section 2.1.1, puisque la modélisation du problème sera différente selon que le robot soit nécessaire ou non pour ce transfert.

2.1.1 Cas d'équivalence et conséquences

Les cellules robotisées sont un cas particulier de flowshop avec robot tels que les temps d'usinage sur la première et la dernière machine sont nuls. Cela implique que, si un problème de cellules robotisées à m machines est $\mathcal{NP}$-Difficile, alors le problème de flowshop avec robot et $m+2$ machines correspondant est lui aussi $\mathcal{NP}$-Difficile. De nombreux résultats de complexité pour les problèmes de flowshop avec robot peuvent ainsi être déduits de résultats existants pour les cellules robotisées. En fait, si on minimise le makespan, $Cmax$, dans des cellules robotisées à trois machines no-wait ou sans buffers, le problème sera $\mathcal{NP}$-Difficile, même si les durées de chargement et déchargement et les durées de transport sont indépendantes des pièces [5, 45]. Pour deux machines, les mêmes problèmes sont polynomiaux [45, 4]. Ainsi, les problèmes correspondants de flowshop avec robot à cinq machines et un robot sont aussi $\mathcal{NP}$-Difficiles alors que la complexité des mêmes problèmes pour deux, trois et quatre machines ne peut pas être dérivée immédiatement des résultats de complexité des cellules robotisées.

Dans le cas où les durées de transport entre la station d'entrée et la première machine et entre la dernière machine et la station de sortie sont nulles, il est plus logique de considérer qu'on n'a pas besoin du robot pour effectuer les transferts des pièces entre IN et M_1 et M_m et OUT. Si cependant on ne souhaite pas faire cette hypothèse, on peut montrer que, si les durées de chargement sur M_1 et OUT et les durées de déchargement de IN et M_m sont nulles, alors, sous certaines conditions sur les espaces de stockage en M_1 et M_m, le problème de cellule robotisée est équivalent au problème de flowshop avec robot associé.

Proposition 1 *On considère un problème de flowshop avec robot à $m \geq 2$ machines tel que la capacité du buffer de sortie de M_1 soit c et la capacité du buffer d'entrée de M_m soit c'. Ce problème est équivalent au même problème de cellules robotisées ayant les contraintes supplémentaires suivantes :*

1. *$\delta_{0,j} = \delta_{m,j} = 0$, $\delta^e_0 = \delta^e_m = 0$, la distance entre IN et M_1 est nulle ;*
2. *$\epsilon^u_{0,j} = \epsilon^u_{m,j} = \epsilon^l_{1,j} = \epsilon^l_{m+1,j} = 0$, les durées des chargements et déchargements qui correspondent à ces trajets sont nulles ;*
3. *la capacité du buffer d'entrée de M_1 est supérieure ou égale à $c+1$;*
4. *la capacité du buffer de sortie de M_m est supérieure ou égale à $c'+1$.*

Démonstration. Pour se convaincre de cette équivalence, il suffit de remarquer que dans le problème de cellules robotisées, le robot commencera l'ordonnancement en remplissant

intégralement le buffer d'entrée de M_1. Puis, à chaque fois qu'il retournera en M_1 pour prendre une pièce, il commencera, avant d'emmener cette pièce, par remplir à nouveau complètement ce buffer (ce qui lui prend une durée nulle par hypothèse). Cela permet de garantir que tout au long de l'ordonnancement, le robot n'aura pas besoin d'être présent pour que l'usinage des pièces commence sur la première machine. De la même façon, dès qu'il se rendra en M_m, il videra le buffer de sortie qui est de capacité suffisante pour recueillir toutes les pièces qui se trouvaient sur la dernière machine ou son buffer d'entrée lors de son dernier passage. □

On peut remarquer que si on ne prend pas l'une ou l'autre des hypothèses 3 ou 4 sur les capacités des buffers d'entrée de M_1 et de sortie de M_m, les problèmes ne sont plus équivalents. On donne ici des exemples qui illustrent ces propos pour différentes configurations.

Si la capacité du buffer d'entrée de M_1 est inférieure strictement à $c+1$, alors les deux problèmes ne sont pas équivalents, même si les conditions 1, 2 et 4 de la Proposition 1 sont vérifiées.

Exemple 1. On considère l'exemple à deux machines suivant.
Pour les deux configurations, les capacités du buffer de sortie de M_1 et d'entrée de M_2 sont nulles. Pour la configuration $RC2$, la capacité du buffer d'entrée de M_1 est nulle et celle du buffer de sortie de M_2 est égale à 1. Les durées d'usinage et de transports sont les suivantes :

pièce j	$p_{1,j}$	$p_{2,j}$	$\delta_{1,j}$	$\epsilon^u_{1,j}$	$\epsilon^l_{1,j}$
1	3	5	4	0	0
2	5	6	4	0	0

avec $\delta^e = 0$.

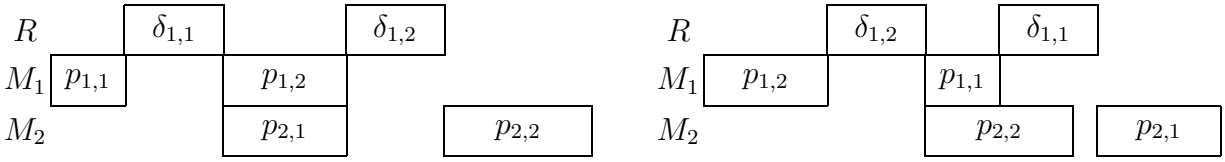


FIG. 2.3 – Configuration $RC2$: séquences $(1,2)$ avec $Cmax = 22$ et $(2,1)$ avec $Cmax = 21$

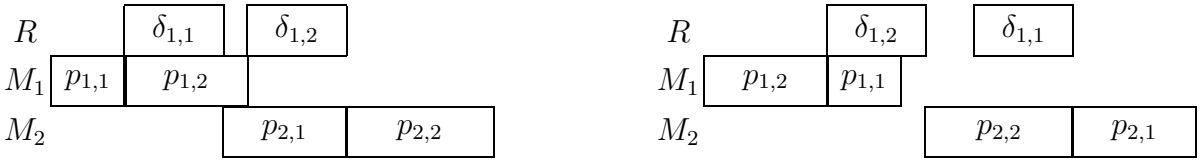


FIG. 2.4 – Configuration $F2R1$: séquences $(1,2)$ avec $Cmax = 18$ et $(2,1)$ avec $Cmax = 20$

Dans cet exemple, les deux problèmes ont des solutions optimales différentes et ne sont donc pas équivalents (voir Figures 2.3 et 2.4).

Si c'est maintenant la condition 4 de la Proposition 1 qui n'est pas vérifiée alors que les autres le sont, on peut à nouveau fournir un exemple pour lequel les deux problèmes ne

sont pas équivalents. Il suffit de considérer l'exemple précédent en inversant les durées d'usinage des pièces sur la première et sur la seconde machine.

Dans la Section 2.2, nous étudions des problèmes de flowshops avec robot dans des cellules sans buffer. Ces problèmes ne sont bien sûr pas équivalents aux problèmes de cellule robotisée associés sans buffers et avec des durées de transport, chargement et déchargement nulles entre IN et M_1 et M_2 et OUT.

Exemple 2. On considère l'exemple à deux machines suivant.
Pour les deux configurations, la capacité du buffer de sortie de M_1 et d'entrée de M_2 sont nulles. Pour la configuration $RC2$, la capacité du buffer d'entrée de M_1 et de sortie de M_2 sont nulles. Les durées d'usinage et de transports sont les suivantes :

pièce j	$p_{1,j}$	$p_{2,j}$	$\delta_{1,j}$	$\epsilon^u_{1,j}$	$\epsilon^l_{1,j}$
1	6	4	6	0	0
2	3	2	6	0	0

avec $\delta^e = 0$.

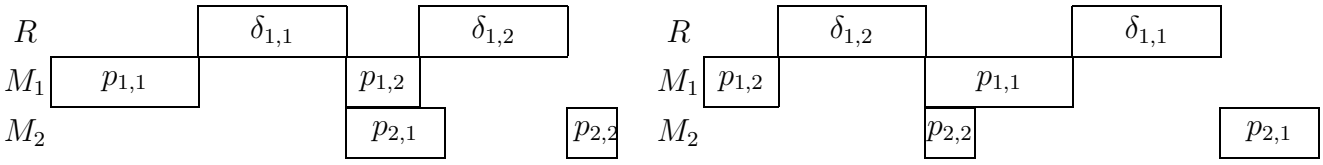


FIG. 2.5 – Configuration $RC2$: séquences $(1,2)$ avec $Cmax = 23$ et $(2,1)$ avec $Cmax = 25$

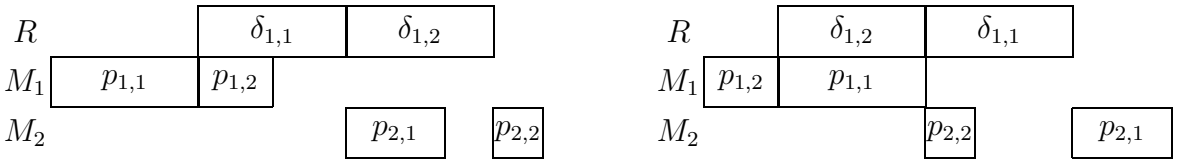


FIG. 2.6 – Configuration $F2R1$: séquences $(1,2)$ avec $Cmax = 20$ et $(2,1)$ avec $Cmax = 19$

Pour cet exemple, les deux problèmes ont des solutions optimales différentes et ne sont donc pas équivalents.

2.1.2 Pertinence des modèles sans durées de chargement et déchargement

Dans de nombreux articles, les auteurs prennent comme hypothèse que les durées de chargement et de déchargement sont nulles [55, 54, 48, 4, 5, 50, 63]. Dans cette section, nous démontrons que cette supposition peut être faite sans perte de généralité dans plusieurs cas dépendant de la capacité des buffers dans la cellule de production.

L'existence ou non de buffers avant et après chaque machine influe sur les ressources nécessaires au transport, au chargement et au déchargement. Dans le cas sans buffer par exemple, le chargement et le déchargement nécessitent la pièce, le robot et la machine, alors que dans le cas de buffers de capacité infinie, seuls la pièce et le robot sont nécessaires

pour ces opérations. Si les ressources utilisées pour réaliser transport, chargement ou déchargement sont identiques, on peut alors considérer que les durées de chargement et déchargement sont nulles : elles peuvent être incluses dans les durées de transport.

Proposition 2 *Si le buffer d'entrée de la machine M_i (respectivement son buffer de sortie) est de capacité infinie, alors on peut considérer sans perte de généralité que la durée de chargement des pièces sur M_i est nulle (respectivement que la durée de déchargement des pièces de M_i est nulle).*

Démonstration. Quand le buffer associé est illimité, le chargement ou le déchargement des pièces nécessite uniquement la pièce et le robot. On peut donc inclure les durées de chargement ou de déchargement dans les durées de transport. □

Pour les cellules robotisées, on peut obtenir un résultat similaire pour le cas sans espaces de stockage.

Proposition 3 *Dans une cellule robotisée sans espaces de stockage, on peut considérer sans perte de généralité que les durées de chargement et déchargement sont nulles.*

Démonstration. Pour les cellules robotisées RCm sans espaces de stockage, on peut inclure les durées de chargement et déchargement dans les durées de transport des pièces. En effet, pendant le transport d'une pièce entre deux machines, le chargement et le déchargement, les deux machines doivent être vides (hormis M_0 et M_{m+1} qui ont des durées de déchargement, et de chargement respectivement, nulles d'après la Proposition 2). Ainsi, on peut considérer que le robot, la pièces et les deux machines sont nécessaires au transport de la pièce comme à son chargement et à son déchargement. On peut donc inclure les durées de chargement et déchargement des pièces dans les durées de transport car ces trois opérations nécessitent les mêmes ressources. □

Dans certains cas particuliers, un espace de stockage de capacité non nulle à la sortie des machines est suffisant pour montrer que les durées de chargement et de déchargement peuvent être considérées comme nulles. Considérons un flowshop robotisé à m machines et les activités du robot $(A_0,)A_1, A_2, \ldots, A_m(, A_{m+1})$ comme définies à la Section 1.1. Un cycle de production dans lequel k pièces entrent dans la cellule et la quittent est une séquence de ces activités $(A_0,)A_1, A_2, \ldots, A_m(, A_{m+1})$ dans laquelle chacune des activités est répétée exactement k fois. On appelle un tel cycle de production un k-*cycle*.

La proposition suivante affirme que si les pièces sont identiques, un 1-cycle est optimal. Un 1-cycle n'utilisant qu'au plus une unité de la capacité des buffers de sortie des machines, les problèmes de cellules robotisées avec buffer de sortie de capacité non nulle seront équivalents aux mêmes problèmes avec une capacité infinie pour tous les buffers de la cellule.

Le lemme suivant a été démontré dans [40] pour des cellules robotisées avec buffers de sortie de capacités non nulles. Nous l'étendons aux flowshops avec robot.

Lemme 1 *Pour un flowshop robotisé, si les pièces sont identiques, la durée du cycle $T(C_k)$, de tout k-cycle C_k est bornée par*

$$T(C_k) \geq k \max \left\{ \sum_{i=0}^{m} (\delta_i + \delta_i^e) + \sum_{i=0}^{m} \epsilon_i^u + \sum_{i=1}^{m+1} \epsilon_i^l \ , \quad \max_h \{\epsilon_h'^l + p_h + \epsilon_h'^u\} \right\}$$

avec les notations suivantes :
- $\delta_{i,j} = \delta_i$, $\delta_{i,j}^e = \delta_i^e$, $\epsilon_{i,j}^u = \epsilon_i^u$, $\epsilon_{i,j}^l = \epsilon_i^l$, $p_{i,j} = p_i$, *en omettant l'indice j pour le cas de pièces identiques ;*
- $\delta_0 = \delta_m = 0$, $\delta_0^e = \delta_m^e = 0$, $\epsilon_0^u = \epsilon_{m-1}^u = 0$, $\epsilon_1^l = \epsilon_m^l = 0$ *dans le cas de flowshops avec robot ;*
- *Pour tout i et tout j,*

$$\epsilon_i'^l = \begin{cases} 0 & \text{si le buffer d'entrée de } M_i \text{ a une capacité non nulle} \\ \epsilon_{i,j}^l & \text{sinon} \end{cases}$$
$$\epsilon_i'^u = \begin{cases} 0 & \text{si le buffer de sortie de } M_i \text{ a une capacité non nulle} \\ \epsilon_{i,j}^u & \text{sinon} \end{cases}$$

Démonstration. Soit C_k un k-cycle. Le robot charge k fois la machine M_m ou son buffer d'entrée et retourne à l'état initial du cycle. Ainsi, il effectue $2k$ fois le transport d'une pièce entre M_{m-1} et M_m, ce qui implique une durée de transport de $k(\delta_m + \delta_m^e)$. Pour la même raison, chaque distance inter-machine est parcourue $2k$ fois. Ainsi, la durée totale des déplacements du robot pendant un k-cycle est supérieure à $k \sum(\delta_i + \delta_i^e)$. Puisque chaque machine (ou ses buffers) est chargée et déchargée par le robot exactement k fois, on obtient :

$$T(C_k) \geq k \left(\sum_{i=1}^{m} (\delta_i + \delta_i^e) + \sum_{i=0}^{m} \epsilon_i^u + \sum_{i=1}^{m+1} \epsilon_i^l \right)$$

De plus, pendant un k-cycle, chaque machine M_h produit complètement k pièces. Ainsi, $T(C_k) \geq k \left(\max_h \{\epsilon_h'^l + p_h + \epsilon_h'^u)\right\}$. □

La proposition suivante a été démontrée dans [40] pour des cellules robotisées avec buffers de capacité unitaire pouvant servir de buffer d'entrée et de sortie.

Proposition 4 *Dans un flowshop robotisé avec buffer de sortie sur chaque machine de capacité supérieure ou égale à 1, le taux de production maximal lorsque les pièces sont identiques est obtenu par le 1-cycle $Id = ((A_0,)A_1, A_2, \ldots, A_m(, A_{m+1}))$.*

Démonstration. Pour démontrer cette propriété, on prouve que le 1-cycle Id atteint la borne donnée par le Lemme 1.

Le calcul de la durée du cycle Id a été faite dans [40, 21] pour les cellules robotisées. On ne donne ici que l'idée de la preuve.

Le robot, après avoir quitté la machine M_i avec une pièce, devra en déposer et en prendre une sur chaque autre machine avant de revenir en M_i. Ainsi, soit l'usinage de la pièce présente sur M_i est toujours terminé lorsque le robot arrive en M_i pour décharger la pièce et $T(Id) = \sum(\delta_i + \delta_i^e) + \sum \epsilon_i^u + \sum \epsilon_i^l$, soit le robot doit attendre que l'usinage de

la pièce présente sur M_i s'achève et dans ce cas, d'après nos hypothèses, le robot sera stationné à la machine précédente pendant le temps d'attente. Le cycle est alors borné par $\max_h\{\epsilon'^l_h + p_h + \epsilon'^u_h\}$. □

Ainsi, pour des pièces identiques, si les buffers de sortie des machines sont de capacités non nulles, alors on peut considérer qu'elles sont infinies et que les capacités des buffers d'entrée sont elles aussi infinies. Avec la Proposition 2 on obtient :

Proposition 5 *Dans les flowshops avec robot, si les pièces sont identiques et si chaque machine a un buffer de sortie de capacité non nulle, alors on peut considérer sans perte de généralité que les durées de chargement et déchargement sont nulles.*

Il est donc intéressant de commencer l'étude d'un problème de flowshop robotisé en vérifiant la pertinence des durées de chargement et déchargement.

2.2 Flowshop avec robot à deux machines

Dans cette section, nous nous intéressons aux flowshops avec robot à deux machines (voir Figure 2.7).

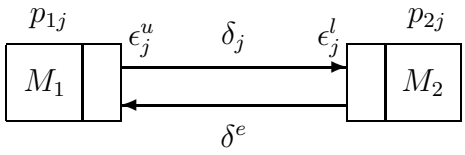


FIG. 2.7 – Notations pour le flowshop avec robot à 2 machines, F_2, R_1

Les notations pour ce problème sont simplifiées de la façon suivante :

- n, le nombre de pièces à produire ;
- p_{ij}, la durée de production de la pièce j sur la machine M_i, $i = 1, 2$;
- δ_j, le temps mis par le robot pour transporter la pièce j entre les machines M_1 et M_2 ;
- δ^e, la durée de déplacement à vide du robot entre M_2 et M_1 ;
- ϵ^u_j, la durée de déchargement depuis la machine M_1 de la pièce j ;
- ϵ^l_j, la durée de chargement de la pièce j sur la machine M_2.

Kise [54] a étudié le problème de flowshop avec robot à deux machines possédant chacune des buffers infinis, sans durées de chargement et déchargement (ce qui n'est pas une restriction, comme le montre la Proposition 2). Il a montré que le problème était $\mathcal{NP}$-Difficile même si les durées de transport des pièces sont toutes identiques et les mouvements à vide du robot de durées nulles. Il compare $F3||Cmax$ à un flowshop avec robot à deux machines tel que les déplacements à vide du robot, les chargements et les déchargements sont de durées nulles. Le robot représente alors la seconde des trois machines de $F3||Cmax$. Il déduit de cette comparaison que $F3|p_{2,j} = p|Cmax$ est aussi $\mathcal{NP}$-Difficile. Hurink et Knust [50] ont montré que si les pièces avaient des durées de production identiques $p_{i,j} = p$, mais des durées de transport différentes, le problème est aussi $\mathcal{NP}$-Difficile. Dans le cas sans attente, le problème $F3|\text{no-wait}|Cmax$ est

$\mathcal{NP}$-Difficile [76], ce qui implique que $F2, R1|\text{no-wait}|Cmax$ est aussi $\mathcal{NP}$-Difficile. Chen et al. [32] montrent que $F3|\text{no-wait}, p_{2,j} = p_2|Cmax$ est polynomial, ce dont on peut déduire que $F2, R1|\text{no-wait}, \delta_j = \delta, \delta_e = 0, \epsilon_j^u = 0, \epsilon_j^l = 0|Cmax$ est polynomial. Cependant, ce problème n'est pas équivalent à $F2, R1|\text{no-wait}, \delta_j = \delta, \delta_e = 0, \epsilon_j^u = \epsilon^u, \epsilon_j^l = \epsilon^l|Cmax$ puisque les durées de chargement et déchargement ne peuvent être incluses ni dans les durées de transport, ni dans les durées d'usinage comme le montrent les deux exemples suivants.

Exemple 3. Considérons l'instance $\delta = 2, \epsilon^u = 2, \epsilon^l = 1, p_{1,1} = p_{2,1} = 2, p_{1,2} = 5, p_{2,2} = 4$. Si les durées de chargement et le déchargement ne sont pas incluses dans les durées de transport, la seule séquence optimale est $(2, 1)$. Si elles sont incluses dans les durées de transport (et dans ce cas $\delta = 5$, $\epsilon^u = \epsilon^l = 0$), la seule séquence optimale est $(1, 2)$ (voir Figures 2.8 et 2.9).

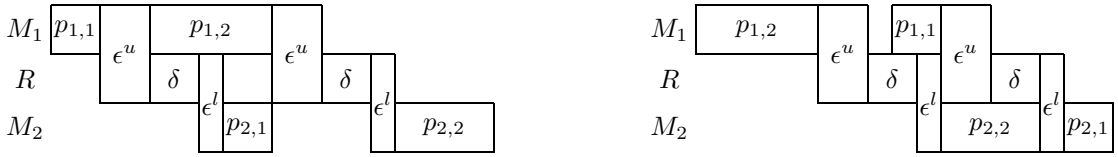


FIG. 2.8 – Durées de chargement et déchargement non incluses dans les durées de transport : séquences $(1, 2)$ avec $Cmax = 18$ et $(2, 1)$ avec $Cmax = 17$

FIG. 2.9 – Durées de chargement et déchargement incluses dans les durées de transport : séquences $(1, 2)$ avec $Cmax = 16$ et $(2, 1)$ avec $Cmax = 17$

Exemple 4. Considérons l'instance $\delta = 3, \epsilon^u = 2, \epsilon^l = 1, p_{1,1} = p_{2,1} = 2, p_{1,2} = 4, p_{2,2} = 3$. Si les durées de chargement et le déchargement ne sont pas incluses dans les durées d'usinage, la seule séquence optimale est $(2, 1)$. Si elles sont incluses dans les durées d'usinage (et dans ce cas $\delta = 3$, $\epsilon^u = \epsilon^l = 0$, $p_{1,1} = p_{2,2} = 4$, $p_{1,2} = 6$, $p_{2,1} = 3$) la seule séquence optimale est $(1, 2)$ (voir Figure 2.10 et 2.11).

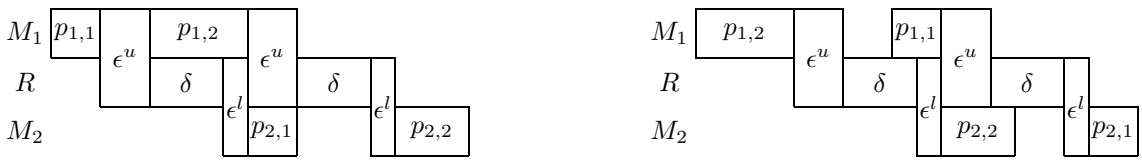


FIG. 2.10 – Durées de chargement et déchargement non incluses dans les durées d'usinage : séquences $(1, 2)$ avec $Cmax = 17$ et $(2, 1)$ avec $Cmax = 18$

FIG. 2.11 – Durées de chargement et déchargement incluses dans les durées d'usinage : séquences $(1,2)$ avec $Cmax = 17$ et $(2,1)$ avec $Cmax = 16$

Le problème de cellules robotisées à deux machines $RC2$ est polynomial dans le cas sans attente si $\delta_{i,j} = \delta_i$ (Agnetis [4] et Lei et Liu [63]). Hall et al. [45] on trouvé un algorithme polynomial pour le cas sans buffers avec attente illimitée quand les durées de chargement et déchargement sont indépendantes de la pièce. Comme le montre la Proposition 3, il est équivalent de considérer des durées de chargement et déchargement nulles dans ce cas et ce résultat prouve ainsi que le problème est polynomial quelle que soit la contrainte sur les durées de transport.

La Table 2.1 résume les résultats de complexité pour les problèmes de flowshop à deux machines.

F2,R1	buffers illimités	$\delta_j = \delta$	$\mathcal{NP}$-Difficile [54, 50]
	buffers illimités	$p_{i,j} = p$	$\mathcal{NP}$-Difficile [50]
	sans attente		$\mathcal{NP}$-Difficile [76]
	sans attente	$\delta_j = \delta$, $\epsilon^u_j = \epsilon^u$, $\epsilon^l_j = \epsilon^l$	?
	sans attente	$\delta_j = \delta$, $\delta_e = 0$, $\epsilon^u_j = 0$, $\epsilon^l_j = 0$	Polynomial [32]
RC2	sans attente	$\delta_{i,j} = \delta$	Polynomial [4, 63]
	sans attente	$\delta_{i,j} = \delta_i$, $\epsilon^u_{ij} = \epsilon^u_i$, $\epsilon^l_{i,j} = \epsilon^l_i$	Polynomial [45]

TAB. 2.1 – Résultats de complexité pour les flowshop robotisés à 2-machine

Dans la section suivante, nous considérons le problème $F2, R1$ sans espaces de stockage, avec attente illimitée ou sans attente, avec des temps de transport identiques. Au regard de ces résultats de complexité, le problème pourrait être aussi bien difficile comme $F2, R1|\text{no-wait}|Cmax$ et $F2, R1|\text{unlimited buffer}, \delta_{i,j} = \delta|Cmax$ ou polynomial comme $RC2|\text{no-wait}, \delta_{i,j} = \delta|Cmax$. Nous prouvons que ce problème est en fait polynomial.

Flowshop avec temps de transports identiques

Dans cette section, nous considérons un problème de flowshop avec robot sans espaces de stockage. Nous supposons que les durées de transport et les durées de chargement et déchargement sont identiques pour toutes les pièces et nous montrons que, quelle que soit la contrainte sur les temps d'attente (sans attente ou avec attente illimitée), le problème est polynomial. Les problèmes avec serveur sont des cas particuliers des problèmes avec

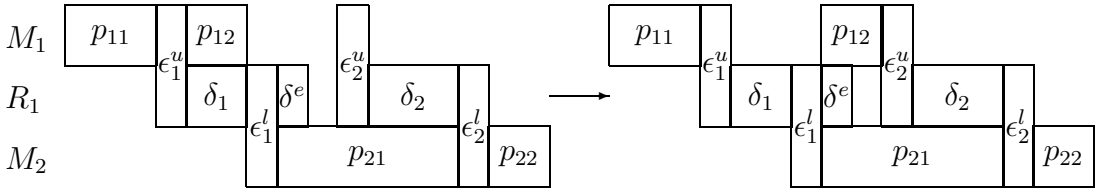


FIG. 2.12 – F_2, R_1 sans espaces de stockage

robot sans espaces de stockage aux machines. On peut donc en déduire le même résultat pour les problèmes avec serveur correspondants.

Nous montrons d'abord les deux lemmes suivants.

Lemme 2 *Un problème de flowshop avec robot à deux machines, sans espaces de stockage et avec des temps d'attente bornés ou non bornés sur M_1 ou M_2 est équivalent au même problème avec une contrainte sans attente (*no-wait*).*

Démonstration. Puisque la machine M_2 n'a pas de buffer d'entrée, les tâches y sont usinées dès qu'elles sont chargées sur la machine. Ainsi, si des temps d'attente existent, c'est uniquement sur la machine M_1.

Cependant, pendant cette durée d'attente, la pièce occupe la machine qui ne peut effectuer aucun autre usinage. Ainsi, l'instant de départ de l'usinage de la pièce sur M_1 peut être repoussé pour que la contrainte sans attente soit respectée sans modifier le makespan. □

On peut remarquer que nous travaillons sous l'hypothèse que la pièce ne peut jamais attendre sur le robot. C'est ce qui différencie les configurations $F3$ et $F2, R1$ avec durée de retour à vide du robot nulle. Si cette contrainte est relaxée, les deux problèmes sont identiques et le problème sans buffers avec attente illimitée est différent du problème sans attente comme le montre l'exemple ci-dessous.

Exemple 5. Considérons l'instance suivante de la configuration $F3$:

pièce j	$p_{1,j}$	$p_{2,j}$	$p_{3,j}$
1	2	2	5
2	1	4	1
3	5	1	4

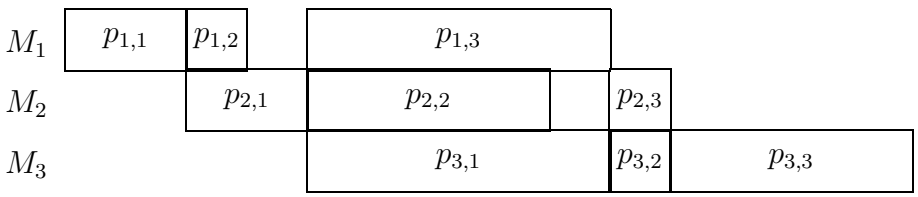


FIG. 2.13 – Séquence $(1, 2, 3)$, sans espaces de stockage, attente illimitée, $Cmax = 14$

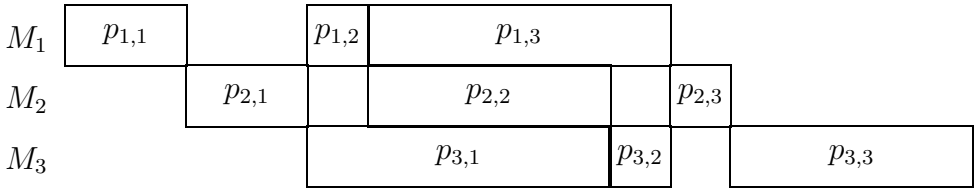


FIG. 2.14 – Séquence $(1, 2, 3)$, sans attente, $Cmax = 15$

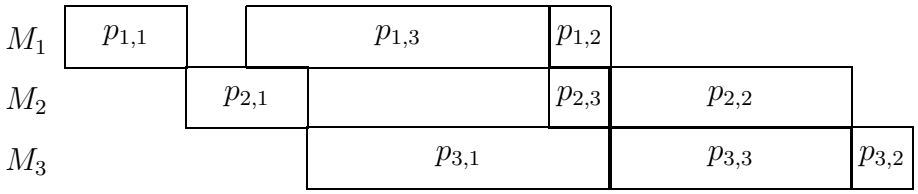


FIG. 2.15 – Séquence $(1, 3, 2)$, sans attente, $Cmax = 14$

La séquence $(1, 2, 3)$ est optimale dans le cas sans buffers avec un makespan de 14 (voir Figure 2.13) alors que cette solution, qui a un makespan de 15 dans le cas sans attente, n'est pas optimale (voir Figure 2.14) : la solution $(1, 3, 2)$ a un makespan de 14 (voir Figure 2.15).

Le lemme suivant est une extension d'une remarque de Hurink et Knust [50].

Lemme 3 *Le problème $F2, R1|$no-wait$|Cmax$ peut être réduit à un problème pour lequel la durée des mouvements à vide du robot entre M_2 et M_1 est nulle ($\delta^e = 0$).*

Démonstration. Considérons une instance
$\mathcal{I} = (n, p_{1,1}, p_{1,2}, \ldots, p_{1,n}, \delta_1, \delta_2, \ldots, \delta_n, \delta^e, p_{2,1}, p_{2,1}, \ldots, p_{2,n}, \epsilon^u_1, \epsilon^u_2, \ldots, \epsilon^u_n, \epsilon^l_1, \epsilon^l_2, \ldots, \epsilon^l_n)$
du problème $F2, R1|$no-wait$|Cmax$.
Nous construisons l'instance
$\mathcal{I}' = (n, p_{1,1}, p_{1,2}, \ldots, p_{1,n}, \delta'_1, \delta'_2, \ldots, \delta'_n, \delta^{e\prime} = 0, p_{2,1}, p_{2,1}, \ldots, p_{2,n}, \epsilon^u_1, \epsilon^u_2, \ldots, \epsilon^u_n, \epsilon^l_1, \epsilon^l_2, \ldots, \epsilon^l_n)$
en ajoutant δ^e à toutes les durées de transport, c'est-à-dire, $\delta'_j = \delta_j + \delta^e$, pour tout $j = 1, 2, \ldots, n$ (voir Figure 2.16). On peut remarquer qu'on obtient l'instance $\mathcal{I}'$ à partir de l'instance $\mathcal{I}$ en temps polynomial.

Chaque ordonnancement réalisable pour $\mathcal{I}$ peut être transformé en un ordonnancement réalisable pour $\mathcal{I}'$: il suffit de décaler l'ordonnancement sur M_2 de δ^e unités sur la droite, ce qui modifie le makespan seulement par une constante.

Un ordonnancement est donc optimal pour $\mathcal{I}$ si et seulement si il est optimal pour $\mathcal{I}'$. Ainsi, le problème $F2, R1|$no-wait$|Cmax$ peut être réduit au problème $F2, R1|$no-wait$, \delta^e = 0|Cmax$ en temps polynomial. □

On peut remarquer que la réduction réciproque est triviale et que les problèmes $F2, R1|$no-wait$|Cmax$ et $F2, R1|$no-wait$, \delta^e = 0|Cmax$ sont équivalents.

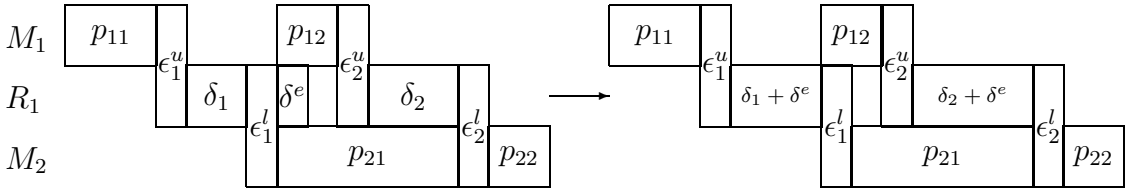


FIG. 2.16 – F_2, R_1 sans buffers, suppression des durées de trajet à vide

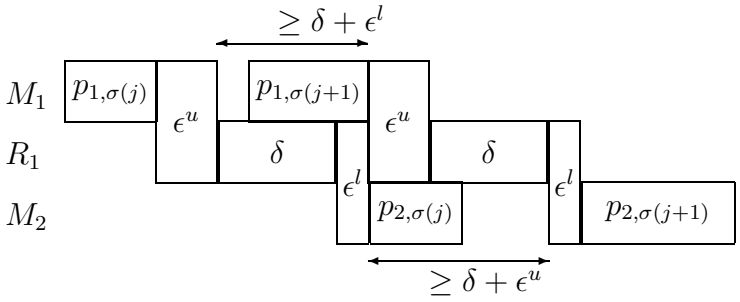


FIG. 2.17 – Extension des durées de production des pièces sur les machines M_1 et M_2

Ces deux lemmes impliquent que la complexité du problème de flowshop avec robot à 2 machines est identique que le problème soit sans attente ou sans buffers et que les durées de trajet à vide du robot soient nulles ou non. Le théorème suivant donne la complexité de ces problèmes.

Théorème 3 *Le problème $F_2, R_1|$no-wait, $\delta_j = \delta$, $\epsilon_j^u = \epsilon^u$, $\epsilon_j^l = \epsilon^l|Cmax$ est polynomial.*

Démonstration. Si le nombre de pièces est 1, alors la résolution du problème est triviale. Nous considérons par la suite que $n \geq 2$.

Le Lemme 3 indique que nous pouvons considérer que $\delta^e = 0$.

Nous prouvons tout d'abord qu'étant donnée une instance telle que $n \geq 2$ et deux pièces distinctes k et l, la meilleure permutation des pièces qui commence par k et finit par l peut être calculée en temps polynomial.

Soient $\mathcal{I} = (n, p_{1,1}, p_{1,2}, \ldots, p_{1,n}, \delta, p_{2,1}, p_{2,1}, \ldots, p_{2,n}, \epsilon^u, \epsilon^l)$ une instance telle que $n \geq 2$ et k et l deux pièces distinctes.

Considérons une permutation σ des tâches qui commence par k et finit par l et un indice $j \in \{1, 2, \ldots, n\}$. Le déchargement de la $(j+1)^{ième}$ pièce de σ, $\sigma(j+1)$, de la machine M_1 est effectué au moins $\delta + \epsilon^l$ unités de temps après celui de la pièce $\sigma(j)$. Cela implique que nous pouvons allonger la durée de production $p_{1,\sigma(j+1)}$ à $\delta + \epsilon^l$, si elle est inférieure à cette valeur, sans modifier le makespan (voir Figure 2.17). Comme $j \in \{1, 2, \ldots, n\}$, nous pouvons réaliser cette opération pour toutes les pièces sauf la pièce $\sigma(1) = k$.

De la même façon, nous pouvons allonger les durées de production $p_{2,\sigma(j)}$ à $\delta + \epsilon^u$ si elles sont inférieures à cette valeur pour toutes les pièces sauf $\sigma(n) = l$.

Notons $p'_{i,j}$ les durées de production ainsi obtenues. Nous avons
- $p'_{1,k} = p_{1,k}$;
- $p'_{1,j} = \max\{p_{1,j}, \delta + \epsilon^l\}$, pour tout $j \neq k$;
- $p'_{2,j} = \max\{p_{2,j}, \delta + \epsilon^u\}$, pour tout $j \neq l$;
- $p'_{2,l} = p_{2,l}$.

Afin de trouver la meilleure permutation qui commence par k et finit par l, nous essayons de minimiser les temps morts sur la machine M_2, c'est-à-dire la durée pendant laquelle elle n'est pas utilisée. Considérons deux pièces distinctes q et r et calculons le temps mort $Idle(q,r)$ introduit sur M_2 en plaçant r immédiatement après q.

Si $p'_{2,q} + \epsilon^l > p'_{1,r} + \epsilon^u$, on n'introduit pas de temps morts sur M_2 (voir Figure 2.18).
Dans le cas où $p'_{2,q} + \epsilon^l \leq p'_{1,r} + \epsilon^u$, on introduit un temps mort sur M_2 de durée $p'_{1,r} + \epsilon^u - (p'_{2,q} + \epsilon^l)$ (voir Figure 2.19).

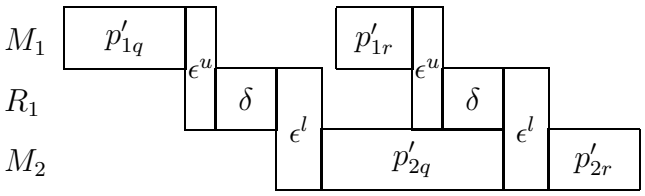


FIG. 2.18 – $Idle(q,r)$: cas où $p'_{2,q} + \epsilon^l > p'_{1,r} + \epsilon^u$

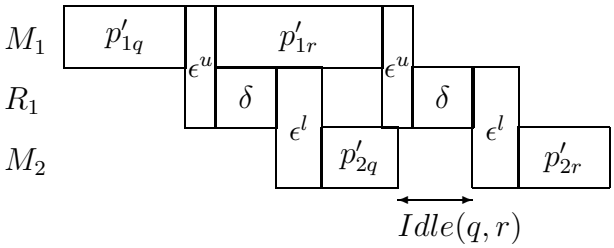


FIG. 2.19 – $Idle(q,r)$: cas où $p'_{2,q} + \epsilon^l \leq p'_{1,r} + \epsilon^u$

Ainsi,

$$Idle(q,r) = \begin{cases} p'_{1,r} + \epsilon^u - (p'_{2,q} + \epsilon^l) & \text{si } p'_{2,q} + \epsilon^l \leq p'_{1,r} + \epsilon^u \\ 0 & \text{si } p'_{2,q} + \epsilon^l > p'_{1,r} + \epsilon^u \end{cases}$$

Nous considérons le problème du voyageur de commerce (TRAVELING SALESMAN PROBLEM) suivant. Chaque pièce j différente de k et l est représentée par un sommet noté j. Un sommet kl représente à la fois k et l. Le graphe est complet et orienté. Le poids $w(q,r)$ de l'arc (q,r) est la durée d'inactivité introduite sur M_2 en plaçant la pièce r immédiatement après la pièce q dans la permutation. Nous avons :
- $w(kl,r) = Idle(k,r)$;
- $w(q,r) = Idle(q,r)$ pour $q \neq kl$ et $r \neq kl$;
- $w(q,kl) = Idle(q,l)$.

Remarquons que la construction de ce graphe se fait en temps polynomial.

Un tour du voyageur de commerce représente une permutation des pièces qui commence par k et finit par l et dont le poids est la somme des durées des temps morts introduits sur M_2 par ce choix de permutation. Ainsi, un tour minimal dans ce graphe correspond à une permutation des pièces qui commence par k et finit par l qui minimise les temps morts sur M_2.

Pour tout arc (q, r) du graphe, on peut écrire

$$w(q,r) = \begin{cases} \int_{\tilde{p}_{2,r}}^{\tilde{p}_{1,q}} f(x)dx, & \text{si } \tilde{p}_{1,q} \geq \tilde{p}_{2,r} \\ \int_{\tilde{p}_{1,q}}^{\tilde{p}_{2,r}} g(x)dx, & \text{si } \tilde{p}_{1,q} < \tilde{p}_{2,r} \end{cases} \tag{2.1}$$

avec

$$\begin{array}{l} \tilde{p}_{1,kl} = p'_{1,k} + \epsilon^u \\ \tilde{p}_{1,q} = p'_{1,q} + \epsilon^u, \text{ pour tout } q \neq kl \\ \tilde{p}_{2,kl} = p'_{2,l} + \epsilon^l \\ \tilde{p}_{2,q} = p'_{2,q} + \epsilon^l, \text{ pour tout } q \neq kl \\ f(x) = 1, \text{ pour tout } x \\ g(x) = 0, \text{ pour tout } x \end{array}$$

Gilmore et Gomory ont prouvé que lorsque les poids des arcs étaient de la forme de l'équation (2.1) avec f et g intégrables, le problème du voyageur de commerce pouvait se résoudre en temps polynomial.

Nous pouvons donc trouver la meilleure permutation qui commence par k et finit par l en temps polynomial.

Nous proposons l'algorithme polynomial suivant pour notre problème : pour chaque paire de pièces distinctes k et l, on utilise l'algorithme de Gilmore et Gomory pour obtenir la meilleure permutation qui commence par k et finit par l. Notons $Cmax(k, l)$ le makespan de cette permutation. Le minimum de tous les $Cmax(k, l)$ sur l'ensemble des couples (k, l) avec $k \neq l$ est le makespan optimal (et nous donne la meilleure permutation des pièces).

Le nombre de couples (k, l) avec $k \neq l$ étant polynomial, la complexité de cet algorithme est polynomiale. □

Donnons maintenant une première conséquence immédiate du Théorème 3.

Corollaire 1 *Le problème $F2, R1|$no-buffer, $\delta_j = \delta$, $\epsilon_j^u = \epsilon^u$, $\epsilon_j^l = \epsilon^l|Cmax$ est polynomial.*

Démonstration. Directement du Théorème 3 et du Lemme 2. □

Dans [32], les auteurs prouvent le résultat suivant qui peut à présent être présenté comme une conséquence du Théorème 3.

Corollaire 2 *Le problème $F3|$no-wait, $p_{2,j} = p_2|Cmax$ est polynomial.*

Démonstration. Le problème $F3|$no-wait, $p_{2,j} = p_2|Cmax$ est un cas particulier du problème $F2, R1|$no-wait, $\delta_j = \delta$, $\epsilon_j^u = \epsilon^u$, $\epsilon_j^l = \epsilon^l, \delta^e = 0|Cmax$ avec $\epsilon_j^u = 0$ et $\epsilon_j^l = 0$. Le robot est alors la seconde machine et M_2 la troisième du problème à 3 machines. □

Comme nous l'avons déjà fait remarqué, il est possible de déduire du Théorème 3 et du Corollaire 1 que le problème avec serveur correspondant est polynomial.

2.3 Conclusion

Nous avons dans ce chapitre souligné les liens et les différences entre deux configurations classiques de flowshop robotisés étudiés jusqu'ici indépendamment. Nous avons proposé une interrogation sur la pertinence des modèles de cellules avec durées de chargement et déchargement en fonction des espaces de stockage disponibles. Enfin, nous avons prouvé que plusieurs problèmes de flowshop avec robot à deux machine étaient équivalents à $F2, R1|$no-wait, $\delta_j = \delta$, $\epsilon_j^u = \epsilon^u$, $\epsilon_j^l = \epsilon^l, \delta^e = 0|Cmax$, que nous montrons être polynomial.

La complexité de ce problème pour trois et quatre machines reste donc ouverte et pourra faire l'objet de futures recherches.

Chapitre 3

Problème industriel de planification d'expériences

Ce thème de recherche est issu d'une problématique industrielle de planification d'expériences proposée par l'Institut Français du Pétrole (IFP). Un changement de processus industriel a conduit l'IFP à souhaiter acquérir un logiciel capable d'ordonnancer des expériences en grand nombre.

L'objectif de l'IFP est de déterminer les conditions optimales de synthèse d'un certain nombre de produits chimiques. Leur protocole expérimental consiste à effectuer pour chaque produit (chaque tâche) un ensemble d'expériences (d'opérations) pouvant donner lieu à des résultats intéressants. On définit pour chaque produit un ensemble de *mélanges* dans lesquels il intervient. Afin de déterminer la durée optimale de cuisson pour chaque mélange, on conduit une première série d'expériences (notées *expériences en phase I*). On ignore alors combien d'expériences seront nécessaires pour obtenir cette durée optimale, mais cette série ne se termine qu'une fois qu'elle est obtenue. Ensuite, différentes conditions opératoires sont testées avec cette durée (*phase II*) pour améliorer la qualité des résultats de chaque mélange. Ceci correspond à une deuxième série d'expériences.

On dispose d'un ensemble de barres pour réaliser les expériences. Dans chaque barre, on peut effectuer simultanément plusieurs expériences de même durée et de même température. La durée des expériences est comprise entre 1 et 21 jours. Le nombre total d'expériences à réaliser est de plusieurs milliers.

L'IFP souhaitait que nous réalisions un logiciel capable de créer un planning de l'ensemble de ces expériences et qui optimise les deux critères suivants :
- maximiser le taux de remplissage des barres et finir l'ensemble des tâches le plus tôt possible ;
- finir régulièrement des tâches.

Le premier critère correspond à une logique de minimisation des coûts et de la durée totale de production, le second à un souhait des utilisateurs d'équilibrer le travail à fournir et d'obtenir de façon régulière des indications sur la qualité qu'il est possible d'obtenir à partir des produits testés.

Les actions à effectuer sur une barre, en plus des expérimentations, se décomposent en opérations élémentaires (lavage, remplissage, ...) ayant des durées fixes. Certaines de ces

opérations requièrent l'intervention d'un opérateur humain. La planification doit tenir compte des jours de congés des opérateurs et des éventuelles pannes et ruptures de stock.

3.1 Description du problème d'ordonnancement

Nous décrivons dans cette section de façon plus précise les composantes principales de ce problème d'ordonnancement, c'est-à-dire les ressources disponibles, les tâches à effectuer et l'objectif à atteindre.

3.1.1 Ressources

Les ressources sont composées d'opérateurs humains et de m barres pouvant contenir c expériences (capacité de la barre). Ces barres sont donc des machines de capacité non unitaire, comme pour un problème d'ordonnancement par batch [74]. Cependant, les opérations effectuées simultanément sur une barre doivent avoir exactement la même durée, ce qui n'est pas le cas classiquement pour les problèmes d'ordonnancement par batch. Dans ces derniers, il est possible de regrouper ensemble des opérations de durées différentes, la durée d'usinage du batch devenant alors celle de la plus longue opération. On parle donc pour le problème d'ordonnancement que nous abordons ici de *compatibilité en terme de batch* (*batch compatibility*) des opérations [16]. Elles sont *batch-compatibles* si elles ont la même durée exactement.

La séquence des opérations d'une barre est la suivante :

$$\text{remplissage} \Rightarrow \text{cuisson} \Rightarrow \text{filtration} \Rightarrow \text{lavage}$$

Les remplissages et filtrations de la barre correspondent respectivement à des set-ups et démontages non-séparables (l'opérateur humain pourrait alors être considéré comme un serveur ou un robot de durées de déplacement nulles), la cuisson à l'usinage et le lavage à un démontage séparable cette fois (l'opérateur humain pourrait pour cette opération être considéré comme un opérateur au sens classique). Chacune de ces opérations a une durée fixée et nécessite des ressources. À l'issue du lavage, la barre est de nouveau disponible pour un remplissage. Les spécificités des ressources sont les suivantes :

- Les durées de filtration, lavage et remplissage indiquent le nombre de jours ouvrés nécessaires. La préemption est autorisée pour ces trois opérations : elles peuvent s'interrompre avant un jour chômé et reprendre lorsque le congé est terminé.
- On ne doit pas finir ou commencer une cuisson pendant un jour chômé car cela requiert l'intervention d'un opérateur humain.

La nécessité de l'intervention d'un opérateur humain au début et à la fin de l'usinage est non classique mais peut être modélisée de la façon suivante. Chaque opérateur humain serait représenté par un robot de durées de déplacement nulles qui effectuerait un chargement et un déchargement de durées nulles. Ces chargements et déchargements seraient liées à l'usinage par une contrainte de type *no-wait*, c'est-à-dire que l'usinage doit commencer immédiatement après la fin du chargement et doit être immédiatement suivi du déchargement. Les absences des opérateurs, qui se répètent au moins tous les week-ends,

seraient alors modélisées par des périodes de maintenance pendant lesquelles les robots sont indisponibles.

3.1.2 Tâches

Pour chaque tâche, D sous-ensembles d'opérations sont définis. Pour chaque sous-ensemble d, s_d expériences sont testées. Une tâche correspondant à un produit, chaque sous-ensemble correspond à un mélange faisant intervenir ce produit. Pour chaque mélange, le déroulement des expérimentations comporte deux phases :

- phase I : on recherche la durée optimale de cuisson pour le mélange donné. Pour trouver cette durée, on parcourt un chemin dans un arbre orienté dont les nœuds représentent les durées possibles. Les arcs sont orientés de la racine vers les feuilles, on parle d'arbre de type *outtree*. Cet arbre est défini a priori par les chimistes en fonction de leurs connaissances et des résultats espérés pour ce mélange. Le choix du chemin est déterminé à chaque pas par le résultat de l'expérience précédente. Si la durée de cuisson est adéquate pour le mélange, il passe en phase II. Sinon, l'exploration de l'arbre continue.
- phase II : elle est composée de plusieurs expériences qui ont la durée optimale obtenue en phase I. On cherche à présent à comparer les différentes conditions expérimentales possibles.

On peut remarquer que pendant la phase I, les expériences d'un même mélange se suivent et ne peuvent pas être réalisées en parallèle. Par contre, lors de la phase II, les expériences qui sont de même durée peuvent être mises dans le même batch, ou effectuées en parallèle dans des batchs de même durée.

Le nombre et la durée des opérations en phase I n'étant pas connu à l'avance, le problème est de type online. Cependant, les durées possibles des prochaines opérations qui arriveront dans le système, ainsi que la prochaine date à laquelle des opérations seront introduites sont connues tout au cours de l'ordonnancement. On parle donc de problème *semi-online* ou *nearly-online* (voir Section 1.2.5).

On peut remarquer que le problème offline correspondant n'est pas comme classiquement un problème avec dates de disponibilité. Les opérations sont toutes disponibles au début de l'ordonnancement mais sont soumises à des contraintes de précédences de type outtree.

3.1.3 Objectif

L'objectif est double :

- Minimiser la durée globale nécessaire pour tester toutes les tâches ($Cmax$). D'autres tâches pouvant arriver "au fil de l'eau", ceci revient à maximiser la charge (c'est-à-dire le nombre d'expériences) par barre sur l'ensemble des cuissons réalisées pour utiliser au mieux cette ressource.
- Minimiser la somme des dates d'achèvement des tâches, $\sum C_i$.

Ceci peut se résumer par : "Terminer rapidement et régulièrement des tâches et maximiser l'occupation des barres".

3.2 Solution proposée

Deux problèmes classiques sont sous-jacents à notre problème : l'ordonnancement sur machines parallèles [64, 73] avec comme objectif la minimisation de la date d'achèvement de toutes les tâches ($Cmax$) et la somme d'achèvement des tâches ($\sum C_i$), avec des ressources additionnelles (par exemple, l'opérateur au début et à la fin des opérations). Pour ces types de problèmes, on utilise généralement des listes de priorité. Par exemple, pour le cas sans ressources, la règle SPT (Shortest Processing Time) est optimale pour le critère $\sum C_i$ [31] et la règle LPT (Longest Processing Time) a une performance garantie de $4/3$ pour le critère $Cmax$ [41] qui est d'ores et déjà $\mathcal{NP}$-Difficile. Dès que l'on rajoute des contraintes de groupement de tâches [27] ou de ressources additionnelles, ces problèmes sont généralement $\mathcal{NP}$-Difficiles. On ne peut donc pas espérer proposer des solutions optimales en un temps raisonnable (quelques minutes) en respectant les contraintes industrielles. De plus, un aspect important de cette application est que de nouvelles expériences peuvent être générées à la fin d'une opération. Leur nombre et leurs durées ne sont connus qu'une fois cette opération terminée. Il n'est donc pas possible de résoudre ce problème de façon optimale et nous avons choisi de développer un algorithme qui remet en cause la solution proposée à chaque événement pour tenir compte de cette caractéristique. Le logiciel a été décomposé en deux parties : une initialisation statique (lancement) et une partie de production dynamique pour la résolution du problème nearly-online.

3.2.1 Lancement

Le but de l'initialisation est de générer un nombre d'expériences suffisant pour favoriser le remplissage des barres. En effet, tant que les mélanges sont en phase I, on ne dispose que de peu d'expériences à réaliser. C'est le passage en phase II de ces mélanges qui permettra de générer un grand nombre d'expériences. La phase de lancement permet d'ordonnancer la plupart des expériences en phase I. En effet, les chimistes de l'IFP peuvent estimer à priori quelles seront les durées de synthèse des différents mélanges. Les arbres ont été choisis à partir de cette connaissance des produits chimiques pour minimiser le chemin qui sera parcouru et donc le nombre d'expériences en phase I pour chaque mélange. Le lancement crée donc un grand nombre d'expériences qui assure le remplissage des barres. Il permet de plus de mettre en route toutes les tâches, d'identifier éventuellement des tâches intéressantes ou qui finiront rapidement.

Une première expérimentation avait été faite avec un logiciel de programmation par contraintes. Après quelques expérimentations, il s'est avéré que le temps d'obtention d'une solution optimale était trop long (supérieur à plusieurs heures, probablement à cause des nombreuses symétries du problème). Nous avons alors utilisé un logiciel de Programmation Linéaire en Nombres Entiers (Cplex interfacé sous OPL Studio). La fonction à minimiser suivante a été retenue :

$$\alpha Cmax + \beta \sum R_i$$

où $\sum R_i$ est la somme des durées d'utilisation des ressources. Ce deuxième critère revient à minimiser le temps pendant lequel les ressources ne sont pas utilisées.

Les variables de décision sont les dates de lancement des barres. On peut alors pré-calculer la date d'achèvement de chacune des opérations liées à la barre en tenant compte des spécificités du problème (jours non travaillés, par exemple). La modélisation est basée sur une discrétisation du temps (par demi-journée) et les contraintes décrivent le flux des opérations en respectant les règles d'utilisation des ressources (contraintes de flot). Des contraintes ont été rajoutées pour casser la symétrie entre les barres de même durée, ainsi que des contraintes disjonctives classiques en ordonnancement pour générer des coupes.

Cette solution a été jugée satisfaisante aussi bien pour la qualité des solutions obtenues que pour le temps nécessaire pour les obtenir (quelques minutes). Des expérimentations numériques ont permis de déterminer les valeurs intéressantes de α et β pour le problème industriel. Ces valeurs résultent évidemment d'un compromis entre les deux critères, compromis qui doit correspondre aux préférences de l'industriel. Ces préférences ont pu être exprimées en considérant les ordonnancements obtenus pour différentes pondérations sur des jeux caractéristiques de données.

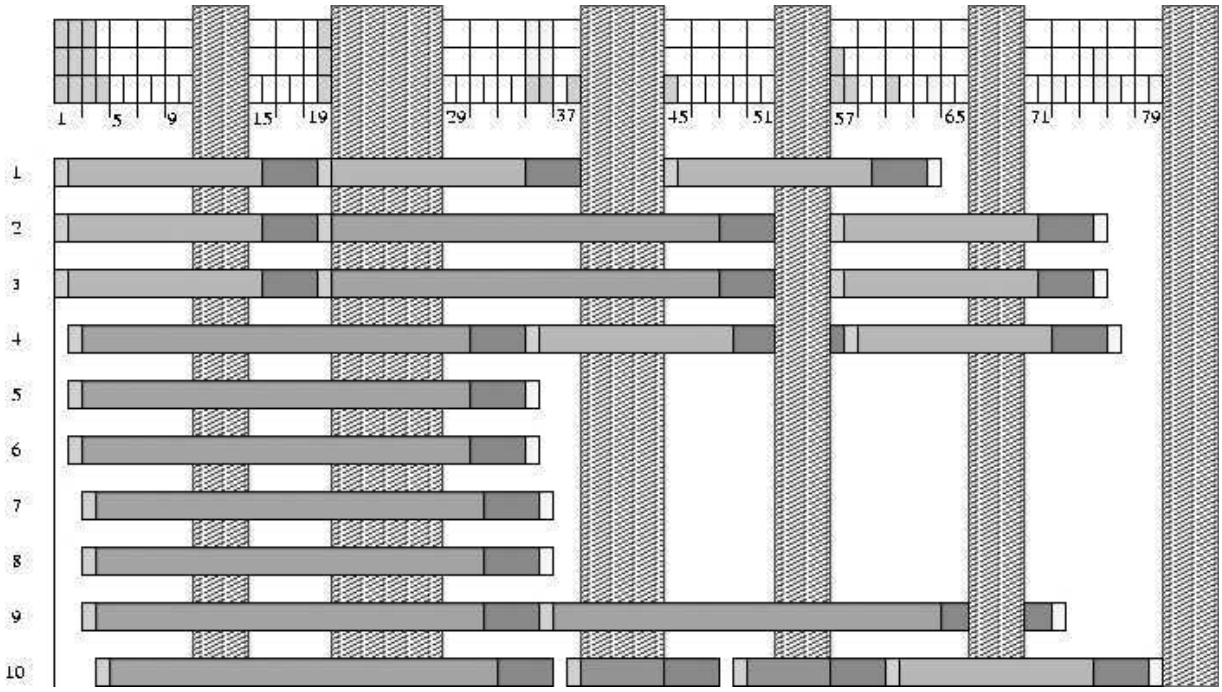


FIG. 3.1 – Ordonnancement optimal pour $\alpha = 0$ et $\beta = 1$

À titre d'exemple, la Figure 3.1 représente la phase d'initialisation pour un jeu de données comportant 10 barres avec des expériences de durées égales à 7 jours, 10 barres à 14 jours et 2 barres avec des expériences à 3 jours. Les pondérations choisies sont $\alpha = 0$ et $\beta = 1$, c'est-à-dire la planification cherche uniquement à optimiser l'utilisation des ressources.

La planification commence un lundi. Sur l'horizon de temps considéré, en plus des week-ends, plusieurs jours fériés doivent être pris en compte, interdisant le début ou la fin du traitement d'une expérimentation pendant cette période (zones verticales hachurées). Au-dessus du diagramme de Gantt, nous avons représenté (par demi-journée) l'occupation des ressources nécessaires à l'une des opérations élémentaires (remplissage) : au plus 3 remplissages des barres peuvent se faire simultanément. La planification se termine au bout de 80 demi-journées. La fin moyenne d'utilisation des ressources est de $58,8$ demi-

journées.

Le diagramme de Gantt de la Figure 3.2 montre la planification obtenue avec les pondérations $(1, 5)$. La date d'achèvement est alors de 72 demi-journées, pour une date de fin moyenne d'utilisation des ressources de 60 demi-journées.

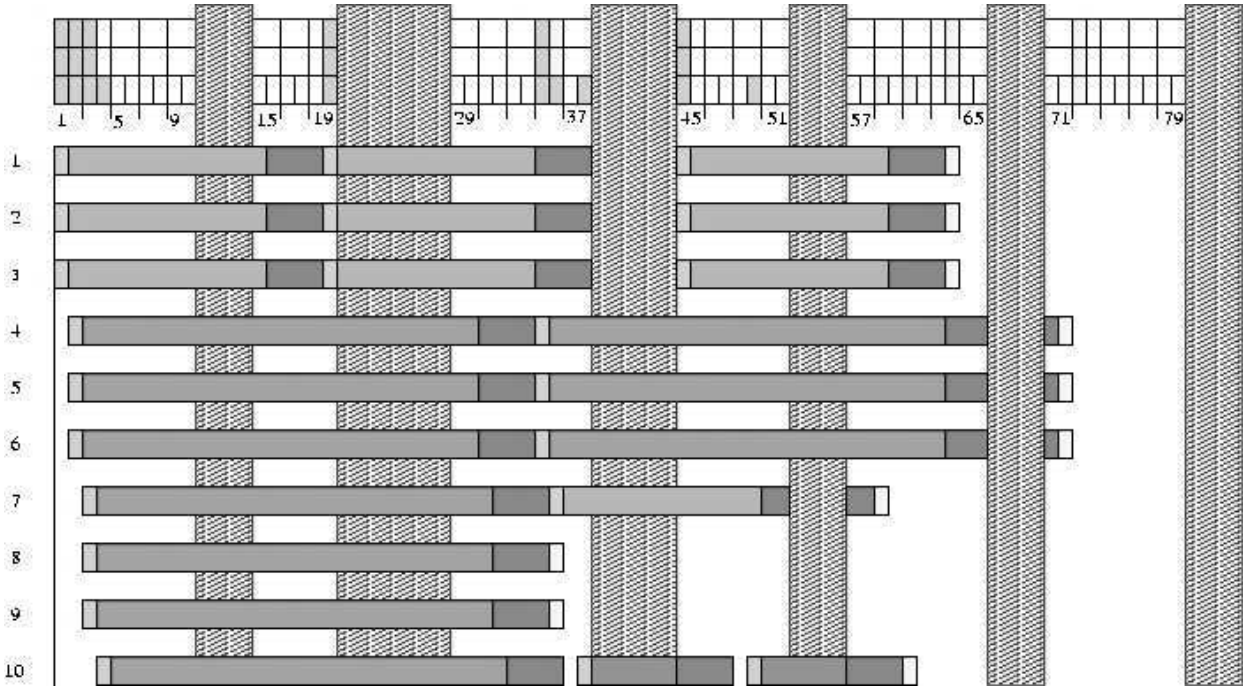


FIG. 3.2 – Ordonnancement optimal pour $\alpha = 1$ et $\beta = 5$

3.2.2 Production

Après l'initialisation, lorsque toutes les tâches sont correctement lancées et qu'un nombre important d'expériences est disponible, on passe en phase de production. Pour cette étape, nous avons opté pour un algorithme de liste. Le principe de l'algorithme est de répartir les ressources (nombre de barres) de manière dynamique entre deux critères.
– une partie des barres est allouée aux tâches "en retard" (avancement de la phase I) ;
– le reste des barres est alloué aux tâches les plus en avance (achèvement de la phase II).
La première partie assure que suffisamment d'expériences sont générées au fur et à mesure : en effectuant des expériences en phase I, on permet à certains mélanges de passer en phase II, ce qui génère de nouvelles expériences. Le taux d'occupation des barres peut grâce à cela rester important. Il répond de plus au critère de minimisation du makespan en diminuant le chemin critique des tâches. La seconde partie des barres est destinée à finir au plus vite des tâches, ce qui correspond à minimiser la somme des dates d'achèvement $\sum C_i$ et à obtenir régulièrement des résultats exploitables par les chimistes. Ils doivent en effet disposer de l'ensemble des résultats des expérimentations sur un produit donné pour conclure quand à une future exploitation.

Le choix des expériences à placer dans les barres de chacune de ces parties résulte d'une optimisation locale suivant le critère concerné. Certaines informations supplémentaires,

qui ne sont pas décrites ici pour des raisons de confidentialité, interviennent aussi dans ce choix et permettent d'améliorer le taux de remplissage des barres.

Algorithme

L'algorithme que nous proposons est énoncé ci-dessous (Algorithme 1). Nous décrivons les notations employées puis chacune de ses étapes est ensuite détaillée.

L'ensemble A correspond aux lièvres (les tâches les plus avancées, déjà en phase II) qu'on souhaite faire avancer encore plus vite et l'ensemble B aux tortues (les tâches les moins avancées, encore en phase I) qu'on ne doit pas oublier.

On définit sur $A \cup B$ des ordres totaux $\leq_A$ et $\leq_B$, l'ordre $\leq_A$ favorisant les tâches de l'ensemble A, l'ordre $\leq_B$ les tâches de l'ensemble B.

On note p le nombre de barres allouées à l'avancement de la phase I à un instant donné. Il est mis à jour à chaque fin ou début de cuisson. p^* est la valeur "idéale" de p.

Algorithme 1 Planification

À un instant où une barre se libère et n'est pas pré-allouée :
Calculer $p*$
Calculer les ordres $\leq_A$ et $\leq_B$
Si $p* > p$ **Alors**
  La première tâche de $\leq_B$ est affectée à la barre
Sinon
  Choisir une tâche a parmi les premières de $\leq_A$ et fixer son ordonnancement
Fin Si
Mettre à jour p en fonction de la tâche choisie

On donne dans la suite de cette section une description plus précise des différentes étapes de l'algorithme.

Calculer les ordres $\leq_A$ et $\leq_B$

Les ordres $\leq_A$ et $\leq_B$ reflètent les priorités (selon chacun des deux critères) que l'on souhaite attribuer aux différentes tâches.

Ils sont définis comme suit :
- Ordre $\leq_A$: les tâches dont la phase I est terminée pour tous les mélanges, triées par durée croissante de la plus longue expérience restante. Puis les tâches encore en phase I.
- Ordre $\leq_B$: les tâches dont la phase I n'est pas terminée, triées par longueur décroissante de leur chemin critique. Puis les tâches dont la phase I est terminée.

On n'affectera pas toujours des expériences de la première tâche d'un des deux ordres à la barre qui se libère : on souhaite maintenir un taux élevé d'occupation des barres, ce qui intervient aussi dans le choix des expériences qui y seront placées.

Calculer p^*

p est le nombre courant de barres allouées à l'avancement de la phase I (ensemble A) et p^* est le nombre "idéal" de barres allouées à la phase I. L'idée est de répartir "équitablement" les barres entre les deux ensembles A et B par rapport aux durées d'ordonnancement de chacun. Si t_1 est l'espérance du nombre de jours nécessaires pour terminer les tâches de la phase I (on suppose que toutes les barres sont disponibles) et t_2 est le nombre de jours nécessaires pour terminer les tâches de la phase II (on suppose que toutes les barres sont disponibles), alors nous choisissons la valeur de p^* suivante, afin que les deux ordonnancements aient la même durée en se partageant les m barres :

$$p^* = m\frac{t_1}{t_1 + t_2}$$

Choisir une tâche et fixer son ordonnancement

Il s'agit de choisir une tâche susceptible de finir au plus tôt. Les candidats possibles sont connus : les attardés encore en phase I ont peu de chance de faire partie des échappés. Le choix se fait alors sur des tâches en phase II, dont on connaît à ce moment toutes les expériences restantes. On peut calculer la date de fin au plus tôt pour chacune en recherchant l'ordonnancement de ses expériences qui minimise le $Cmax$. Ce problème est difficile dans le cas général, mais peut être résolu efficacement avec peu d'expériences, et peu de durées différentes. Néanmoins le temps de calcul peut être prohibitif si nous considérons toutes les tâches de A : l'ordre $\leq_A$ nous permet de limiter le temps de calcul en nous restreignant uniquement aux premières tâches.

Le choix de la tâche pourrait se faire selon plusieurs critères :

1. Durée totale de réalisation de l'ensemble des expériences en fonction des disponibilités des barres ;
2. Aire de l'ensemble des expérimentations pour cette tâche sur le diagramme de Gantt. Ceci permet de favoriser le remplissage des barres par les expériences de cette tâche.

Nous retenons ici le choix 1.

- Choisir la tâche a : parmi les premières tâches de $\leq_A$, choisir celle qui peut terminer le plus rapidement (en tenant compte des dates de disponibilité des barres et en réservant p barres à la phase I) en calculant l'ordonnancement optimal de ses expérimentations pour le $Cmax$.
- Pré-allocation des barres : fixer le début de chaque expérience restante pour la tâche a.

La Figure 3.3 montre une partie de la fenêtre principale de l'application en cours d'exécution. Elle permet à l'utilisateur d'interagir avec l'ordonnanceur.

Pour des raisons de confidentialité, les informations concernant les tâches ne sont pas affichées sur cette figure, mais il est possible de les visualiser grâce à cette interface. Le logiciel offre la possibilité de déplacer un chargement en cas d'absence imprévue ou d'indisponibilité des opérateurs, d'ajouter des jours de congés ou de gérer les pannes matérielles. Si cela est nécessaire, l'ensemble des actions prévues sera recalculé pour prendre en compte le mieux possible les évènements récents.

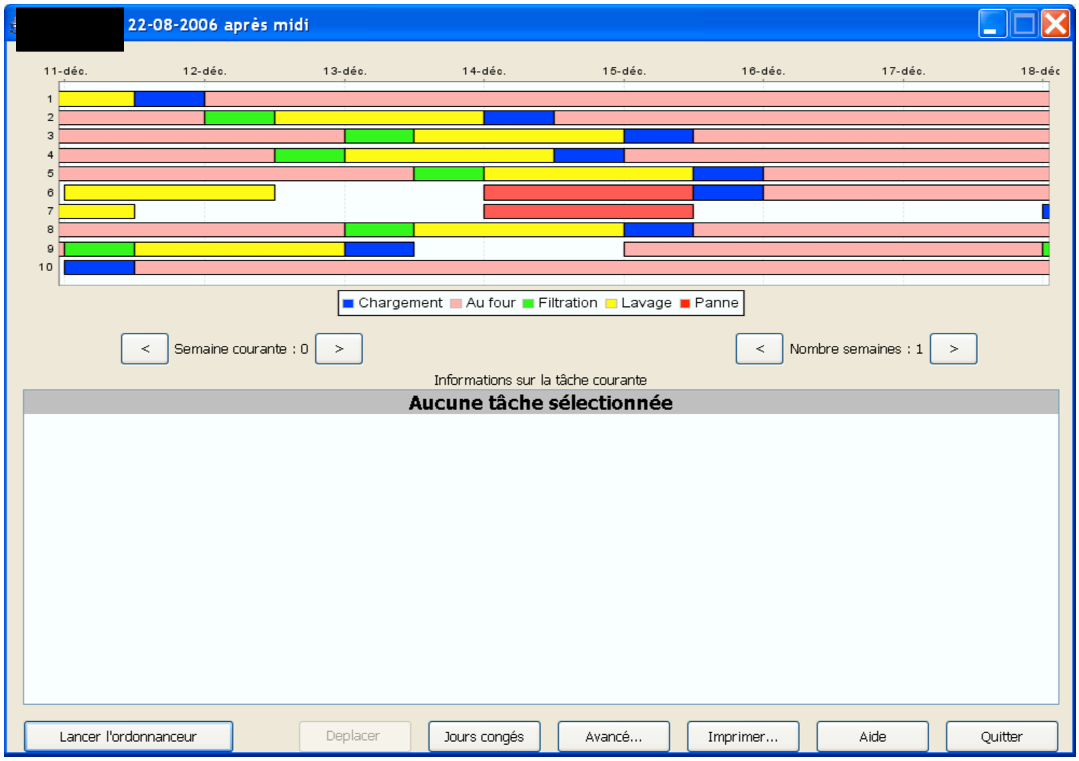


FIG. 3.3 – Fenêtre principale de l'application

3.3 Résultats numériques

Des tests numériques ont été effectués pour évaluer les performances de l'algorithme d'ordonnancement. Vu la taille du problème, il n'a pas été possible de chercher des solutions optimales pour l'un ou l'autre des deux objectifs définis (temps de calcul prohibitifs). Par contre, les objectifs industriels (utilisation des barres, sorties régulières des produits, temps pour sortir les 20 premiers produits) étaient clairement définis et ont été atteints. Nous détaillons quelques résultats numériques ci-dessous. La Figure 3.4 indique les dates de sorties des différents produits. Ces dates sont relativement régulières : un produit sort tous les 10 jours en moyenne avec un écart type de 7,5 jours. La durée totale pour sortir les 20 premiers produits est légèrement supérieure à 200 jours.

Une des exigences de l'industriel était de bien remplir les barres et de ne pas les laisser inutilisées. Pour les jeux de données testés, les barres étaient utilisées 90% du temps et remplies en moyenne à 82% ce qui satisfaisait les critères retenus.

Les Figures 3.5 et 3.6 indiquent l'influence du nombre de barres utilisées sur le résultat (écart moyen entre deux produits et durée totale de l'ordonnancement). On note que lorsqu'on augmente le nombre de barres, on améliore ces critères. Ceci montre que, malgré les anomalies de Graham connues pour les algorithmes de listes, notre approche utilise efficacement les ressources supplémentaires.

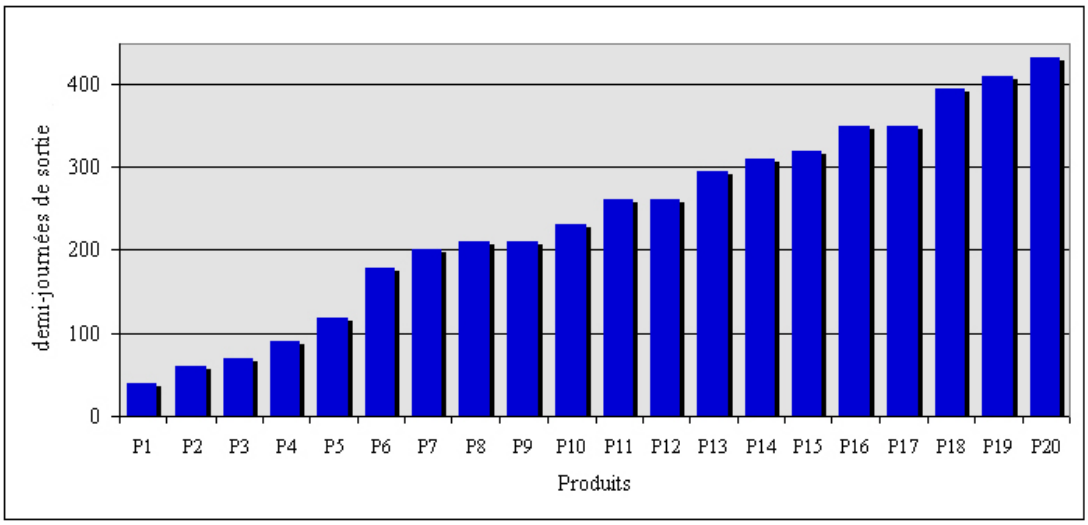


FIG. 3.4 – Dates de sortie des produits

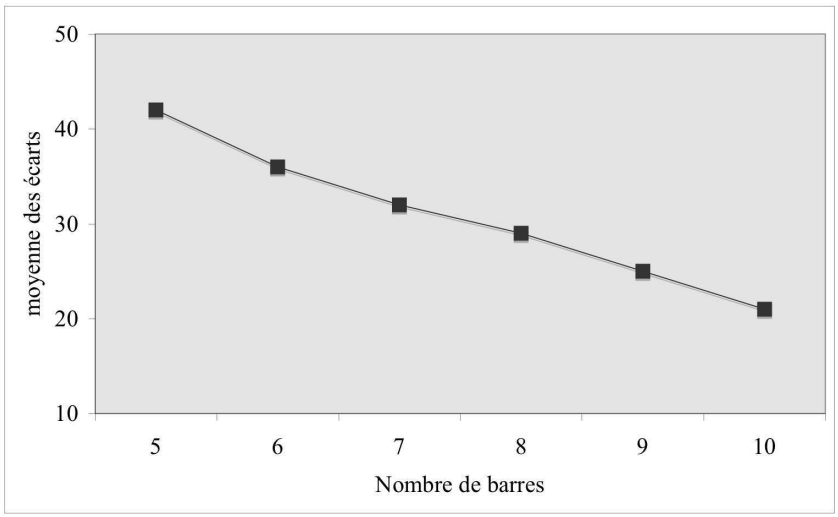


FIG. 3.5 – Écart moyen entre les dates de sortie de deux produits (en demi-journées) en fonction du nombre de barres utilisées

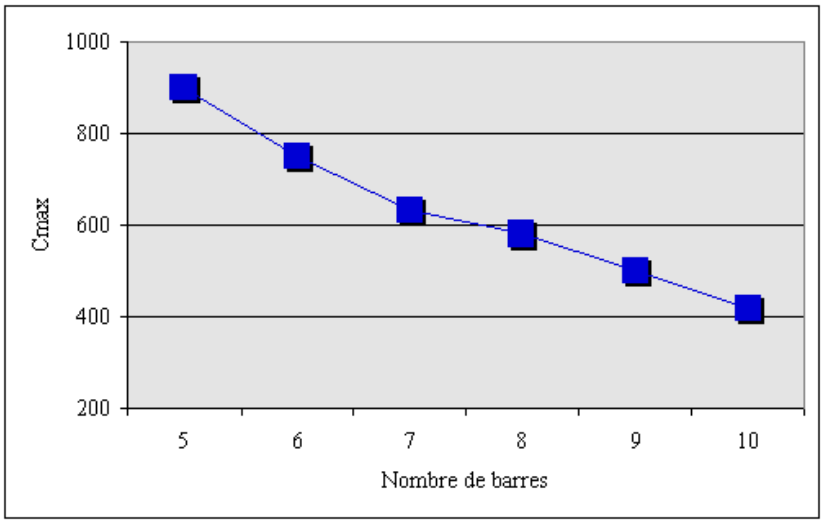


FIG. 3.6 – Valeur du $Cmax$ (en demi-journées) en fonction du nombre de barres utilisées

3.4 Problèmes théoriques dérivés

Nous avons tiré de cette collaboration avec l'IFP plusieurs problèmes théoriques qui représentent chacun un aspect du problème industriel. Ces problèmes théoriques sont tous basés sur sa version offline.

3.4.1 Périodes durant lesquelles aucune tâche ne peut commencer ou finir

La notion de périodes d'indisponibilité pour les machines est bien connue dans la littérature, puisque de nombreuses applications réelles demandent l'arrêt des machines pendant des durées plus ou moins longues, par exemple pour effectuer la maintenance [78, 79]. Dans notre cas, les machines sont disponibles tous les jours, mais pas les chimistes qui les font fonctionner. Pour les opérations de remplissage, filtration et lavage, les opérateurs sont nécessaires pendant la durée complète de l'opération, et leur absence pendant les jours fériés ou les week-ends peut être assimilée à une période d'indisponibilité classique. Il n'en est pas de même pour les cuissons, lors desquelles les chimistes ne doivent intervenir qu'au début et à la fin de l'opération. Nous pouvons donc décrire ainsi le cadre du problème : on dispose de machines parallèles pour effectuer un ensemble de tâches et on connaît l'ensemble $S(I)$ des périodes d'indisponibilité des opérateurs sous forme d'intervalles ouverts. Les solutions admissibles seront des ordonnancements dans lesquels aucune opération ne commence ni ne termine dans les intervalles de $S(I)$.

Pour des pièces différentes, minimiser le makespan est $\mathcal{NP}$-Difficile, même sur une seule machine avec un unique intervalle d'indisponibilité (le Chapitre 4 étudie plus particulièrement ce critère). On peut formuler le problème comme suit. Étant donnés n opérations de durées d'usinage $p_1, p_2, \ldots, p_n$ et une unique période d'indisponibilité de l'opérateur (période ONA pour *Operator Non Availability* en anglais) de durée Δ située en $]s, s+\Delta[$,

on peut considérer trois classes de problèmes.

(1) Toutes les durées de production sont plus longues que Δ : $p_j \geq \Delta \quad \forall j$;

(2) Toutes les durées de production sont plus petites que Δ : $p_j < \Delta \quad \forall j$;

(3) Il y a des durées d'usinage plus petites que Δ et d'autres plus grandes que Δ.

Le cas (1) en particulier correspond bien à notre application. En effet, les cuissons excèdent le plus souvent 3 jours, alors que les durées pendant lesquelles l'ensemble des chimistes sont absents sont en général inférieures à 2 jours (il s'agit des week-ends et des jours fériés). Dans ce cas, nous montrons que pour une machine et un intervalle, minimiser le makespan peut être fait en $O(n \log n)$ où n est le nombre de cuissons.

3.4.2 Ordonnancer des séquences d'opérations avec traitement par batch et compatibilité des tâches

Dans le problème offline associé au problème de planification de l'IFP, les opérations en phase I d'un même mélange sont liées entre elles par des contraintes de précédences de type *chains*, c'est-à-dire que leur graphe de précédences est une chaîne.

De plus, les machines que nous considérons sont de capacités non unitaires, c'est-à-dire qu'elles peuvent effectuer simultanément plusieurs opérations. Pour pouvoir les placer dans un même batch, il faut que les opérations soient batch-compatibles comme expliqué à la Section 3.1.1.

On souhaite minimiser le makespan pour un problème faisant apparaître ces deux aspects.

Exemple 1. L'exemple suivant est composé de deux tâches dont la séquence des opérations est donnée par :

$$21 \rightarrow 5 \rightarrow 14 \quad \text{et} \quad 5 \rightarrow 14 \rightarrow 21$$

Une séquence possible serait

21	5 5	14 14	21

où les opérations de longueurs 5 et 14 sont regroupées dans des batchs de capacité 2 et les opérations de durées 21 sont ordonnancées seules. Le makespan est alors de 61.

L'ordonnancement optimal est de durée 59 :

5	14	21 21	5	14

3.4.3 Minimiser la somme des dates d'achèvement des produits

Lorsqu'un produit a terminé sa phase I, les opérations restantes (en phase II) ne sont plus liées par des contraintes de précédences.

On considère uniquement les produits passés en phase II. La date de fin d'un produit est la date d'achèvement de la dernière expérience qui sera effectuée pour ce produit. Comme nous l'avons dit, minimiser la somme des dates d'achèvement des produits permet aux chimistes d'exploiter régulièrement des résultats au cours du temps alors que la minimisation du makespan seul ne garanti pas ce résultat. On peut donc poser le problème comme suit. Une tâche T_i est un ensemble d'opérations. On note C'_i la date d'achèvement de T_i, c'est-à-dire la fin de sa dernière opération. Les tâches sont toutes indépendantes. L'objectif est de minimiser $\sum C'_i$. On remarque que ce problème est très proche du problème proposé dans [66]. Une fois encore, nous devons effectuer l'ensemble des opérations dans un environnement de type machines parallèles. Ce problème est bien entendu différent du problème classique $Pm||\sum C_i$ où C_i est la date d'achèvement de chaque opération. La règle SPT (*Shortest Processing Time* en anglais) donne alors une solution optimale en temps polynomial [73]. Le problème n'est pas non plus directement relié à $Pm||\sum w_i C_i$ puisque l'on ne peut pas savoir quelle sera la dernière opération de chaque tâche. Ce serait le cas si des précédences quelconques entre les opérations existaient, puisqu'il suffirait d'ajouter à chaque tâche une opération fictive de durée nulle. Cette opération ne pourrait se dérouler qu'une fois toutes les autres effectuées. Le poids associé à cette opération dans la fonction objectif serait alors de 1 tandis que les autres opérations de la tâche se verraient attribuer un poids nul.

Si nous considérons maintenant de nouveau notre problème, nous constatons que, s'il est restreint à une unique machine, alors la règle SPT appliquée à la somme des durées des opérations de la tâche est optimale. Ainsi, $1||\sum C'_i$ est polynomial.

Pour deux machines ($P2||\sum C'_i$), le problème est $\mathcal{NP}$-Difficile : il contient la minimisation du makespan ($P2||Cmax$) comme sous-cas (c'est le cas particulier où on organise les opérations d'une unique tâche), et est donc $\mathcal{NP}$-Difficile [68].

3.5 Conclusion

Un logiciel d'ordonnancement d'expériences a été élaboré et implémenté en JAVA pour répondre à un problème industriel. Les tests effectués montrent que les objectifs sont atteints : le taux d'occupation des barres est élevé et les dates de fin des tâches sont rapprochées et régulières. Pour des raisons de confidentialité, il n'est pas possible de donner ici le diagramme des classes du logiciel. Il est actuellement en cours d'utilisation à l'IFP et donne satisfaction aux utilisateurs finaux. Des problèmes théoriques en ordonnancement ont également été dérivés de cette collaboration. L'un d'entre eux, complètement nouveau, est décrit en détail dans le Chapitre 4 ainsi que les résultats de complexité qui s'y rapportent.

Chapitre 4

Ordonnancement avec indisponibilité des opérateurs

La notion de période d'indisponibilité pour les machines est bien connue dans la littérature, puisque de nombreuses applications réelles demandent l'arrêt des machines pendant des durées plus ou moins longues, par exemple pour effectuer la maintenance [78, 79, 65]. Dans l'application industrielle présentée au Chapitre 3, une nouvelle notion de période d'indisponibilité apparaît. Ce ne sont plus les machines qui font défaut mais des opérateurs humains qui quittent le site pendant les week-ends, les jours fériés et leurs vacances. Ces absences ne perturbent pas le fonctionnement des machines qui sont opérationnelles tous les jours. Les opérateurs sont par contre nécessaires à chaque début et à chaque fin de tâche. En effet, dans le problème de planification du Chapitre 3, les opérations sont des expériences et leur usinage correspond à une cuisson. Ces cuissons peuvent se dérouler sans les chimistes, mais ils sont nécessaires pour préparer les mélanges, introduire et retirer le matériel du four.

Nous obtenons donc un nouveau type de problèmes d'ordonnancement qui doit respecter les périodes d'indisponibilité des opérateurs humains. La définition formelle de ces périodes est la suivante :

Définition 15 (Période Ona) *Une* période d'indisponibilité de l'opérateur *ou* période Ona *(pour* operator non-availability *en anglais) de longueur* Δ *correspond à un intervalle ouvert* $]s, s+\Delta[$ *durant lequel aucune opération ne peut commencer ou finir. On dit d'une opération qui commence avant (ou à) l'instant* s *et termine après (ou à) l'instant* $s+\Delta$ *qu'elle* couvre *la période* Ona.

Nous nous intéressons dans ce chapitre aux problèmes à une unique machine avec périodes d'indisponibilité de l'opérateur. Le critère qui focalise notre attention est le makespan. Les problèmes étudiés étant en général $\mathcal{NP}$-Difficiles, nous analysons les performances des algorithmes s'appuyant sur une liste de priorité.

Nous noterons le problème de minimisation du $Cmax$ sur une machine avec $\mathcal{K}$ périodes d'indisponibilité $1\,|ona(\mathcal{K})|\,Cmax$ si $\mathcal{K}$ ne fait pas partie de l'instance et $1\,|ona|\,Cmax$ sinon. Nous utiliserons pour les données du problème les notations suivantes :

- n, le nombre de tâches. Les tâches sont notées $T_1, T_2, \ldots, T_n$;
- p_j, la durée d'usinage de la tâche T_j, $j = 1, 2, \ldots, n$;
- $\mathcal{K}$, le nombre de périodes ONA ;
- s_q, l'instant de début de la $q^{ième}$ période ONA, $q = 1, 2, \ldots, \mathcal{K}$;
- Δ_q, la durée de la $q^{ième}$ période ONA, $q = 1, 2, \ldots, \mathcal{K}$;
- $s_0 = \Delta_0 = 0$ et $s_{\mathcal{K}+1} = +\infty$ par convention.

Le problème se pose donc de la façon suivante :

OPERATOR NON-AVAILABILITY PERIODS SCHEDULING (ONAS)
INSTANCE : Un ensemble de n tâches, de durées $p_1, p_2, \ldots, p_n$, et un ensemble de $\mathcal{K}$ intervalles $]s_q, s_q + \Delta_q[$, $q = 1, 2, \ldots, \mathcal{K}$;
SOLUTION : Un ordonnancement S, tel qu'aucune tâche ne commence ni ne termine dans aucun des intervalles $]s_q, s_q + \Delta_q[$;
CRITÈRE : Le makespan de S, $Cmax(S)$.

Chaque période ONA plus longue que toutes les durées d'usinage se comporte comme une période d'indisponibilité de la machine. Ainsi, si c'est le cas de l'ensemble des périodes ONA, le problème est équivalent au cas classique des périodes d'indisponibilité des machines.

Sauf mention contraire, nous supposons dans ce chapitre que les tâches sont numérotées par ordre croissant de leurs durées d'usinage $p_1 \leq p_2 \leq \cdots \leq p_n$.

Afin de ne pas traiter de cas triviaux, nous prenons aussi l'hypothèse que $n \geq 2$ et $\sum_{j=1}^{n} p_j > s_1$, c'est-à-dire que plus de deux tâches doivent être usinées et qu'aucun ordonnancement réalisable ne termine avant la première période ONA.

Les preuves qui apparaissent dans ce chapitre utilisent les notations suivantes :
- Pour tout ensemble E, on note $p(E) = \sum_{j \in E} p_j$ et lorsque des entiers $a_1, a_2, \ldots$ sont utilisés, $a(E) = \sum_{j \in E} a_j$;
- Étant donné un ordonnancement S,
  - k est l'indice de la dernière période ONA qui termine avant la fin de S : $k = \max\{q \leq \mathcal{K} | s_q + \Delta_q \leq Cmax(S)\}$;
  - x_q est le temps mort total dans l'intervalle $[s_{q-1} + \Delta_{q-1}, s_q]$, pour tout $q = 1, 2, \ldots, k$;
  - X_q est le temps mort total dans l'intervalle $[s_{q-1} + \Delta_{q-1}, s_q + \Delta_q]$, pour tout $q = 1, 2, \ldots, k$.

> **Remarque.** Considérons un ordonnancement S. Si la $q^{ième}$ période ONA est couverte dans S, alors $X_q = x_q$ et $X_q = x_q + \Delta_q$ sinon.

4.1 Une seule période ONA

Le problème $1\,|ona(1)|\,Cmax$ peut être formulé comme suit. Étant données n tâches et une seule période d'indisponibilité $]s, s+\Delta[$ de longueur Δ, trouver un ordonnancement S qui minimise le makespan.

Nous donnons d'abord la propriété de dominance suivante :

Lemme 4 *Pour le problème* $1\,|ona(1)|\,Cmax$*, si la tâche la plus longue,* T_n*, est telle que* $p_n \geq \Delta$*, alors il existe un ordonnancement optimal dans lequel* T_n *couvre la période* ONA.

Démonstration. Considérons un ordonnancement optimal S^* de makespan $Cmax(S^*)$. Supposons que dans S^*, la tâche T_n ne couvre pas la période d'indisponibilité de l'opérateur.

Soit E l'ensemble des tâches qui finissent avant ou à l'instant s. Supposons que T_n est située dans l'ensemble E. Sans perte de généralité, la tâche T_n est ordonnancée en dernière parmi les tâches de l'ensemble E. Si la période $]s, s+\Delta[$ n'est couverte par aucune tâche, on peut repousser l'instant de départ de T_n pour qu'elle termine à l'instant $s+\Delta$ (elle couvrira alors l'intervalle) sans modifier le makespan. Si une certaine tâche T_v avec $p_v \leq p_n$ recouvre l'intervalle, alors la tâche T_v suit immédiatement T_n et on peut échanger T_n et T_v dans S^* sans changer $Cmax(S^*)$.

Supposons maintenant que la tâche T_n est incluse dans l'ensemble E' des tâches qui commencent après la période ONA, c'est-à-dire après ou à l'instant $s+\Delta$. Sans perte de généralité, la tâche T_n est ordonnancée la première parmi les tâches de l'ensemble E'. Si la période $]s,\ s+\Delta[$ n'est pas couverte, on peut commencer la tâche T_n à l'instant s et diminuer de la même durée les instants de départ de toutes les autres tâches de E', diminuant ainsi le makespan. Si une tâche T_v avec $p_v \leq p_n$ couvre la période, alors la tâche T_v précède immédiatement la tâche T_n et on peut échanger les tâches T_n et T_v sans augmenter le makespan. □

Pour le problème $1\,|ona(1)|\,Cmax$, on peut distinguer trois classes d'instances qui définissent chacune un sous problème particulier :

Problème (1) : toutes les durées d'usinage sont plus longues que Δ : $p_j \geq \Delta$ pour tout $j = 1, 2, \ldots, n$;
Problème (2) : toutes les durées d'usinage sont plus courtes que Δ : $p_j < \Delta$ pour tout $j = 1, 2, \ldots, n$;
Problème (3) : certaines durées d'usinage sont plus courte que Δ et d'autres plus longues.

Le Problème (1) est particulièrement intéressant pour notre application industrielle : les durées de cuisson des expériences sont presque toujours supérieures à 3 jours, alors que les périodes durant lesquelles les chimistes sont absents, sont le plus souvent de durées inférieures à deux jours puisqu'il s'agit des week-ends et des jours fériés.

Le Problème (2) est un cas particulier du problème avec une période d'indisponibilité de la machine.

Les deux théorèmes suivants indiquent la complexité de ces trois problèmes. Le problème $1|ona(1)|Cmax$ est polynomial dans le cas où la période ONA est plus courte que la plus petite durée d'usinage. On montre que ce cas particulier est lié au problème SUBSET SUM.

Théorème 4 *Le Problème (1) admet un algorithme polynomial en* $O(n \log n)$.

Démonstration. Le Lemme 4 implique qu'il existe un ordonnancement optimal S^* dans lequel la plus longue tâche T_n couvre la période ONA. On peut distinguer deux cas :

1. Aucun temps mort ne survient dans l'ordonnancement optimal ;
2. Un temps mort survient dans l'ordonnancement optimal avant la période ONA.

Notons E^* l'ensemble des tâches qui finissent avant la période ONA dans S^* et $\delta = p_n - \Delta$ la différence entre la plus longue tâche et la durée de la période ONA. Le makespan de l'ordonnancement optimal est donné par $Cmax(S^*) = \sum_{j=1}^{n} p_j + \max\{s - \delta - p(E^*),\ 0\}$.

Clairement, dans le second cas, E^* doit être le sous-ensemble de tâches dont la somme des durées est maximale tout en restant inférieure à $s - \delta$.

Dans le premier cas, le makespan est égal à $\sum_{j=1}^{n} p_j$, et ainsi $p(E^*) \in [s - \delta, s]$.

Remarquons que tout sous-ensemble E de $\{1, 2, \ldots, n-1\}$ tel que $p(E) \in [s - \delta, s]$ définit un ordonnancement optimal en plaçant les tâches de E dans un ordre quelconque, suivies de T_n, puis des tâches restantes, une fois encore dans un ordre quelconque. Ainsi, décider s'il existe un ordonnancement de makespan $\sum p_j$ est un cas particulier du problème suivant :

SUBSET SUM WITH TOLERANCE
INSTANCE : $m + 2$ entiers $a_1, a_2, \ldots, a_m$, B et δ ;
QUESTION : Existe-t-il un sous-ensemble J de $\{1, 2, \ldots, m\}$ tel que $\sum_{j \in J} a_j \in [B - \delta, B]$?

Ce problème est $\mathcal{NP}$-Difficile dans le cas général puisque le problème SUBSET SUM correspond au cas particulier où $\delta = 0$.

Proposition 6 *Si $\delta \geq \max_{i,j} |a_i - a_j|$, le problème* SUBSET SUM WITH TOLERANCE *admet un algorithme exact en temps $O(m \log m)$.*

Démonstration. On montre que l'Algorithme 2 fournit un ensemble $J \subseteq \{1, 2, \ldots, m\}$ tel que $a(J) \in [B - \delta, B]$, si un tel ensemble existe ou, dans le cas contraire, le plus grand sous-ensemble (relativement à la somme de ses éléments) tel que la somme des a_j est plus petite que $B - \delta$.

Algorithme 2 Algorithme SubSetSumWithTolerance

ENTRÉES: $m + 2$ entiers $a_1, a_2, \ldots, a_n$, B et δ.
SORTIES: un sous-ensemble $J \subseteq \{1, 2, \ldots, m\}$, si il existe, tel que $a(J) \in [B - \delta, B]$, et sinon le plus grand sous-ensemble J inférieur à $B - \delta$.
Trier l'ensemble $\{a_1, a_2, \ldots, a_m\}$ de sorte que les a_j soient indexés par ordre croissant de leur valeur.
Soit η le plus grand indice tel que $\sum_{j=1}^{\eta} a_j \leq B$.
Initialiser $J = \{1, 2, \ldots, \eta\}$.
Tant que $a(J) < B - \delta$ et $\min\{a_j | j \in J\} < \max\{a_i | i \notin J\}$ **Faire**
  Choisir deux indices quelconques $j \in J$ et $i \notin J$ tels que $a_j < a_i$.
  $J = (S \backslash \{j\}) \cup \{i\}$.
Fin Tant que
Renvoyer J.

Soient $J_0 = \{1, 2, \ldots, \eta\}$, J_1, ..., J_q les ensembles successifs considérés par l'algorithme. Par construction, les $a(J_l)$ forment une suite croissante. De plus, entre deux pas de l'algorithme, la somme des éléments de l'ensemble J ne peut pas augmenter de plus de δ,

puisque $a(J_{l+1}) = a(J_l) + a_j - a_i \leq a(J_l) + \delta$ où i et j sont les indices utilisés à ce pas de l'algorithme. Ainsi, $a(J_l) \leq B - \delta$ implique que $a(J_{l+1}) \leq B$.

Supposons maintenant que l'algorithme ne parvienne pas à trouver un sous-ensemble qui finisse dans $[B - \delta, B]$. Puisque $a(J_0) < B - \delta$, cela implique, d'après ce qui précède, que $a(J_q)$ est aussi inférieur à $B - \delta$. Par construction, l'ensemble J_q contient dans ce cas les η plus longues tâches à la fin de l'exécution de l'algorithme. Mais la définition de η implique qu'il n'y a pas de sous-ensemble de $\eta + 1$ tâches de durée totale inférieure à B. Ainsi, J_q est le plus grand sous-ensemble inférieur à B.

Une implémentation possible de cet algorithme est d'échanger à chaque pas le plus petit élément de J et le plus grand élément qui n'est pas dans J, bornant par η le nombre de pas. Des pointeurs vers ces éléments peuvent être maintenus en temps constant puisque les a_j sont triés. Ainsi, la complexité totale de l'algorithme est dominée par le tri des éléments qui s'effectue en $O(m \log m)$. □

En utilisant l'ensemble J fourni par l'Algorithme 2 pour l'entrée $(p_1, p_2, \ldots, p_{n-1}, s, p_n - \Delta)$, on peut résoudre de façon optimale le problème $1|ona(1)|Cmax$ en ordonnançant en premier les tâches de J dans n'importe quel ordre, puis T_n, et enfin les tâches restantes dans n'importe quel ordre. □

Étant donné que pour le problème avec une période d'indisponibilité de la machine est $\mathcal{NP}$-Difficile [62] et que clairement la taille de la période d'indisponibilité de la machine n'influe pas sur cet aspect du problème, le Problème (2) est $\mathcal{NP}$-Difficile. Nous donnons ici une réduction du problème SUBSET SUM vers le Problème (2) afin d'être complet et de souligner les liens entre les problèmes $1|ona(1)|Cmax$ et SUBSET SUM.

Théorème 5 *Les problèmes de décision associés aux Problèmes (2) et (3) sont $\mathcal{NP}$-Complets.*

Démonstration. Clairement, les deux problèmes sont dans $\mathcal{NP}$. Considérons le problème suivant :

SUBSET SUM
INSTANCE : n nombres $a_1, a_2, \ldots, a_n$ et une borne B ;
QUESTION : Existe-t-il un sous-ensemble $J \subseteq \{1, 2, \ldots, n\}$ tel que $\sum_{j \in J} a_j = B$?

Il se réduit vers chacun des Problèmes (2) et (3) de la façon suivante.

Problème (2) : on utilise n tâches avec $p_j = a_j$, $\Delta = \max_j a_j + 1$, l'intervalle $]B, B + \Delta[$, la borne $D = \sum_{j=1}^{n} a_j + \Delta$ et la question "Existe-t-il un ordonnancement S de makespan $Cmax(S) \leq D$?".

La période ONA n'est couverte dans aucun ordonnancement admissible. Ainsi, pour tout ordonnancement S, la condition $Cmax(S) \leq D$ implique $Cmax(S) = D$ et l'ordonnancement n'a pas de temps morts (autre que la période ONA), c'est-à-dire qu'il existe un sous-ensemble J tel que $\sum_{j \in J} a_j = B$.

Problème (3) : on utilise $n+1$ tâches avec $p_j = a_j$ pour les tâches $T_1, T_2, \ldots, T_n$; et $p_{n+1} = \sum_{j=1}^{n} a_j + 1$ pour la tâche T_{n+1}. On pose $\Delta = \sum_{j=1}^{n} a_j + 1$ et l'intervalle d'indisponibilité de l'opérateur est $]B, B + \Delta[$, la borne $D = \sum_{j=1}^{n+1} p_j$.

De même que précédemment, $Cmax(S) \leq D$ implique $Cmax(S) = D$ et il n'y a aucun temps mort. Or, la tâche T_{n+1} couvre dans ce cas exactement la période ONA. Ainsi, il existe un sous-ensemble J tel que $\sum_{j \in J} a_j = B$. □

Pour les Problèmes (2) et (3), même si aucun algorithme ne permet d'obtenir la solution optimale en temps polynomial (sauf si $\mathcal{P} = \mathcal{NP}$), un schéma d'approximation pleinement polynomial peut être proposé.

Théorème 6 *Le problème* $1\,|ona(1)|\,Cmax$ *est* $\mathcal{NP}$*-Difficile au sens faible et admet un* $\mathcal{FPTAS}$.

Démonstration. D'après le Théorème 5, le problème $1\,|ona(1)|\,Cmax$ est $\mathcal{NP}$-Difficile dans le cas général.

Considérons deux cas. Si $p_n < \Delta$, alors l'intervalle d'indisponibilité n'est couvert dans aucun ordonnancement optimal. Dans ce cas, on peut transformer l'instance de $1|ona(1)|Cmax$ en l'instance suivante de SUBSET SUM : trouver un ensemble d'indices $E \subseteq \{1, 2, \ldots, n\}$ de tâches tel que $p(E) \leq s$ et $p(E)$ est maximal.

Si $p_n \geq \Delta$, alors d'après le Lemme 4 on peut supposer que la tâche T_n couvre la période d'indisponibilité dans l'ordonnancement optimal. Cette fois, notre instance se transforme en l'instance suivante de SUBSET SUM : trouver un ensemble d'indices $E \subseteq \{1, 2, \ldots, n-1\}$ tel que $p(E) \leq s$ et $p(E)$ est maximal.

Dans les deux cas, la réduction est polynomiale.

Le problème $1|ona(1)|Cmax$ admet donc un algorithme exact en temps pseudo-polynomial, puisque le problème SUBSET SUM peut être résolu en temps pseudo-polynomial par un programme dynamique. De plus, le problème SUBSET SUM admet un $\mathcal{FPTAS}$ qui peut être utilisé pour développer un $\mathcal{FPTAS}$ pour $1\,|ona(1)|\,Cmax$.

Supposons qu'un $\mathcal{FPTAS}$ appliqué au problème SUBSET SUM requis trouve un ensemble E_ε, alors que la solution optimale serait donnée par l'ensemble E^*. Par définition, $p(E_\varepsilon) \geq (1-\varepsilon)p(E^*)$.

- Si $p_n < \Delta$, alors pour tout ordonnancement optimal S^* on a $Cmax(S^*) = s + \Delta + \sum_{j=1}^{n} p_j - p(E^*)$. En ordonnançant les tâches T_j telles que $j \in E_\varepsilon$ avant la période ONA, on obtient un ordonnancement S_ε et $Cmax(S_\varepsilon) = s + \Delta + \sum_{j=1}^{n} p_j - p(E_\varepsilon)$. Il s'en suit que

$$Cmax(S_\varepsilon) - Cmax(S^*) = p(E^*) - p(E_\varepsilon) \leq \varepsilon\, p(E^*) \leq \varepsilon\, s \leq \varepsilon\, Cmax(S^*)$$

- Si $p_n \geq \Delta$, alors il existe un ordonnancement optimal S^*, où la tâche T_n suit l'ensemble des tâches T_j telles que $j \in E^*$. Si $p(E^*) + p_n > s + \Delta$, alors il n'existe pas de temps mort dans S^* ; sinon, la tâche T_n se termine exactement à l'instant $s + \Delta$. Ainsi on a $Cmax(S^*) = \max\left\{\sum_{j=1}^{n} p_j,\ s + \Delta + \sum_{j=1}^{n} p_j - p(E^*) - p_n\right\}$.

$$Cmax(S^*) \;=\; \sum_{j=1}^{n} p_j + \max\left\{s + \Delta - (p(E^*) + p_n),\ 0\right\}$$

De même, en ordonnaçant la tâche T_n après l'ensemble des tâches T_j telles que $j \in E_\varepsilon$, on obtient un ordonnancement S_ε vérifiant $Cmax(S_\varepsilon) = \sum_{j=1}^{n} p_j + \max\{s + \Delta - (p(E_\varepsilon) + p_n), 0\}$. Si $s + \Delta \leq p(E_\varepsilon) + p_n$ alors $s + \Delta \leq p(E^*) + p_n$ et $Cmax(S_\varepsilon) = \sum_{j=1}^{n} p_j$, ce qui signifie que S_ε est optimal.
Supposons maintenant que $s + \Delta > p(E_\varepsilon) + p_n$.
Si $s + \Delta > p(E^*) + p_n$, alors

$$Cmax(S_\varepsilon) - Cmax(S^*) \leq p(E^*) - p(E_\varepsilon) \leq \varepsilon\, p(E^*) \leq \varepsilon\, Cmax(S^*)$$

Sinon, si $s + \Delta \leq p(E^*) + p_n$ et $Cmax(S^*) = \sum_{j=1}^{n} p_j$, alors $Cmax(S_\varepsilon) - Cmax(S^*) = s + \Delta - (p(E_\varepsilon) + p_n) \leq s + \Delta - ((1-\varepsilon)p(E^*) + p_n) \leq \varepsilon p(E^*)$. Dans ce cas,

$$Cmax(S_\varepsilon) - Cmax(S^*) \leq \varepsilon\, p(E^*) \leq \varepsilon s \leq \varepsilon\, Cmax(S^*)$$

ce qui complète la preuve.

□

4.2 $\mathcal{K}$ périodes d'indisponibilité

Le problème $1|ona(1)|Cmax$ est $\mathcal{NP}$-Difficile dans le cas général d'après le Théorème 5. Dans le cas où la seule période d'indisponibilité est plus petite que toutes les durées d'usinage, alors le problème est polynomial. Nous montrons que dès que deux périodes d'indisponibilité sont présentes, le problème devient $\mathcal{NP}$-Difficile au sens faible si le nombre de périodes est fixé et au sens fort si le nombre de périodes fait partie de l'instance.

Définition 16 (Période ONA petite) *Une période d'indisponibilité de durée Δ est dite* petite *si elle est plus courte ou de même durée que la plus petite durée d'usinage :*

$$\Delta \leq p_j, \quad \textit{pour tout } j = 1, 2, \ldots, n$$

4.2.1 Nombre constant de périodes ONA

Pour une unique période ONA petite, le problème est polynomial. Une question naturelle est donc de déterminer si il reste polynomial lorsqu'on a un nombre $\mathcal{K} \geq 2$ de périodes ONA, toutes petites. La réponse est négative :

Théorème 7 *Le problème $1|ona(2)|Cmax$ est $\mathcal{NP}$-Difficile même si les deux périodes* ONA *sont petites, de même durée et placées de façon périodique (c'est-à-dire toujours séparées par un intervalle de même durée).*

Démonstration. Considérons le problème suivant :

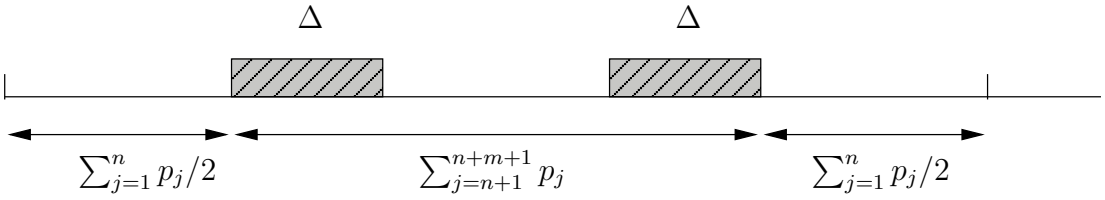
EQUAL PARTITION
INSTANCE : n entiers $a_1, a_2, \ldots, a_n$;

QUESTION : Existe-t-il une partition $A \cup B$ de $\{1, 2, \ldots, n\}$ telle que $\sum_{j\in A} a_j = \sum_{j\in B} a_j$ et $|A| = |B|$?

Ce problème est $\mathcal{NP}$-Complet (problème [SP12] dans [68]). Pour éviter les cas triviaux, nous supposons que n et $\sum_j a_j$ sont pairs : $n = 2m$ et $\sum_j a_j = 2\alpha$. On associe à une instance I de EQUAL PARTITION l'instance $f(I)$ suivante de $1|ona(2)|Cmax$:

- La durée des deux périodes ONA est $\Delta = m\alpha$;
- On considère $n + m + 1$ tâches de durées d'usinage $p_j = a_j + \Delta$, $j = 1, 2, \ldots, n$ et $p_j = \Delta + \alpha$, $j = n + 1, n + 2, \ldots, n + m + 1$;
- La première période ONA commence à l'instant $s_1 = \sum_{j=1}^{n} p_j/2 = m\Delta + \alpha$;
- La seconde période ONA commence à l'instant $s_2 = (2m+1)\Delta+2\alpha$, ainsi, $s_2-(s_1+\Delta) = s_1$ et les périodes ONA sont placées périodiquement.

Cette transformation f est clairement polynomiale. Remarquons que les périodes sont petites, puisque $p_i \geq \Delta$ pour tout i. Nous montrons à présent que la réponse pour l'instance I de EQUAL PARTITION est "oui" si et seulement si il existe un ordonnancement pour l'instance $f(I)$ de makespan au plus $D = \sum_j p_j$.



Supposons tout d'abord qu'il existe une partition $A \cup B$ telle que $\sum_{i\in A} a_i = \sum_{i\in B} a_i$ et $|A| = |B|$. On a

$$p(A) = \sum_{i\in A} a_i + |A|\Delta = \left(\sum_i a_i + n\Delta\right)/2 = s_1$$

On peut donc ordonnancer sans temps morts et dans n'importe quel ordre les tâches dont l'indice appartient à l'ensemble A pour finir exactement à l'instant s_1. Remarquons ensuite que $s_2+\Delta-s_1 = s_1+2\Delta$ et que $\sum_{j=n+1}^{n+m+1} p_j = (m+1)\Delta+(m+1)\alpha = (m+2)\Delta+\alpha$. Ainsi, $s_2 + \Delta - s_1 = \sum_{j=n+1}^{n+m+1} p_j$. Les périodes ONA étant petites, il est possible d'ordonnancer toutes les tâches $T_{n+1}, T_{n+2}, \ldots, T_{n+m+1}$ dans l'intervalle $[s_1, s_2+\Delta]$, là encore sans temps morts. Reste à ordonnancer les tâches dont l'indice appartient à B à partir de l'instant $s_2 + \Delta$ dans n'importe quel ordre pour obtenir un ordonnancement de makespan $\sum_j p_j = D$.

Réciproquement, supposons qu'un tel ordonnancement existe. Notons A l'ensemble des tâches qui finissent avant s_1. Remarquons tout d'abord que $|A| \leq m$. En effet, pour toute tâche T_j, $p_j \geq \Delta$ et donc si $|A| > m$, on a $\sum_{j\in A} p_j \geq (m + 1)\Delta > s_1$. Supposons que $|A| < m$. On a cette fois

$$\sum_{j\in A} p_j \leq |A| \max_j p_j = |A|(\alpha + \Delta) \leq (m - 1)(\alpha + \Delta) = m\Delta - \alpha = s_1 - 2\alpha$$

La tâche qui suit celles indicées dans A doit couvrir la période ONA et donc finir après $s_1 + \Delta$ pour que l'ordonnancement soit sans temps morts. Or pour toute tâche T_j, $s_1 - 2\alpha + p_j < s_1 + \Delta$. On a donc nécessairement $|A| = m$.

On peut maintenant en déduire que $A \subseteq \{1, 2, \ldots, n\}$. En effet, si il existe $j \in A$ tel que $j \geq n+1$, alors $p(A) = p(A \setminus \{j\}) + \Delta + \alpha > m\Delta + \alpha = s_1$.

Considérons maintenant l'ensemble B des tâches produites entre les deux périodes ONA, c'est-à-dire dans l'intervalle $[s_1+\Delta,\ s_2]$. Cet intervalle étant de longueur s_1, on a, de même que précédemment, $|B| \leq m$ et $|B| = m \Rightarrow B \subseteq \{1, 2, \ldots, n\}$. Supposons $|B| = m$, on a donc $A \cup B = \{1, 2, \ldots, n\}$ et si on note T_a la tâche qui couvre la première période, alors $a \geq n+1$ et $p_a + p(A) + p(B) = p_a + \sum_{j=1}^{n} p_j = \Delta + \alpha + n\Delta + 2\alpha = (n+1)\Delta + 3\alpha = s_2 + \alpha$. On obtient donc une contradiction puisque la somme des durées d'usinage de ces tâches doit être inférieure ou égale à s_2. Ainsi, $|B| \leq m-1$.

Si on note maintenant T_b la tâche qui couvre la seconde période, on a

$$p_a + \sum_{j \in B} p_j + p_b \leq (2 + |B|)(\Delta + \alpha) \leq (m+1)(\Delta + \alpha) = s_2 + \Delta - s_1$$

Ainsi, si on veut que l'ordonnancement soit sans temps morts, il faut que cette borne soit atteinte, c'est-à-dire que $p_a + \sum_{j \in B} p_j + p_b = s_2 + \Delta - s_1$ et que $B \cup \{a\} \cup \{b\} = \{n+1,\ n+2, \ldots,\ n+m+1\}$.

Puisqu'on a exactement $p_a + \sum_{j \in B} p_j + p_b = s_2 + \Delta - s_1$, il faut aussi que $\sum_{j \in A} p_j = s_1$. On a donc un ensemble $A \subseteq \{1, 2, \ldots, n\}$ tel que $|A| = n/2$ et $\sum_{j \in A} a_j = s_1 - \Delta n/2 = \alpha = \left(\sum_{j=1}^{n} a_j\right)/2$.

Si on note $C = \{1, 2, \ldots, n\} \setminus A$, on obtient $\sum_{j \in A} a_j = \sum_{j \in C} a_j$ et $|A| = |C|$. Ce qui prouve que la réponse pour $\mathcal{I}$ est "oui". □

Cependant, le problème $1|ona(\mathcal{K})|Cmax$ est $\mathcal{NP}$-Difficile "seulement" au sens faible comme le déclare la proposition suivante :

Proposition 7 *Le problème $1|ona(\mathcal{K})|Cmax$ est $\mathcal{NP}$-Difficile au sens faible pour tout $\mathcal{K} \geq 2$.*

Démonstration. Nous allons construire un algorithme pseudo-polynomial pour $1|ona(\mathcal{K})|Cmax$. On associe à tout ordonnancement S la partition $\phi(S) = (\phi(S)_j)_{j=1}^{\mathcal{K}+1}$, où $\phi(S)_j$ est l'ensemble des tâches qui commencent dans l'intervalle $[s_{j-1} + \Delta_{j-1}, s_j]$. On rappelle que par convention, on pose $s_0 = \Delta_0 = 0$ et $s_{\mathcal{K}+1} = +\infty$. On dit qu'une partition Π est une *représentation* d'un ordonnancement si il existe un ordonnancement S tel que $\Pi = \phi(S)$. On peut remarquer qu'un ordonnancement définit une partition unique ϕ, mais une partition Π peut être la représentation de plusieurs ordonnancements, ou d'aucun. Pour déterminer si une partition Π représente un ordonnancement, on définit les quantités (R_j) par :

$$\begin{cases} R_1 = 0 \\ R_j = \max\{R_{j-1} + p(\Pi_{j-1}),\ s_{j-1} + \Delta_{j-1}\}, \quad j = 2, 3, \ldots, \mathcal{K}+1 \end{cases}$$

R_j représente l'instant de départ au plus tôt de Π_j. On appelle *valeur de la partition* la quantité $\max\{R_l + p(\Pi_l) | \Pi_l \neq \emptyset\}$. Pour un sous-ensemble J d'indices, on note $p_{\max}(J)$ la

plus grande durée d'usinage de l'ensemble $\{p_j | j \in J\}$, c'est-à-dire $p_{\max}(J) = \max_{j \in J} p_j$. On dit qu'une partition Π est *valide* si et seulement si

$$R_{j+1} - \max\{p_{\max}(\Pi_j),\, \Delta_j\} \leq s_j, \quad \text{pour tout } j = 1,\, 2,\, \ldots,\, \mathcal{K}$$

On a la propriété suivante :

Proposition 8 *Une partition* $\Pi = (\Pi_j)_{j=1}^{\mathcal{K}+1}$ *est la représentation d'un ordonnancement si et seulement si* Π *est une partition* valide. *De plus, si* Π *représente un ordonnancement semi-actif* S*, alors la valeur de* Π *est* $Cmax(S)$.

Démonstration.

En se basant sur une partition Π, nous pouvons construire un ordonnancement S en séquençant chaque ensemble Π_j de façon semi-active, en commençant à l'instant $R_j \geq s_{j-1} + \Delta_{j-1}$. On choisit comme séquence pour Π_j n'importe quelle séquence qui place la plus longue tâche en dernière position. L'ordonnancement S est alors réalisable et admet Π comme représentation si et seulement si la condition est remplie, c'est-à-dire si et seulement si lorsque la plus longue tâche de Π_j est plus longue que Δ_j, elle peut commencer avant l'instant s_j et lorsqu'elle est strictement plus courte que Δ_j, elle peut finir avant l'instant s_j. □

Ainsi, trouver un ordonnancement optimal équivaut à trouver une partition valide de valeur minimale. Remarquons que déterminer si une partition Π est valide et déterminer sa valeur peut être fait en $O(\mathcal{K})$ si on fournit pour chaque ensemble la somme des durées d'usinage et son plus grand élément.

Nous proposons maintenant un programme dynamique qui fournit une partition optimale. Supposons que les tâches sont ordonnées par durées d'usinage décroissantes pour cette preuve uniquement. Pour deux vecteurs P et q, on définit la formule booléenne $\chi(i, P, q)$, comme étant vraie si et seulement si il existe une partition valide Π de $\{1,\, 2,\, \ldots,\, i\}$ telle que $P_j = p(\Pi_j)$ et q_j est l'indice de la plus longue tâche de Π_j, pour $j = 1,\, 2,\, \ldots,\, \mathcal{K}+1$. Par convention, $q_j = 0$ si $P_j = 0$, c'est-à-dire que le $j^{ième}$ ensemble de la partition est vide. Le calcul de $\chi(i, P, q)$ se fait de la façon suivante. Si χ est vraie, $\Pi \backslash \{i\}$ doit être une partition valide de $\{1,\, 2,\, \ldots,\, i-1\}$. Remarquons que i est le plus grand élément d'un ensemble si et seulement si cet ensemble est précisément $\{i\}$. Réciproquement, si on considère une partition Π' de $\{1,\, 2,\, \ldots,\, i-1\}$, on a $\mathcal{K}+1$ choix pour l'étendre en une partition valide Π de $\{1,\, 2,\, \ldots,\, i\}$, dépendant de l'intervalle $[s_{j-1} + \Delta_{j-1},\, s_j]$ dans lequel on décide d'ordonnancer la tâche T_i. Nous avons donc l'équation de récurrence suivante :

$$\chi(i, P, q) = valide(i, P, q) \wedge \bigvee_{l=1}^{\mathcal{K}+1} \chi(i-1, P^l, q^l)$$
$$\text{avec } P^l = (P_1, \ldots, P_l - p_i, \ldots, P_{\mathcal{K}+1}) \text{ et } q^l = (q_1, \ldots, q_l.1_{\{q_l \neq i\}}, \ldots, q_{\mathcal{K}+1})$$

Le prédicat $valide(i, P, q)$ indique si la partition est valide. Il est vrai si et seulement si $R_{j+1} - \max\{p_{q_j},\, \Delta_j\} \leq s_j$ pour tout $j = 1,\, 2,\, \ldots,\, \mathcal{K}$. De plus, il faut que tous les P_j soient positifs et que tous les q_j soient dans $\{0,\, 1,\, \ldots,\, i\}$, avec $q_j = 0$ si et seulement si $P_j = 0$. Ce test peut être réalisé en temps constant $O(\mathcal{K})$. Ainsi, déterminer si $\chi(i, P_j, q_j)$ est vraie en connaissant les $\chi(i-1, ., .)$ demande seulement un nombre constant d'opérations.

En fixant $\chi(0,0,0) = vraie$, on peut calculer dynamiquement les valeurs des χ par induction sur i. Pour un indice donné i, puisque la taille de $P_{\mathcal{K}+1}$ est fixée par les autres ensembles et que $q_{\mathcal{K}+1}$ ne joue aucun rôle, on a $O(p(\{1, 2, \ldots, i\})^{\mathcal{K}} i^{\mathcal{K}})$ différents vecteurs (P, q) à tester. Ainsi, l'algorithme trouve toutes les partitions valides en temps $O(p(N)^{\mathcal{K}} n^{\mathcal{K}+1})$ avec un espace mémoire de $O(p(N)^{\mathcal{K}} n^{\mathcal{K}})$. Il est suffisant de renvoyer la meilleure valeur d'une partition trouvée parmi tous les $\chi(n, ., .)$ dont la valeur est $vraie$. □

Le programme dynamique précédent, bien que pseudo-polynomial, s'avérerait trop lent pour être utilisé en pratique sur des instances de grandes tailles. Nous nous tournons à la Section 4.3 vers des heuristiques polynomiales qui peuvent se montrer très efficaces pour les cas concrets.

Avant de nous intéresser plus en détail à ces heuristiques, nous prouvons que le problème est $\mathcal{NP}$-Difficile au sens fort si le nombre de périodes fait partie de l'instance, et ce même si les périodes sont petites.

4.2.2 Problème $1|ona|Cmax$

Rappelons que le problème $1|ona|Cmax$ décrit le problème d'ordonnancement pour lequel le nombre de périodes n'est pas une constante, mais fait partie de l'instance. Ce problème est bien sûr au moins aussi difficile que $1|ona(\mathcal{K})|Cmax$ et est donc $\mathcal{NP}$-Difficile. Le théorème suivant précise que $1|ona|Cmax$ est $\mathcal{NP}$-Difficile au sens fort.

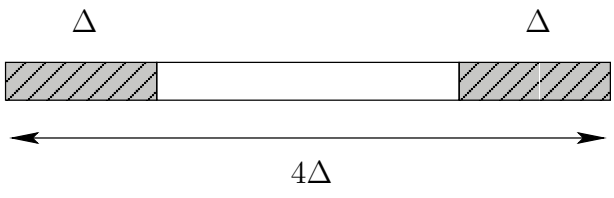
Proposition 9 *Le problème $1|ona|Cmax$ est $\mathcal{NP}$-Difficile au sens fort, même si toutes les périodes* ONA *sont égales et petites.*

Démonstration. Considérons le problème de décision suivant :

3-PARTITION
INSTANCE : $3m+1$ entiers $a_1, a_2, \ldots, a_{3m}$ et B vérifiant $\sum_{i=1}^{3m} a_i = mB$ et $B/4 < a_i < B/2$ $\forall i = 1, 2, \ldots, 3m$;
QUESTION : Existe-t-il une partition $A_1 \cup A_2 \cup \cdots \cup A_m$ de $[1, 3m]$ telle que $\sum_{i \in A_j} a_i = m, \ \forall j = 1, 2, \ldots, m$?

Sans perte de généralité, on peut supposer que $B = 0 \bmod 4$ (sinon, on multiplie tous les entiers par 4). On code l'instance $\mathcal{I}$ de 3-PARTITION en une instance $f(\mathcal{I})$ du problème de décision associé à $1|ona|Cmax$ de la façon suivante :

– On a $\mathcal{K} = 2m$ périodes de durées $\Delta = B/4$. Elles reproduisent le motif suivant toutes les 4Δ unités de temps :



- On a $n = 3m$ tâches, de durées $p_i = a_i$. Remarquons que $\Delta < p_i < 2\Delta$ pour tout $i = 1, 2, \ldots, 3m$ et que les périodes ONA sont donc petites.
- QUESTION : Existe-t-il un ordonnancement de makespan inférieur ou égal à $C = \sum_i p_i$?

La transformation est une réduction polynomiale. Considérons une instance I du problème 3-PARTITION pour laquelle la réponse est "oui". Pour prouver qu'il existe une solution admissible pour $f(I)$, il suffit de montrer que les tâches associées à chaque ensemble A_j de la partition peuvent être ordonnancées exactement dans un motif. En fait, les périodes étant petites, toute séquence de trois tâches dont la somme des durées est $B = 4\Delta$ est valide.

Réciproquement, supposons que la réponse à l'instance $f(I)$ est "oui". Un ordonnancement de makespan C n'a aucun temps mort. De plus, chaque tâche est ordonnancée dans un motif (elle commence et se termine dans le même motif), puisqu'à la frontière entre deux motifs, on a deux périodes adjacentes de durée totale $2\Delta > p_i$ pour tout i. Cela implique que les tâches ordonnancées à l'intérieur de chaque motif ont une durée de $4\Delta = B$ exactement. L'ordonnancement définit alors une partition valide pour I.

La réduction est pseudo-polynomiale. Ainsi, puisque 3-PARTITION est $\mathcal{NP}$-Difficile au sens fort, cela implique que $1|ona|Cmax$ est aussi $\mathcal{NP}$-Difficile au sens fort. □

On peut utiliser la réduction précédente pour établir des résultats d'inapproximabilité pour $1|ona|Cmax$:

Proposition 10 *Si $\mathcal{P} \neq \mathcal{NP}$, le problème $1|ona|Cmax$ n'est pas dans $\mathcal{APX}$, même si toutes les périodes sont égales et petites. De plus, approximer $1|ona|Cmax$ avec un facteur $\mathcal{K}^{1-\varepsilon}$ est $\mathcal{NP}$-Difficile pour toute constante $\varepsilon > 0$.*

Démonstration. Soit $\varepsilon > 0$ une constante et l'entier $\alpha = \lceil 1/\varepsilon - 1 \rceil$. Considérons de nouveau le problème 3-PARTITION et modifions la réduction f en ajoutant $(4m)^{\alpha+1} - 4m$ périodes après le dernier motif (le "gadget" de la Figure 4.1). L'instance $g(I)$ de $1|ona|Cmax$ qui en résulte a $\mathcal{K} = (4m)^{\alpha+1} - 2m$ périodes au total.

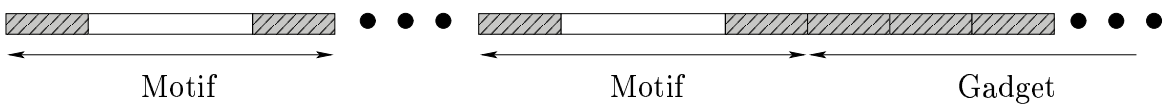


FIG. 4.1 – Périodes ONA pour la preuve de la Proposition 10

La transformation g est polynomiale par rapport à la taille $|I|$ de l'instance I, et $|I| \geq \max\{m, \log \Delta\}$. Puisque f est une réduction, on a clairement :

$$\begin{aligned} I \in \text{3-PARTITION} &\Leftrightarrow Cmax^*(g(I)) = 4m\Delta \\ I \notin \text{3-PARTITION} &\Leftrightarrow Cmax^*(g(I)) \geq (4m)^{\alpha+1}\Delta \end{aligned}$$

où $Cmax^*(g(I))$ est le makespan optimal pour l'instance $g(I)$. Si la réponse à I est "non", un ordonnancement valide pour $g(I)$ doit finir après $4m\Delta$. Ainsi, à cet instant, au moins

une tâche n'a pas pu être usinée. Puisque seules des périodes d'indisponibilité se produisent alors, et que toutes les tâches ont des durées dans $]\Delta, 2\Delta[$, la tâche qui n'a pas pu être usinée avant $4m\Delta$ est nécessairement ordonnancée après la dernière période.

Le théorème du Gap implique qu'approximer $1|ona|Cmax$ à moins de $(4m)^{\alpha} > \mathcal{K}^{\alpha/(\alpha+1)} \geq \mathcal{K}^{\frac{1-\varepsilon}{1+\varepsilon}} \geq \mathcal{K}^{1-\varepsilon}$ est $\mathcal{NP}$-Difficile. □

4.3 Heuristiques et leurs performances

La section précédente a établi que les problèmes $1|ona(\mathcal{K})|Cmax$ et $1|ona|Cmax$ sont tous les deux $\mathcal{NP}$-Difficiles. Il est alors naturel d'explorer des approches heuristiques. Dans la théorie de l'ordonnancement, les plus fameuses sont basées sur les *listes de priorité*, introduites par Graham. On établit dans cette section les performances qu'on peut espérer pour deux types d'algorithmes basés sur des listes et des améliorations substantielles pour certaines classes d'instances utiles en pratique.

Tous les ordonnancements que nous analysons sont calés autant que possible sur la gauche (ordonnancements dits *semi-actifs*). Nous considérons deux types d'algorithmes qui s'appuient sur des listes de priorité. Ils produisent tous deux des ordonnancements semi-actifs.

- *First fit (FF)* : On ordonnance le plus tôt possible la première tâche parmi les tâches restant dans la liste. On peut remarquer que les périodes ONA peuvent créer dans l'ordonnancement partiel des temps morts suffisamment larges pour que cette tâche soit placée avant la fin de l'ordonnancement partiel.
- *List Scheduling (LS)* : Aussitôt que la machine devient libre, on parcourt la liste des tâches restantes et on ajoute à l'ordonnancement partiel, parmi celles pouvant commencer le plus tôt, la tâche de plus grande priorité. Les tâches ne sont donc pas nécessairement placées dans l'ordre de la liste.

Les algorithmes de type FF ne parcourent pas la liste pour trouver la première tâche disponible, cependant, ils permettent à une pièce d'être ordonnancée au sein de l'ordonnancement partiel sans augmenter le makespan. Les algorithmes de type LS ne permettent pas aux périodes d'inactivité de la machine d'être suffisamment longues dans l'ordonnancement partiel pour contenir une des tâches restantes, puisqu'on produit une tâche dès que c'est possible, même si elle n'est pas première de la liste.

Remarquons que tout ordonnancement S semi-actif pour une instance I peut être créé par un algorithme de type FF dont la liste de priorité correspond à la séquence des tâches dans S. Cela inclut donc les ordonnancements produits par tout algorithme de type LS.

Considérons tout d'abord le problème avec périodes d'indisponibilité classiques, c'est-à-dire des périodes durant lesquelles les machines sont indisponibles. On note ce problème $1\,|nre(\mathcal{K})|\,Cmax$, lorsque la préemption n'est pas autorisée et que le makespan est le critère choisi. On sait que pour $\mathcal{K} \geq 2$, le problème $1\,|nre(\mathcal{K})|\,Cmax$ n'est pas approximable avec un ratio constant sauf si $\mathcal{P} \neq \mathcal{NP}$ [25]. La preuve, cependant, permet aux intervalles d'indisponibilité d'être arbitrairement larges. Le problème $1|ona(\mathcal{K})|Cmax$ est équivalent à $1\,|nre(\mathcal{K})|\,Cmax$ sur l'ensemble des instances telles que toute période d'indisponibilité est plus longue que n'importe quelle durée d'usinage. Il faut donc borner la durée

des périodes ONA si nous souhaitons obtenir des heuristiques de performance garantie constante.

4.3.1 Une première borne sur la taille des périodes

Supposons que la durée de chaque période est bornée par la somme des durées d'usinage. Nous choisissons cette valeur car c'est une borne inférieure triviale du makespan.

$$\Delta_q \leq \sum_{j=1}^{n} p_j, \quad \forall q = 1, 2, \ldots, \mathcal{K} \tag{4.1}$$

On dit de toute période qui vérifie l'équation 4.1 qu'elle est *bornée*.

Montrons d'abord que sous cette hypothèse, il existe des algorithmes avec une performance garantie constante pour le problème $1\,|nre(\mathcal{K})|\,Cmax$. Pour cela, nous étudions les performances des algorithmes LS et FF. Nous prenons les mêmes notations pour ce problème que pour les périodes ONA et commençons par quelques remarques générales valables pour les deux problèmes.

Remarque. Par conservation du travail, pour tout ordonnancement S

$$Cmax(S) = \sum_{j=1}^{n} p_j + \sum_{q=1}^{k} X_q \leq \sum_{j=1}^{n} p_j + \sum_{q=1}^{k} x_q + \sum_{q=1}^{k} \Delta_q \tag{4.2}$$

Remarque. Considérons un ordonnancement semi-actif S. Chaque x_q doit être plus petit que la plus longue durée d'usinage, p_n, sinon, nous aurions trouvé une tâche qui remplierait le temps mort correspondant :

$$x_q \leq p_n \leq Cmax(S^*), \quad \forall q = 1, 2, \ldots, k \tag{4.3}$$

Nous pouvons à présent démontrer les propriétés suivantes.

Proposition 11 *Pour le problème $1\,|nre(\mathcal{K})|\,Cmax$ et des périodes bornées, tout algorithme $\mathcal{A}$ produisant uniquement des ordonnancements semi-actifs a une performance garantie de $2\mathcal{K}$, c'est-à-dire*

$$\textit{pour toute instance } I, \ \frac{Cmax(\mathcal{A}(I))}{Cmax^*(I)} \leq 2\mathcal{K}$$

(où $Cmax^*(I)$ est la valeur optimale du makespan pour une instance I) et il existe des listes de priorité pour lesquelles l'algorithme de type LS ou FF associé a un ratio de performance qui atteint cette borne asymptotiquement pour $\mathcal{K} \geq 2$.

Démonstration. Considérons une instance de $1\,|nre(\mathcal{K})|\,Cmax$ et un ordonnancement optimal S^* pour cette instance, de makespan $Cmax(S^*)$.

Considérons un ordonnancement S semi-actif de makespan $Cmax(S)$. L'ordonnancement étant calé à gauche, on peut remarquer qu'il existe au plus une période d'inactivité entre deux périodes d'indisponibilité.

On sait que tout ordonnancement optimal se termine après la première période car nous avons supposé que $\sum_{j=1}^{n} p_j > s_1$. On a

$$x_1 + \Delta_1 \leq s_1 + \Delta_1 \leq Cmax(S^*)$$

On obtient donc avec (4.2),

$$Cmax(S) \leq \sum_{j=1}^{n} p_j + Cmax(S^*) + \sum_{q=2}^{k} \Delta_q + \sum_{q=2}^{k} x_q.$$

En utilisant (4.3) et le fait que les périodes sont bornées, le résultat est immédiat :

$$Cmax(S) \leq Cmax(S^*) + \mathcal{K} \sum_{j=1}^{n} p_j + (\mathcal{K}-1)p_n \leq 2\mathcal{K}\ Cmax(S^*)$$

ce qui nous donne la borne requise.

Pour vérifier que la borne $2\mathcal{K}$ est atteinte pour certaines listes par les algorithmes de type LS et FF associés, considérons l'instance suivante de $1\,|nre(\mathcal{K})|\,Cmax$. Pour une constante positive ε suffisamment petite telle que $2\varepsilon < \Delta$ (Figure 4.2) :
- deux tâches T_1 et T_2 telles que $p_1 = \varepsilon$ et $p_2 = \Delta$
- $\Delta_1 = \varepsilon$ et $\Delta_q = \Delta + \varepsilon$ pour $2 \leq q \leq \mathcal{K}$
- $s_1 = \Delta$ et $s_q = s_{q-1} + \Delta_{q-1} + \Delta - \epsilon$ pour $2 \leq q \leq \mathcal{K}$

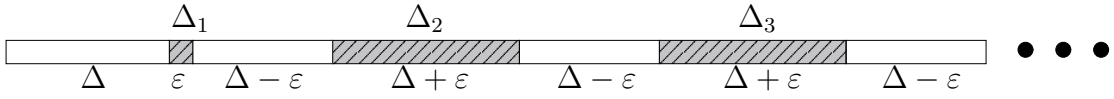


FIG. 4.2 – Périodes ONA pour la preuve de la Proposition 11

Dans un ordonnancement optimal S^*, la tâche T_2 est placée avant la première période d'indisponibilité, et la tâche T_1 est ordonnancée immédiatement après cette période, ainsi $Cmax(S^*) = \Delta + 2\varepsilon$. Cependant, un algorithme basé sur la liste de priorité (T_1, T_2) (qu'il soit de type LS ou FF) créera l'ordonnancement S dans lequel la tâche T_2 suit la tâche T_1. Dans un tel ordonnancement, la tâche T_2 peut commencer uniquement après la dernière période d'indisponibilité, de telle sorte que $Cmax(S) = 2\Delta(\mathcal{K}-1)+(\Delta+\varepsilon)+\Delta = 2\Delta\mathcal{K}+\varepsilon$. Lorsque $\varepsilon \to 0$, le ratio $Cmax(S)/Cmax(S^*)$ tend vers $2\mathcal{K}$. □

Puisqu'ordonnancer avec des périodes ONA ouvre plus de possibilités (les périodes ONA peuvent être couvertes par une tâche), on peut s'attendre à une amélioration des performances des algorithmes semi-actifs ou des algorithmes basés sur des listes de priorité par rapport au cas classique de périodes d'indisponibilité des machines. La proposition suivante montre que ce n'est pas le cas pour les ordonnancements semi-actifs.

Proposition 12 *Pour le problème $1\,|ona(\mathcal{K})|\,Cmax$ avec des périodes bornées, les ordonnancements semi-actifs ont une performance garantie de $2\mathcal{K}$ et si $\mathcal{K} \geq 2$, alors il existe des algorithmes de type FF dont le ratio de performance est $2\mathcal{K}$.*

Démonstration. Considérons un ordonnancement S calé à gauche de makespan $Cmax(S)$ et un ordonnancement optimal S^*. On a sait que

$$\begin{aligned} x_1 + \Delta_1 &\leq \sum_{j=1}^{n} p_j \leq Cmax(S^*) \\ \Delta_q &\leq \sum_{j=1}^{n} p_j \leq Cmax(S^*) \quad \text{pour tout } q = 1, 2, \ldots, \mathcal{K} \end{aligned}$$

ce qui implique en utilisant les équations (4.2) et (4.3) que

$$Cmax(S) \leq Cmax(S^*) + Cmax(S^*) + 2(k-1)Cmax(S^*) \leq 2\mathcal{K}Cmax(S^*)$$

On montre maintenant qu'il existe une instance et une liste pour lesquelles l'ordonnancement créé par l'algorithme FF atteint cette borne.

Considérons l'instance suivante du problème $1|ona(\mathcal{K})|Cmax$ (Δ arbitraire et ϵ suffisamment petit, $4\epsilon < \Delta$, voir Figure 4.3) :

- $p_1 = \epsilon, \quad p_2 = \Delta - \epsilon/2$
- $\Delta_1 = \Delta - \epsilon, \quad \Delta_q = \Delta \quad$ pour $2 \leq q \leq \mathcal{K}$
- $s_1 = 0, \quad s_q = s_{q-1} + \Delta_{q-1} + \Delta - 2\epsilon \quad$ pour $2 \leq q \leq \mathcal{K}$

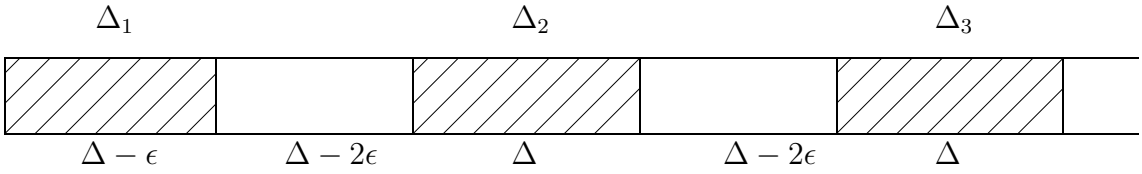


FIG. 4.3 – Périodes ONA pour la preuve de la Proposition 12

La solution optimale est donnée par la séquence (T_2, T_1), où T_2 commence à l'instant zéro, avec un makespan $\Delta + \frac{\epsilon}{2} = \sum_{j=1}^{n} p_j$.

Avec l'algorithme FF et la liste de priorité (T_1, T_2), on obtient la séquence (T_1, T_2) qui a un makespan égal à $s_{\mathcal{K}} + \Delta + p_2$ puisque la tâche T_2 est trop petite pour couvrir les périodes ONA 2 à $\mathcal{K}$ et est trop grande pour être placée entre deux de ces périodes. Alors $Cmax(T_1, T_2) = \Delta - \epsilon + (\mathcal{K}-1)(2\Delta - 2\epsilon) + \Delta - \epsilon/2 = 2\mathcal{K}(\Delta - \epsilon) + \epsilon/2$.

Ainsi, on a

$$\frac{Cmax(T_1, T_2)}{Cmax(T_2, T_1)} = \frac{2\mathcal{K}(\Delta - \epsilon) + \epsilon/2}{\Delta + \epsilon/2} \longrightarrow_{\epsilon \to 0} 2\mathcal{K}$$

Ce qui permet de conclure. □

On note ρ_L le ratio de performance de l'algorithme de type LS associé à la liste L.

Théorème 8 *Si $\mathcal{K} \geq 4$, pour toute liste de priorité L, on a $\rho_L = 2(\mathcal{K}-1)$.*

Nous montrons ce théorème à l'aide de plusieurs lemmes.

Lemme 5 *Pour toute liste de priorité L et tout $\mathcal{K} \geq 2$,*

$$\rho_L \geq 2(\mathcal{K}-1)$$

Démonstration. Considérons l'instance suivante avec Δ quelconque et ϵ et suffisamment petit (voir Figure 4.4) :

- $p_1 = \epsilon$, $p_2 = \Delta + \epsilon/2$
- $\Delta_1 = \Delta$, $\Delta_2 = \epsilon$, $\Delta_q = \Delta + \epsilon$ pour $3 \leq q \leq \mathcal{K}$
- $s_1 = \epsilon$, $s_2 = \Delta + \epsilon$, $s_3 = 2\Delta + \epsilon$, $s_q = s_{q-1} + 2\Delta + \epsilon \quad \forall q \geq 4$

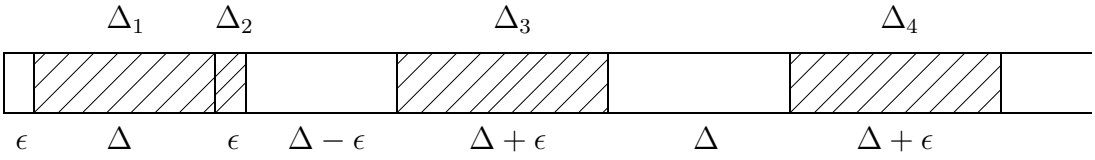


FIG. 4.4 – Périodes ONA pour la preuve du Lemme 5

La solution optimale est donnée par la séquence (T_2, T_1), où T_2 commence à l'instant $\epsilon/2$. Ce temps mort de durée $\epsilon/2$ au début de l'ordonnancement (alors que l'on pourrait produire T_1 à l'instant 0) implique qu'aucun algorithme de type LS, quelle que soit la liste de priorité sur laquelle il est basé, ne peut produire cette séquence. Elle a un makespan de $\Delta + 2\epsilon$.

En appliquant un algorithme LS, on obtient la séquence (T_1, T_2), quelle que soit la liste de priorité considérée. On a alors

$$Cmax(T_1, T_2) = (\mathcal{K} - 1)(2\Delta + \epsilon) + \epsilon/2 \quad \text{pour tout } \mathcal{K} \geq 2$$

Ainsi, $\frac{Cmax(T_1,T_2)}{Cmax(T_2,T_1)} \longrightarrow_{\epsilon \to 0} 2(\mathcal{K} - 1)$ pour tout $\mathcal{K} \geq 2$. Le ratio de performance de tout algorithme de type LS est donc supérieur ou égal à $2(\mathcal{K} - 1)$ pour tout $\mathcal{K} \geq 2$. □

Lemme 6 *Soit I une instance de $1|ona(\mathcal{K})|Cmax$ telle que l'ordonnancement optimal S^* finit après la seconde période* ONA. *Pour tout algorithme $\mathcal{A}$ de type LS on a $R(I, \mathcal{A}(I)) \leq 2(\mathcal{K} - 1)$ pour tout $\mathcal{K} \geq 2$.*

Démonstration. Soit I une instance de $1|ona(\mathcal{K})|Cmax$ telle que l'ordonnancement optimal S^* finit après la seconde période ONA. On considère l'ordonnancement S créé par un algorithme de type LS. On a $x_1 + \Delta_1 + x_2 + \Delta_2 \leq Cmax(S^*)$, et d'après (4.2) et (4.3) :

$$Cmax(S) \leq \sum_{j=1}^{n} p_j + Cmax^* + \sum_{q=3}^{\mathcal{K}} (x_q + \Delta_q) \leq (2 + 2(\mathcal{K} - 2))Cmax(S^*).$$

ce qui complète la preuve. □

Il nous reste à étudier le cas où $s_1 + \Delta_1 \leq Cmax(S^*) \leq s_2$, avec $s_2 = +\infty$ si $\mathcal{K} = 1$.

Lemme 7 *Étant donnée une instance I de $1|ona(\mathcal{K})|Cmax$, si l'ordonnancement optimal S^* se termine entre les deux premières périodes* ONA *(c'est-à-dire $s_1 + \Delta_1 \leq Cmax(S^*) \leq s_2$), alors pour tout algorithme $\mathcal{A}$ de type LS, on a $R(I, \mathcal{A}(I)) \leq 3\mathcal{K}/2$ pour tout $\mathcal{K} \geq 1$.*

Démonstration. Soit I une instance de $1|ona(\mathcal{K})|Cmax$, S^* l'ordonnancement optimal pour cette instance et $\mathcal{A}$ un algorithme de type LS. Soit T_l la dernière tâche dans l'ordonnancement S produit par $\mathcal{A}$ pour l'instance I. On sait que la tâche T_l doit être plus longue que toute période d'inactivité x_q, pour tout $q \leq k$.

On montre tout d'abord que si l'ordonnancement S n'est pas optimal, alors nécessairement $p_l \leq \frac{1}{2}Cmax(S^*)$. On considère 2 cas :

1. La première période ONA est couverte par S. On remarque que si aucun temps mort ne se produit avant s_1 ($x_1 = 0$), alors S est optimal puisque $\sum_{j=1}^{n} p_j \leq s_2$. Sinon, soit t le premier instant où apparaît un temps mort. Observons les tâches qui n'ont pas encore été usinées à l'instant t. Pour toute tâche T_j dans ce cas, on ne peut pas avoir $t + p_j > s_2$ sans contredire le fait que $Cmax(S^*) \leq s_2$. Ainsi, toute tâche T_j plus longue que Δ_1 peut être ordonnancée au plus tôt à l'instant $s_1 + \Delta_1 - p_j$, en couvrant la période ONA. Cela implique que la tâche T_β qui couvre Δ_1 est nécessairement la plus longue des tâches restantes. Remarquons que les tâches T_β et T_l sont différentes, puisque sinon $Cmax(S^*) = s_1 + \Delta_1 = Cmax(S)$. Ainsi,

$$Cmax(S^*) \geq p_\beta + p_l \geq 2p_l$$

2. La première période ONA n'est pas couverte par S. Comme il n'était pas possible d'ordonnancer T_l pour qu'elle couvre Δ_1, on a $p_l < \Delta_1$ en suivant le raisonnement précédent. Si l'ordonnancement optimal ne couvre pas non plus cette période, alors $Cmax(S^*) \geq \sum_{j=1}^{n} p_j + \Delta_1 \geq 2p_l$. Sinon, il existe une tâche T_α plus longue que Δ_1. Une fois encore, $Cmax(S^*) \geq p_\alpha + p_l \geq 2p_l$.

En utilisant (4.2) et (4.3), on peut borner le makespan de S comme suit :

$$Cmax(S) \leq \sum_{j=1}^{n} p_j + X_1 + \sum_{q=2}^{k}(x_q + \Delta_q) \leq \sum_{j=1}^{n} p_j + X_1 + (\mathcal{K} - 1)\left(p_l + \sum_{j=1}^{n} p_j\right)$$

Clairement $X_1 = x_1 + \Delta_1 \leq Cmax(S^*)$, ce qui implique un ratio de performance de I par rapport à $A(I)$ de $\frac{3}{2}\mathcal{K} + \frac{1}{2}$. Mais il est possible d'être un peu plus précis : en fait, $\sum_{j=1}^{n} p_j + X_1 \leq 2\,Cmax(S^*) - p_l$ si S n'est pas optimal. Pour voir ceci, considérons les différents cas possibles.

- Si la première période ONA est couverte dans S, alors on a simplement $X_1 \leq x_1 \leq p_l$. En utilisant le fait que $p_l \leq \frac{1}{2}Cmax(S^*)$, on obtient $\sum_{j=1}^{n} p_j + X_1 \leq Cmax(S^*) + p_l \leq 2\,Cmax(S^*) - p_l$;
- Si la première période ONA n'est pas couverte par l'ordonnancement optimal, alors $Cmax(S^*) \geq \sum_{j=1}^{n} p_j + \Delta_1$. Ainsi, $\sum_{j=1}^{n} p_j + X_1 \leq Cmax(S^*) + x_1 \leq Cmax(S^*) + p_l \leq 2Cmax(S^*) - p_l$;
- Si la première période ONA est couverte dans l'ordonnancement optimal S^* et pas dans S. En utilisant les notations précédentes, il existe une tâche T_α plus grande que Δ_1. Puisque $Cmax(S^*) \leq s_2$, toute tâche plus grande que Δ_1 peut être ordonnancée à un certain instant entre t, le début de la période d'inactivité, et s_1. La première période ONA n'est pas couverte dans S, ce qui implique que T_α a déjà été ordonnancée et que $p_l \leq p_\alpha$. On obtient $Cmax(S^*) \geq s_1 + \Delta_1 \geq p_\alpha + x_1 + \Delta_1 \geq p_l + X_1$, et le résultat est immédiat.

En rassemblant ces résultats on obtient

$$Cmax(S) \leq (2\,Cmax(S^*) - p_l) + (\mathcal{K}-1)(p_l + Cmax(S^*)) \leq \frac{3}{2}\mathcal{K}\,Cmax(S^*)$$

□

Puisque $3\mathcal{K}/2 \leq 2(\mathcal{K}-1)$ pour tout $\mathcal{K} \geq 4$, le Lemme 7 complète la preuve du Théorème 8.

En résumé, on obtient les résultats suivants

Théorème 9 *Pour le problème $1|ona(\mathcal{K})|Cmax$, quelle que soit la liste de priorité L, le ratio de performance ρ_L de l'algorithme de type LS associé vérifie*
- *Si $\mathcal{K} = 1$, alors $\rho_L \leq 3/2$;*
- *Si $\mathcal{K} = 2$, alors $2 \leq \rho_L \leq 3$;*
- *Si $\mathcal{K} = 3$, alors $4 \leq \rho_L \leq 4.5$;*
- *Si $\mathcal{K} \geq 4$, alors $\rho_L = 2(\mathcal{K}-1)$.*

> **Remarque.** Pour $\mathcal{K} = 1$ et une instance I, il existe toujours une liste telle que l'algorithme de type LS associé donne la solution optimale. Dans le cas où la période est petite, cette liste est directement fournie par l'Algorithme 2. Si la période n'est pas petite, il suffit de considérer un ordonnancement optimal S^* calé à gauche. Si aucun temps mort n'intervient dans S^* on a fini. Sinon supposons qu'un temps mort se produise à l'instant $t \leq s$. Le Lemme 4 implique que si la période ONA est couverte, on peut considérer que c'est par la plus longue tâche. Alors, elle finit en $s + \Delta$ et il n'est pas possible de la faire commencer plus tôt. Dans tous les cas, on ne peut produire en t aucune des tâches produites après $s+\Delta$ puisque cela contredirait le fait que l'ordonnancement est optimal. Il n'est donc pas possible d'obtenir une borne inférieure du ratio de performance valable pour tout algorithme de type LS lorsque $\mathcal{K} = 1$.

On peut remarquer que pour $\mathcal{K} = 2$, un algorithme de type LS, quelle que soit la liste qui lui est associée, ne peut avoir un ratio de performance inférieur à 2. C'est en fait le cas de tout heuristique polynomiale de $1|ona(2)|Cmax$.

Théorème 10 *Pour le problème $1|ona(2)|Cmax$, aucun algorithme polynomial n'a une performance garantie inférieure ou égale à 2, sauf si $\mathcal{P} = \mathcal{NP}$: approximer $1|ona(2)|Cmax$ avec un ratio de performance inférieur ou égal à 2 est $\mathcal{NP}$-Difficile.*

Démonstration. Considérons une instance du problème

BIPARTITION
INSTANCE : n entiers, $a_1, a_2, \ldots, a_n$.
QUESTION : Existe-t-il un sous-ensemble J de $\{1, 2, \ldots, n\}$ tel que $\sum_{j\in J} a_j = \sum_i a_j/2$?

On sait que ce problème est $\mathcal{NP}$-Complet. On construit une instance I du problème $1|ona(2)|Cmax$ dont les périodes sont bornées à partir d'une instance du problème BIPARTITION de la façon suivante

- On considère $n+1$ tâches de durées d'usinage $p_j = a_j$, pour tout $j \in \{1, 2, \ldots, n\}$, et $p_{n+1} = \max_j a_j + 1$;
- La durée Δ_1 de la première période ONA est $\max_j a_j + 1$;

- La durée Δ_2 de la seconde période est $\sum_{j=1}^{n+1} p_j$;
- La première période ONA commence à l'instant $s_1 = \frac{\sum_j a_j}{2}$;
- La seconde période ONA commence à l'instant $s_2 = \Delta_1 + \sum_j a_j = \sum_{j=1}^{n+1}$.

Le problème BIPARTITION a une solution pour l'instance $\{a_1, a_2, \ldots, a_n\}$ si et seulement si il existe un ordonnancement pour l'instance I dont le makespan C vérifie $C = \sum_{j=1}^{n+1} p_j$ puisque T_{n+1} est la seule tâche qui peut couvrir la période ONA.

De plus, le problème BIPARTITION n'a pas de solution pour l'instance $\{a_1, a_2, \ldots, a_n\}$ si et seulement si, dans l'ordonnancement optimal pour l'instance I, au moins une tâche T_β est ordonnancée après $s_2 + \Delta_2$. Cela implique que le makespan optimal pour I est plus grand que $s_2 + \Delta_2 + \min p_j$ et on a alors $Cmax(I) > 2C$. Ainsi, d'après le Théorème du Gap, approximer $1|ona(2)|Cmax$ avec un ratio inférieur ou égal à 2 est $\mathcal{NP}$-Difficile. □

Ces performances sont constantes dans le cas où $\mathcal{K}$ ne fait pas partie de l'instance, mais ne le sont plus pour le problème $1|ona|Cmax$. La section suivante montre que si la borne sur la durée des périodes est renforcée, on peut, pour ce problème, obtenir une garantie constante pour tout algorithme de type LS.

4.3.2 Périodes λ-bornées

Jusqu'ici, nous avons considéré des périodes bornées par $\sum_{j=1}^{n} p_j$. Cette supposition est assez faible puisque pour certaines instances la vérifiant, aucune période ne peut être couverte. Les faibles performances des algorithmes de listes sont peut-être en partie dues à ce fait. Afin d'augmenter le nombre de périodes couvertes, nous définissons la condition suivante que nous appelons *périodes λ-bornées* :

$$\sum_{j=1}^{n} p_j \geq \lambda \ \Delta_q, \quad q = 1, 2, \ldots, \mathcal{K} \tag{4.4}$$

La théorie que nous avons développée précédemment correspond donc à des périodes 1-bornées.

Nous introduisons la définition suivante. Une tâche qui peut être placée de façon à terminer dans l'intervalle $[0, s_\mathcal{K}]$ est dite *ordonnançable*.

Lemme 8 *Considérons une instance I. Si il existe une tâche qui n'est pas ordonnançable, tout algorithme $\mathcal{A}$ qui produit des ordonnancements semi-actifs a une performance garantie de 2 pour l'instance I.*

Démonstration. L'existence d'une tâche qui n'est pas ordonnançable implique que $Cmax(S^*) \geq s_\mathcal{K} + \Delta_\mathcal{K}$. Ainsi, $Cmax(\mathcal{A}(I)) \leq s_\mathcal{K} + \Delta_\mathcal{K} + \sum_{j=1}^{n} p_j \leq 2Cmax(S^*)$. □

Lemme 9 *Considérons une instance I. Si les périodes* ONA *de I sont λ-bornées et que λ est suffisamment grand, tout algorithme $\mathcal{A}$ qui produit des ordonnancements semi-actifs a un ratio de performance de 2 pour l'instance I.*

Démonstration. Posons $\lambda > s_{\mathcal{K}}/max_q\Delta_q$. Alors, pour tout ordonnancement optimal S^*, $Cmax(S^*) \geq \sum_{j=1}^{n} p_j > (s_{\mathcal{K}}/\max_q \Delta_q)\max_q \Delta_q = s_{\mathcal{K}}$ et $Cmax(S^*) \geq s_{\mathcal{K}} + \Delta_{\mathcal{K}}$. Alors, $Cmax(\mathcal{A}(I)) \leq s_{\mathcal{K}} + \Delta_{\mathcal{K}} + \sum_{j=1}^{n} p_j \leq 2Cmax(S^*)$. □

Théorème 11 *Pour le problème* $1|ona(\mathcal{K})|Cmax$ *restreint aux instances dont les périodes sont* λ*-bornées, tout algorithme de type LS a une performance garantie de* $1 + \frac{2\mathcal{K}}{\lambda}$.

Démonstration. Considérons une instance I λ-bornée, une liste de priorité L et l'algorithme de type LS qui lui est associé. Cet algorithme produit l'ordonnancement S pour l'instance I et on note S^* l'ordonnancement optimal pour cette instance. Soit t l'instant de départ d'une période d'inactivité de S de longueur X_i située dans l'intervalle $[s_{i-1} + \Delta_{i-1},\ s_i]$. Supposons que la tâche T_j est la première tâche qui est placée après t dans l'ordonnancement S. Étant donné que T_j n'a pas pu commencer à l'instant t, nécessairement $t + p_j = t_l \in]s_l,\ s_l + \Delta_l[$ pour un indice $l \geq i$ (voir Figure 4.5).

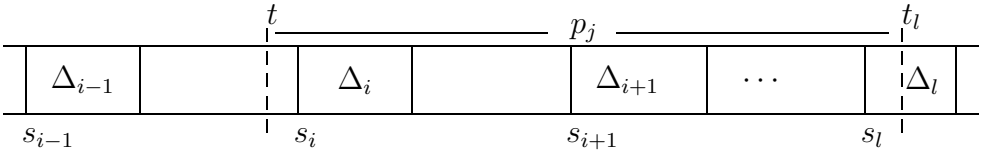


FIG. 4.5 – Période d'inactivité et périodes ONA pour la preuve du Théorème 11

On peut distinguer deux cas.

- La $i^{ième}$ période ONA est couverte par T_j dans S. Dans ce cas, $s_l + \Delta_l - t_l \leq s_i - t$ et la longueur de l'intervalle d'inactivité est $X_i = s_l + \Delta_l - t_l \leq \Delta_l$.
- La $i^{ième}$ période ONA n'est pas couverte par T_j dans S. Dans ce cas, $s_l + \Delta_l - t_l > s_i - t$ et donc $\Delta_l > s_i - t$. Alors, $X_i = s_i - t + \Delta_i \leq \Delta_l + \Delta_i$.

Dans les deux cas, on a donc $X_i \leq 2\max_q \Delta_q$. Ainsi en utilisant l'équation (4.2) et le fait que les périodes sont bornées, on obtient

$$Cmax(S) \leq \sum_{q=1}^{k} X_q + \sum_{j=1}^{n} p_j \leq \left(1 + \frac{2\mathcal{K}}{\lambda}\right) \sum_{j=1}^{n} p_j \leq \left(1 + \frac{2\mathcal{K}}{\lambda}\right) Cmax(S^*)$$

ce qui complète la preuve. □

Corollaire 3 *Pour toute instance* I λ*-bornée telle que* λ *tend vers l'infini, tout algorithme* $\mathcal{A}$ *de type LS a un ratio de performance* $R(I, \mathcal{A}(I))$ *qui tend vers* 1.

Ainsi, si λ est suffisamment grand tout algorithme de liste donnera une solution proche de l'optimal ou optimale.

Dans la section précédente, nous avions $\lambda = 1$. À partir du Théorème 11, nous obtenons une performance garantie de $1 + 2\mathcal{K}$ pour tout algorithme de type LS. Ce résultat est plus faible que celui énoncé par le Théorème 8 mais s'en approche.

Dans le cas où toutes les périodes sont petites, on obtient immédiatement la majoration $\lambda \geq n$ puisque chaque tâche est plus longue que n'importe quelle période ONA. On en déduit le corollaire suivant

Corollaire 4 *Pour le problème* $1|ona(\mathcal{K})|Cmax$ *avec des périodes* ONA *petites, les algorithmes de type LS ont une performance garantie de* $1 + 2\mathcal{K}/n$.

Lorsque que le nombre de tâches tend vers l'infini, les algorithmes de type LS donnent des solutions dont le makespan tend vers l'optimal pour ce type d'instances.

De plus, sur l'ensemble des instances telles que $n \geq \mathcal{K}$, ils ont une performance garantie constante.

Corollaire 5 *Pour toute instance du problème* $1|ona|Cmax$ *où les K périodes* ONA *sont petites et moins nombreuses que les tâches, tout algorithme de type LS a un ratio de performance de* 3.

Lorsque $2 \leq n < \mathcal{K}$, les propositions suivantes indiquent que le ratio de performance des algorithmes de type LS est de l'ordre de $\mathcal{K}/2$.

Proposition 13 *Pour le problème* $1|ona(\mathcal{K})|Cmax$ *avec des périodes* ONA *petites, tout algorithme de type LS a une performance garantie de* $(\mathcal{K}+1)/2$, *si* $\mathcal{K} \geq 3$.

Démonstration. Considérons un ordonnancement S de makespan $Cmax(S)$ obtenu par un algorithme de type LS et un ordonnancement optimal S^*. Rappelons que l'équation de conservation du travail nous donne

$$Cmax(S) = \sum_{i=1}^{n} p_i + \sum_{j=1}^{k} X_j$$

Remarquons que puisque les périodes ONA sont petites, la dernière période ONA active peut toujours être couverte en ordonnançant la dernière tâche T_l de S, à l'instant s_k.

On donne tout d'abord 3 bornes supérieures sur la durée des périodes d'inactivité.

Lemme 10 *Soient I une instance de* $1|ona(\mathcal{K})|Cmax$ *telle que les périodes d'indisponibilité sont petites et S un ordonnancement semi-actif admissible. Soit* $p_{\min} = \min_j\{p_j\} = p_1$ *la plus petite durée d'usinage. Pour tout* $j \leq k$, *on a la majoration* $x_j \leq p_{\min}$.

Démonstration. De la preuve du Théorème 11, on déduit que si $x_j \neq 0$, alors il existe m tel que $x_j \leq \Delta_m$. Le fait que les périodes soient petites permet de conclure. □

Ce lemme implique que $X_j \leq 2\,p_{\min}$ pour tout j. Si la $j^{ième}$ période ONA n'est pas couverte, on obtient une nouvelle borne supérieure :

Lemme 11 *Soient I une instance de* $1|ona(\mathcal{K})|Cmax$ *telle que les périodes* ONA *sont petites et S un ordonnancement produit par un algorithme de type LS. Si la* $j^{ième}$ *période* ONA *n'est pas couverte dans S, alors* $X_j + x_{j+1} \leq p_l$ *où l est l'indice de la dernière tâche de S.*

Démonstration. Considérons comme précédemment le premier instant t de la période d'inactivité entre la $(j-1)^{ième}$ et la $j^{ième}$ période. L'intervalle d'inactivité est alors $[t,\ s_j]$. Remarquons que $[t,\ s_j]$ peut-être réduit éventuellement au singleton $\{s_j\}$ mais n'est jamais vide. On a comme précédemment $t + p_l \in]s_m,\ s_m + \Delta_m[$ pour un certain indice $m \geq j$. Montrons que $m \geq j+1$. Supposons donc que $m = j$. Dans ce cas, la tâche T_l, qui est plus longue que la période ONA, peut commencer à l'instant $s_j + \Delta_j - p_l \geq t$, ce qui contredit le fait qu'aucune tâche ne peut être ordonnancée pendant la période d'inactivité X_j. On a donc $m \geq j + 1$ et ainsi $p_l > s_{j+1} - t$, ce qui complète la preuve. □

Lemme 12 *Soient I une instance de $1|ona(\mathcal{K})|Cmax$ telle que les périodes* ONA *sont petites et S un ordonnancement produit par un algorithme de type LS. Si l est l'indice de la dernière tâche de S, alors pour tout $j \leq k$, on a $X_j \leq \min\{p_l,\ 2p_{\min}\}$.*

Démonstration. C'est une conséquence des deux lemmes précédents. Si la $j^{ième}$ période ONA n'est pas couverte, le Lemme 11 donne directement $X_j \leq p_l$. Le Lemme 10 indique que $x_j \leq p_{\min}$. Les périodes étant petites, $X_j = x_j + \Delta_j \leq 2p_{\min}$. Sinon, $X_j = x_j$, ce qui est inférieur à $p_{\min} \leq p_l$ grâce au Lemme 10. Dans les deux cas on a $X_j \leq x_j + \Delta_j \leq 2\,p_{\min}$. □

Le lemme suivant est la clé de voûte de la preuve. Il indique que les périodes d'inactivité sont égales en moyenne à $(p_l + p_{\min})/2$.

Lemme 13 *Soient I une instance de $1|ona(\mathcal{K})|Cmax$ telle que les périodes* ONA *sont petites et S un ordonnancement produit par un algorithme de type LS. Si l est l'indice de la dernière tâche de S et m un indice inférieur ou égal à k,*

$$X_m + X_{m+1} + \cdots + X_k \leq (k - m + 1)\frac{p_l + p_{\min}}{2} - \frac{p_l - p_{\min}}{2}$$

Démonstration. Pour deux indices u et v tels que $u \leq v$, on définit le *bloc* B_{uv} comme l'intervalle $[s_{u-1} + \Delta_{u-1},\ s_v + \Delta_v[$. Un bloc est dit *non couvert* si les périodes ONA u, $u + 1, \ldots, v - 1$ ne sont pas couvertes, et la période ONA v l'est. Dans un *bloc couvert*, on demande que chaque période ONA soit couverte. L'intervalle $[s_{m-1} + \Delta_{m-1},\ s_k + \Delta_k]$ peut être partitionné en intervalles couverts et non couverts, puisque la dernière période ONA avant $Cmax(S)$, k, est couverte. Par définition, la durée total des temps morts dans le bloc B_{uv} est $X_u + X_{u+1} + \cdots + X_v$. En notant $\mathcal{X}(B_{uv})$ cette quantité, il suffit de prouver que $\mathcal{X}(B_{uv}) \leq (v - u + 1)(p_l + p_{\min})/2 - (p_l - p_{\min})/2$ pour tout bloc couvert et non couvert.

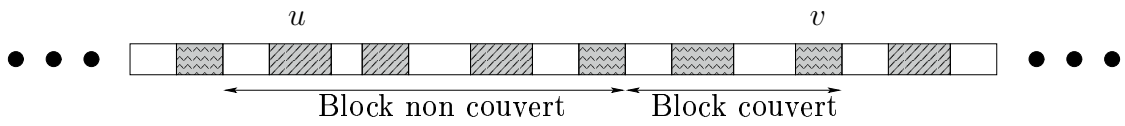


FIG. 4.6 – Blocs couverts et non couverts

Soit B_{uv} un bloc, contenant $q = v - u + 1$ périodes ONA. Considérons tout d'abord que B_{uv} est un bloc couvert. On a pour chaque période d'inactivité $X_j = x_j$. Le Lemme 10 implique que $\mathcal{X}(B_{uv}) \leq q\, p_{\min}$. Puisque $p_{\min} = (p_l + p_{\min} - p_l + p_{\min})/2$, le résultat est vrai pour le bloc.

Supposons maintenant que B_{uv} est un bloc non couvert. Il contient $q-1$ périodes ONA non couvertes, et une période couverte, la dernière. Pour toute période ONA non couverte j, le Lemme 11 permet d'affirmer que $X_j + X_{j+1} = X_j + x_{j+1} + \Delta_{j+1} \leq p_l + p_{\min}$. Si $q-1$ est pair, on a alors $\mathcal{X}(B_{uv}) \leq (q-1)(p_l + p_{\min})/2 + x_v$. On peut conclure en utilisant le Lemme 10, en écrivant simplement que $x_v \leq p_{\min} = (p_l + p_{\min})/2 - (p_l - p_{\min})/2$.

Si $q-1$ est impair, on peut grouper les périodes ONA deux par deux jusqu'à la période $v-1$ qui reste avec la période v. Il s'en suit que $\mathcal{X}(B_{uv}) \leq (q-2)(p_l + p_{\min})/2 + X_{v-1} + x_v$. En utilisant le Lemme 12, on a $X_{v-1} + x_v \leq 3p_{\min}$. Mais le Lemme 11 est vrai pour $v-1$, qui n'est pas couverte, et donc on a aussi $X_{v-1} + x_v \leq p_l$. Il s'en suit que $X_{v-1} + x_v \leq (p_l + 3\,p_{\min})/2 = (p_l + p_{\min}) - (p_l - p_{\min})/2$. □

Notons m le plus petit indice tel que $Cmax(S^*) < s_m$. Pour conclure, on écrit l'équation de la conservation du travail à l'instant $Cmax(S^*)$ pour l'ordonnancement S. Si $\bar{p}$ représente la quantité de travail effectuée à cet instant dans S, on a :

$$Cmax(S) \leq Cmax(S^*) + \sum_{q=m}^{k} X_q + \sum_{j=1}^{n} p_j - \bar{p}$$

Puisque $Cmax(S^*)$ est plus grand que $\sum_j p_j$, en utilisant le Lemme 13 pour les périodes m, $m+1, \ldots, k$, on obtient la majoration :

$$Cmax(S) \leq 2\,Cmax(S^*) + \frac{k-m+1}{2}(p_l + p_{\min}) - (p_l - p_{\min})/2 - \bar{p}$$

Considérons deux cas, dépendants de la valeur optimale. Supposons que $m \geq 3$, c'est-à-dire que l'ordonnancement optimal finit après la seconde période ONA. Montrons qu'à l'instant $Cmax(S^*)$, l'ordonnancement S a terminé au moins sa première tâche T_a. Si r est l'instant auquel S commence cette tâche, aucune tâche parmi les tâches restantes ne peut commencer plus tôt que r dans aucun ordonnancement réalisable d'après la définition de LS. Ainsi, $Cmax(S^*) \geq r + \sum_j p_j \geq r + p_a$. Cela implique en particulier que $\bar{p} \geq p_{\min}$. Puisque nous supposons que $n \geq 2$, on a aussi la majoration $Cmax(S^*) \geq p_l + p_{\min}$. Enfin, on obtient par la conservation du travail :

$$\begin{array}{rcl} Cmax(S) & \leq & 2\,Cmax(S^*) + \frac{k-2}{2}(p_l + p_{\min}) - (p_l - p_{\min})/2 - p_{\min} \\ & \leq & 2\,Cmax(S^*) + \frac{k-3}{2}(p_l + p_{\min}) \\ & \leq & \frac{k+1}{2}\,Cmax(S^*) \leq \frac{\mathcal{K}+1}{2}\,Cmax(S^*) \end{array}$$

Supposons maintenant que l'ordonnancement optimal se termine avant la seconde période ONA. Dans cette situation, on obtient une borne plus large sur le temps mort total de S, puisque le terme $(p_l + p_{\min})/2$ est ajouté à la borne précédente. Ainsi, nous avons besoin d'améliorer la borne inférieure de $\bar{p}$. Plus précisément, nous montrons qu'une quantité

de travail d'au moins $p_{\min} + p_l$ est effectuée dans S à l'instant $Cmax(S^*)$. Tout d'abord, de la discussion précédente, on peut supposer que S commence une tâche à l'instant 0. Soit t_I le premier instant d'inactivité dans S. Observons les tâches que S n'a pas encore ordonnancées à cet instant. Puisque la machine est occupée continuellement dans l'intervalle $[0, t_I[$, pour toute tâche T_h, on a $t_I + p_h \leq \sum_j p_j \leq Cmax(S^*) \leq s_2$. Et puisque T_h n'est pas ordonnancée à l'instant t_I, cela implique que $t_I + p_h \in]s_1, s_1 + \Delta_1[$, c'est-à-dire qu'un temps mort se produit avant la première période ONA (sauf si S est optimal). Toute tâche restante T_h à une date de début au plus tôt de $s_1 + \Delta_1 - p_h$ où elle peut être ordonnancée, couvrant la première période ONA. D'après la définition de LS, l'algorithme ordonnance la plus grande tâche restante, notée T_b, pour qu'elle couvre la première période ONA, et se finisse à l'instant $s_1 + \Delta_1$. Ainsi, à l'instant $Cmax(S^*)$ au moins la première tâche (notée T_a, commençant à l'instant 0) et T_b ont été usinées dans S. On obtient $\bar{p} \geq p_a + p_b \geq p_{\min} + p_l$. L'équation de conservation du travail à l'instant $Cmax(S^*)$ nous donne alors :

$$\begin{array}{rcl} Cmax(S) & \leq & 2\,Cmax(S^*) + \frac{k-1}{2}(p_l + p_{\min}) - (p_l + p_{\min}) \\ & \leq & \frac{k+1}{2} Cmax(S^*) \leq \frac{\mathcal{K}+1}{2} Cmax(S^*) \end{array}$$

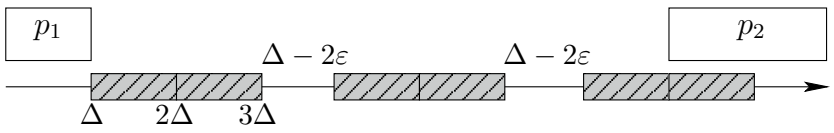
ce qui conclut la preuve. □

Cette performance garantie pour les algorithmes de type LS est acceptable, compte tenu du résultat d'inapproximabilité donné par le Théorème 10. Cependant, de façon surprenante, toutes les listes possèdent la même borne inférieure sur le ratio de performance de l'algorithme.

Proposition 14 *Même si toutes les périodes sont petites, aucun algorithme de liste ne peut avoir une performance garantie meilleure que* $\mathcal{K}/2 + 1/6$ *pour* $\mathcal{K} \geq 3$ *impair, et* $\mathcal{K}/2 + 1/3$ *pour* $\mathcal{K} \geq 4$ *pair.*

Démonstration. Considérons un algorithme $\mathcal{A}$ de type LS et l'instance suivante de $1|ona(\mathcal{K})|Cmax$ pour $\mathcal{K} \geq 4$ pair :
- Les périodes ONA sont groupées deux par deux. Chacune a une durée de Δ ;
- La durée entre deux blocs ONA est $\Delta - 2\epsilon$, sauf pour le premier bloc qui commence à l'instant Δ. Ainsi, la dernière période ONA finit à l'instant $3\Delta\mathcal{K}/2 + O(\varepsilon)$;
- On doit ordonnancer deux 2 tâches, avec $p_1 = \Delta$ et $p_2 = 2\Delta - \epsilon$, et donc $2\Delta - 2\epsilon < p_2 < 2\Delta$.

Si la tâche T_1 est ordonnancée à l'instant 0, il n'est pas possible de placer la tâche T_2 avant la dernière période ONA. Ainsi, le makespan est égal à $3\Delta\mathcal{K}/2 + \Delta - \mathcal{K}\varepsilon/2$.



Mais un ordonnancement optimal consiste à usiner d'abord la tâche T_2, qui commence à l'instant ε, et ensuite la tâche T_1 qui couvre exactement la seconde période ONA. Le makespan est de 3Δ.

Le ratio entre les deux makespans tend vers $\mathcal{K}/2+1/3$ quand ε tend vers 0. Remarquons que tout algorithme LS commence par ordonnancer la tâche T_1 pour éviter le temps mort de ε au début de l'ordonnancement. Ce sera donc le cas de $\mathcal{A}$, et ce quelle que soit la liste qui lui est associée. Cela prouve le résultat pour $\mathcal{K}$ paire. Si $\mathcal{K}$ est impaire, on utilise la même construction, à l'exception du fait que le dernier bloc est remplacé par une unique période Ona. Le makespan de l'ordonnancement produit par $\mathcal{A}$ est alors $3\Delta(\mathcal{K}-1)/2+2\Delta+O(\varepsilon) = 3\Delta\,\mathcal{K}/2+\Delta/2+O(\varepsilon)$. □

Cela montre que pour $1|ona(\mathcal{K})|Cmax$, tout algorithme de type LS a un ratio de performance qui se trouve dans l'intervalle $[\mathcal{K}/2+1/3,\ \mathcal{K}/2+1/2]$. Ainsi, on ne peut pas obtenir de performance garantie constante pour les instances telles que $n < \mathcal{K}$ et $\mathcal{K}$ fait partie de l'instance, même à l'aide d'une liste "intelligente" lorsque nous utilisons des algorithmes de type LS.

Introduisons maintenant une hypothèse supplémentaire pour obtenir un ratio constant même dans le cas $1|ona|Cmax$ pour des périodes Ona petites et plus nombreuses que les tâches. On dit qu'une tâche T_j est *fortement ordonnançable* si pour chaque $q = 1,\ 2, \ldots,\ \mathcal{K}$, il existe un instant $t_q \in [s_{q-1}+\Delta, s_q]$ où peut commencer la tâche T_j (c'est-à-dire que toute tâche peut commencer dans n'importe quelle période). Dans le cas où toutes les tâches sont fortement ordonnançables et les périodes petites, il est possible d'obtenir un meilleur ratio pour les algorithmes de type LS.

Théorème 12 *Pour le problème $1|ona(\mathcal{K})|Cmax$, lorsque toutes les périodes* Ona *sont petites et toutes les tâches sont fortement ordonnançables, tout algorithme de type LS a une performance garantie de 2.*

Démonstration. Soit L une liste de priorité. Soit Y une période d'inactivité commençant à l'instant $t \in [s_{i-1}+\Delta_{i-1},\ s_i]$. $|Y|$ représente la durée totale d'inactivité, même si elle couvre plusieurs périodes Ona. Soit T_l la tâche qui suit la période d'inactivité Y. Il existe un indice $m \geq i$ tel que $t+p_l \in]s_m,\ s_m+\Delta_m[$. On distingue les deux cas suivants :
(1) $m = i$. Dans ce cas, la tâche T_l commence à l'instant $s_i+\Delta_i-p_l$ puisque $p_l \geq \Delta_i$ et la durée de Y vérifie l'inégalité $|Y| \leq s_i - t \leq p_l$.
(2) $m \geq i+1$. Dans ce cas, T_l étant fortement ordonnançable, elle commence dans l'intervalle $[s_i+\Delta_i,\ s_{i+1}]$ et on obtient $|Y| \leq s_{i+1}-t \leq p_l$.
Ainsi, la durée de chaque période d'inactivité Y_u est bornée par une tâche différente. On a donc $\sum |Y_u| \leq \sum_j p_j$. Le makespan de l'ordonnancement S créé par l'algorithme de type LS associé à L vérifie alors $Cmax(S) = \sum |Y_u| + \sum_j p_j \leq 2\,Cmax(S^*)$ où S^* est un ordonnancement optimal. □

Ce théorème est particulièrement utile lorsque les périodes Ona sont périodiques (par exemple se répétant tous les week-ends). Alors les s_q sont de la forme $s_q = q\,L + (q-1)\Delta$ pour tout $q = 1,\ 2, \ldots,\ \mathcal{K}$. Dans ce cas, toutes les tâches ordonnançables sont aussi fortement ordonnançables. On obtient donc le corollaire suivant :

Corollaire 6 *Pour le problème $1|ona(\mathcal{K})|Cmax$ avec des périodes* Ona *petites, de durées identiques et périodiques, tout algorithme de type LS à une performance garantie de 2.*

Démonstration. Pour les instances sans tâche non ordonnaçable, on utilise le Théorème 12. En effet, toute tâche ordonnançable l'est fortement dans le cas périodique. Si il

existe une tâche non ordonnançable, on a $\rho_L \leq 2$ d'après le Lemme 8. □

Remarquons que l'utilisation d'un algorithme de type LS pour le cas périodique reste intéressante puisque d'après le Théorème 7, le problème $1|ona(\mathcal{K})|Cmax$ reste $\mathcal{NP}$-Difficile même si les périodes sont petites, égales et périodiques pour tout $\mathcal{K} \geq 2$.

4.4 Conclusion

Dans ce chapitre, nous avons introduit un nouveau concept en ordonnancement : les périodes d'indisponibilité des opérateurs. Les classes de problèmes qui en résultent offrent plus de possibilités que les périodes d'indisponibilité classiques qui sont relatives aux machines. Nous établissons la complexité du problème général avec périodes d'indisponibilité des opérateurs ainsi que celles de plusieurs cas particuliers pertinents en pratique. Ces problèmes étant en général difficiles, nous étudions les performances de deux types d'algorithmes, LS et FF, utilisant les listes de priorité. Nous bornons les durées des périodes d'indisponibilité pour obtenir des ratios de performance constants pour les algorithmes de type LS quelle que soit la liste choisie pour créer l'algorithme et que le nombre de périodes d'indisponibilité fasse ou non partie de l'instance. Les tables 4.1 et 4.2 résument l'ensemble des résultats de ce chapitre.

Problème	Complexité	Source
$1\|ona(1)\|Cmax$	$\mathcal{NP}$-Difficile au sens faible, admet un $\mathcal{FPTAS}$	Th. 6
$1\|ona(1)\|Cmax$, période ONA petite	Polynomial	Th. 4
$1\|ona(\mathcal{K})\|Cmax$, $\mathcal{K} \geq 2$	$\mathcal{NP}$-Difficile au sens faible	Prop. 7
$1\|ona\|Cmax$	$\mathcal{NP}$-Difficile au sens fort, n'appartient pas à $\mathcal{APX}$	Prop. 9 et 10

TAB. 4.1 – Complexité des problèmes avec périodes ONA et minimisation du makespan

Nombre de périodes	Type de période	Type d'algorithme	Ratio de performance ρ_L pour la liste L	Source
1	1-bornée	LS	$\rho_L \leq 1.5$	Th. 9
$\mathcal{K} \geq 2$	1-bornées	FF	$\rho_L \leq 2\mathcal{K}$	Prop. 11
2	1-bornées	LS	$2 \leq \rho_L \leq 3$	Th. 9
3	1-bornées	LS	$4 \leq \rho_L \leq 4.5$	Th. 9
$\mathcal{K} \geq 4$	1-bornées	LS	$\rho_L = 2(\mathcal{K}-1)$	Th. 8
$\mathcal{K}$	λ-bornées	LS	$\rho_L \leq 1 + \frac{2\mathcal{K}}{\lambda}$	Th. 11
$\mathcal{K}$	petites, $\mathcal{K} \leq n$	LS	$\rho_L \leq 3$	Coro. 5
$\mathcal{K} \geq 3$	petites, $\mathcal{K} \geq n+1$	LS	$\mathcal{K}/2 + 1/6 \leq \rho_L \leq \mathcal{K}/2 + 1/2$	Prop 13 et 14
$\mathcal{K}$	petites, périodiques	LS	$\rho_L \leq 2$	Coro. 14

TAB. 4.2 – Performance des heuristiques avec liste de priorité pour les problèmes avec périodes ONA et minimisation du makespan

Nous nous sommes dans ce chapitre intéressé au critère $Cmax$. Notre problème industriel faisait apparaître un autre critère, $\sum C_j$, qui pourra faire l'objet de futures études. Les

environnements multi-machines et en particulier machines parallèles ont aussi un intérêt pratique et pourront être abordés avec ce type de contraintes d'indisponibilité.

Deuxième partie

Ordonnancement avec tâches agrégées en catégories

1 Introduction

Dans la partie précédente, nous avons décrit des problèmes d'ordonnancement pour lesquels les tâches à effectuer étaient toutes différentes. Alors, la taille d'une instance était un $O(sL)$ où s est le nombre de tâches à effectuer, et L la taille du plus grand de tous les attributs de ces tâches. Si on suppose maintenant que ces tâches ne sont pas toutes différentes, mais qu'il est possible de les agréger en catégories, en $n << s$ types de tâches, alors il est possible d'envisager un codage plus compact des instances. En effet, il n'est plus nécessaire de décrire pour chaque tâche l'ensemble de ses caractéristiques puisqu'il est suffisant de le faire une seule fois pour chaque catégorie de tâches. L'instance du problème sera alors constituée
– du nombre n_i de tâches de type i, pour $i = 1, 2, \ldots, n$;
– des attributs d'une tâche de type i, pour $i = 1, 2, \ldots, n$.
Lorsqu'il est possible de coder les données du problème sous cette forme compacte, on parle d'*ordonnancement de tâches agrégées en catégories* (*high-multiplicity scheduling problem* en anglais) [43]. Ce type de problème se rencontre dans les environnements manufacturiers où des tâches répétitives doivent être effectuées. Plus généralement, n'importe quel problème dont les tâches sont identiques est un problème d'*ordonnancement de tâches agrégées en catégories* puisque on peut considérer que les tâches appartiennent alors a un type unique ($n = 1$). La taille de l'instance est un $O(\log s + L) << O(sL)$. On dira en fait qu'un problème d'ordonnancement est un problème high-multiplicity si s n'est pas polynomial de la taille de l'instance, c'est-à-dire s'il n'existe aucune constante k telle que $n \in O((nL)^k)$ pour toute instance. Des classes de complexité spécifiques aux problèmes d'ordonnancement high-multiplicity sont décrites dans [43, 19].

Dans cette partie, nous nous intéressons plus précisément à deux problèmes dont les tâches sont agrégées en catégories. Le premier est un problème d'ordonnancement juste-à-temps (Chapitre 5) pour lequel les tâches sont agrégées en plusieurs types. Pour ce problème, aucune des modélisations proposées n'est polynomiale (ni le nombre de variable, ni le nombre de contraintes, ni le calcul de la fonction objectif ne sont polynomiaux). Le second est le problème dit des *tâches couplées* (Chapitre 6). Nous étudions la complexité du cas où toutes les pièces sont identiques, l'entrée étant alors composée de trois ou quatre entiers seulement (quatre dans le cas non cyclique, trois dans le cas cyclique).

Chapitre 5

Ordonnancement juste-à-temps

Les systèmes de production Juste-à-Temps (JAT) ont pour but d'ordonnancer la production sur une même ligne de petites quantités de produits de types différents. Le principe de ce modèle est de limiter au maximum les coûts de stockage et les pénalités de retard. Ces systèmes ont été introduits pour les ateliers d'assemblage de voitures de Toyota [70]. Les types de produits sont alors les différentes versions d'un même modèle ou la palette d'options disponibles pour une version.

Lorsqu'une même ligne est utilisée pour l'usinage de produits différents, on inclut en général dans la fonction de coût des pénalités de changement. Par exemple, si on veut produire une voiture rouge alors que la précédente était blanche, il faudra nettoyer les cannons à peinture. En ordonnancement Juste-à-Temps, les types de produits sont supposés suffisamment semblables pour que les coûts de changement d'un type de produit à un autre soient considérés comme nuls (en fait négligeables par rapport aux autres coûts d'usinage). La ligne de production est alors appelée *ligne flexible*. Pour la même raison, les durées de production des différentes pièces sont toutes identiques et fixées à une unité de temps.

Ainsi, la seule donnée du problème est la liste $d_1, d_2, \ldots, d_n$ des demandes des n types de pièces. La taille de cette donnée est donc un $O(n \log(\max d_i))$. Par conséquent, la description complète d'une solution, c'est-à-dire une séquence de production des pièces, n'est pas polynomiale de la taille de l'instance car elle sera de l'ordre de $\sum d_i$. Il est donc difficile de savoir, et ce quelle que soit la fonction objectif, si le problème de décision associé appartient ou non à $\mathcal{NP}$.

Monden [70] considère que le but des systèmes JAT est d'obtenir un ordonnancement aussi *équilibré* que possible, c'est-à-dire de répartir au mieux l'usinage des pièces sur la durée totale de production de sorte que le taux de production de chaque pièce soit aussi constant que possible. De nombreuses façons de mesurer la déviation de la production réelle par rapport à la production idéale parfaitement équilibrée ont été étudiées dans la littérature. Dans un premier temps, nous présenterons rapidement les travaux effectués sur ce sujet et les notations, puis nous décrirons en détail les différentes formulations adoptées pour ce problème en termes de théorie des votes, de recherche d'un couplage parfait et de problème d'affectation de poids minimum. Dans la Section 5.2, nous com-

Les résultats de ce chapitre ont fait l'objet d'un article publié dans EJOR [61]

parons les trois fonctions objectif les plus étudiées et nous réfutons certaines conjectures qui ont été formulées ces dernières années. Nous décrivons aussi des outils et des modèles qui permettent de caractériser les instances pour lesquelles certaines solutions optimisent plusieurs fonctions objectif simultanément. L'Annexe A.1.2 présente des heuristiques polynomiales et les remarques sur lesquelles elles sont basées. Leurs performances n'ayant pas pu être évaluées par manque de temps, elles ne sont pas placées dans ce chapitre. Le reste de l'Annexe A est consacré aux programmes linéaires utilisés pour les tests de la Section 5.2 et aux exemples proposés dans ce chapitre.

5.1 État de l'art et formulations du problème

Dans cette section, nous présentons une vue d'ensemble de la littérature sur l'ordonnancement juste-à-temps avant de faire un exposé plus détaillé des articles les plus significatifs pour le travail effectué dans cette thèse.

L'intérêt pour l'ordonnancement équilibré a vu le jour après la description par Monden [70] des systèmes de production de Toyota. En proposant une formulation en tant que problème d'optimisation, Miltenburg [69] a initié un ensemble considérable de recherche sur ce problème. Cette formulation a pour but de minimiser la somme des déviations (ou *déviation totale*) de la production réelle par rapport à la production idéale rationnelle. Lorsque les déviations sont des fonctions convexes positives, Kubiak et Sethi [58, 59] montrent qu'on peut reformuler ce modèle en tant que problème d'affectation. Dans cette formulation, des pénalités de retard et d'avance sont introduites pour représenter les déviations des solutions réelles par rapport à la solution idéale. Ces pénalités représentent le coût induit lorsque l'on place une pièce trop tôt ou trop tard dans la séquence par rapport à son instant idéal de production. Inman et Bulfin [51] proposent une heuristique pseudo-polynomiale pour minimiser la somme des déviations avec des fonctions de pénalité légèrement différentes. Ils considèrent que la date d'échéance souhaitée pour la pièce est l'instant idéal de production et résolvent le problème en triant les pièces selon leur dates d'échéance croissantes (*Earliest Due Date Rule* en anglais). Steiner et Yeomans [83] prouvent qu'il est en fait possible de résoudre ce problème de façon exacte en temps pseudo-polynomial en utilisant la formulation en tant que problème d'affectation.

Une autre classe de fonctions objectif a été beaucoup étudiée dans la littérature. Les contraintes sont identiques, mais le but est de minimiser la *déviation maximale* de la production réelle par rapport à la production idéale. Steiner et Yeomans [82] montrent que, lorsque l'on considère ce problème comme un problème d'ordonnancement avec dates d'échéance et dates de disponibilité, on peut le réduire à la recherche d'un couplage parfait dans un graphe biparti. À partir de cette approche, ils obtiennent un algorithme exact pseudo-polynomial. Brauner et Crama [18] montrent que le problème de décision associé à la minimisation du maximum des déviations est dans la classe Co-$\mathcal{NP}$.

5.1.1 Notations, contraintes et fonctions objectif

Contraintes

Une production diversifiée répartie en lots de petite taille (*diversified small-lot production* en anglais) consiste en un ensemble de n types de pièces ayant chacun une demande $d_i \in \mathbb{N}$, $i = 1, 2, \ldots, n$. Chaque pièce est produite en une unité de temps. Nous noterons $D = \sum_{i=1}^{n} d_i$ la demande totale. On dit qu'un ordonnancement est *uniformément équilibré* (*uniformly leveled*), si à chaque instant k, la ligne a produit exactement $k\, d_i/D$ pièces de type i. La proportion $r_i = d_i/D$ est appelée *taux de production idéal* et un ordonnancement juste-à-temps essaie de garder la production réelle aussi proche que possible de cet idéal. Monden [70] déclare que c'est le but principal des systèmes de production juste-à-temps de Toyota.
Pour formuler ce problème comme un problème d'optimisation, nous introduisons les variables $x_{i,k}$, pour $i = 1, 2, \ldots, n$; $k = 1, 2, \ldots, D$ qui représentent le nombre de pièces de type i produites pendant les instants 1 à k.

Miltenburg [69] a formulé les contraintes du problème comme suit :

$$\begin{array}{lll} \sum_{i=1}^{n} x_{i,k} = k, & k = 1, 2, \ldots, D & (5.1.a) \\ x_{i,D} = d_i, & i = 1, 2, \ldots, n & (5.1.b) \\ 0 \leq x_{i,k} - x_{i,k-1}, & i = 1, 2, \ldots, n;\ k = 2, 3, \ldots, D & (5.1.c) \\ x_{i,k} \in \mathbb{N}, & i = 1, 2, \ldots, n;\ k = 1, 2, \ldots, D & (5.1.d) \end{array} \tag{5.1}$$

L'égalité $(5.1.a)$ indique que k pièces sont produites pendant les k premières unités de temps ; l'égalité $(5.1.b)$ signifie que toutes les demandes doivent être satisfaites à l'instant D ; l'inégalité $(5.1.c)$ déclare que pour un type i donné, le nombre de pièces produites ne peut pas diminuer au cours de la production. Ainsi, les inégalités $(5.1.a)$ et $(5.1.c)$ prises ensemble impliquent qu'une et une seule pièce est produite à chaque instant. La taille d'une solution $X = (x_{i,k})$ du problème ainsi formulé est un $O(nD \max_i \log d_i)$ et n'est donc pas polynomiale de la taille de l'instance $d_1, d_2, \ldots, d_n$. Ainsi, quelle que soit la fonction objectif, cette formulation ne permet pas de conclure quand à l'appartenance du problème d'optimisation à la classe $\mathcal{NPO}$. Il est donc difficile de savoir si le problème de décision associé est dans $\mathcal{NP}$ ou pas. Des problèmes similaires se retrouvent dans une classe plus importante de problèmes high-multiplicity [19].

Fonctions objectif

La fonction objectif doit refléter le fait que nous souhaitons conserver la production réelle *aussi proche que possible* de l'idéal et donc minimiser la distance entre une séquence réalisable et la production parfaitement équilibrée. Il n'y a pas de consensus sur la distance la plus adéquate et de nombreuses fonctions objectif ont été étudiées. Ici, nous en considérons trois : *max-abs*, *sum-abs* et *sum-sqr* qui ont fait l'objet de plusieurs articles sur l'ordonnancement juste-à-temps.

Si l'on souhaite minimiser le maximum des déviations, la fonction objectif est de la forme $F_{max} = \max_{1 \leq i \leq n,\, 1 \leq k \leq D} F_i(x_{i,k} - k r_i)$. Dans [82, 56, 18] par exemple, $F_i(x) = |x|$, pour $i = 1, 2, \ldots, n$ et la fonction objectif est $F_{max} = \max_{i,k} |x_{i,k} - k r_i|$. Nous notons *max-abs* cette

fonction et le problème d'optimisation associé. Dans la Section 5.1.2.2, on reformule le problème max-abs en tant que recherche d'un couplage parfait dans un graphe biparti particulier comme dans [82]. Un résultat concernant les autres choix pour les fonctions F_i est présenté dans la Section 5.2.2.2.

Si on souhaite minimiser la somme des déviations, la fonction objectif est de la forme $F_{sum} = \sum_{k=1}^{D} \sum_{i=1}^{n} F_i\left(x_{i,k} - kr_i\right)$. Ce problème est appelé *min-sum*. Dans [59], les déviations F_i, pour $i = 1, 2, \ldots, n$, sont des fonctions convexes vérifiant

$$F_i\left(0\right) = 0, \quad i = 1, 2, \ldots, n \quad \text{et} \quad F_i\left(y\right) > 0 \quad \text{pour tout } y \neq 0;\ i = 1, 2, \ldots, n \tag{5.2}$$

Pour les problèmes min-sum, on prend en général comme déviations $F_i(x) = |x|$ ou $F_i(x) = x^2$ [69, 59, 56]. Dans le premier cas, $F_{sum} = \sum_{k=1}^{D} \sum_{i=1}^{n} |x_{i,k} - kr_i|$, et nous obtenons le problème *sum-abs*. Dans le second cas, $F_{sum} = \sum_{k=1}^{D} \sum_{i=1}^{n} \left(x_{i,k} - kr_i\right)^2$ et nous nous référons aux problèmes *sum-sqr*. Dans la Section 5.1.2.3, les problèmes juste-à-temps pour lesquels on minimise la somme des déviations sont formulés comme des problèmes d'affectation [59].

On remarque qu'aucune de ces fonctions objectif ne se calcule en temps polynomial par rapport à la taille de l'instance.

instant		1	2	3	4	5	6	7	8	9	10
production		c	d	a	c	d	c	d	b	c	d
1	$x_{a,k}$	0	0	1	1	1	1	1	1	1	1
	déviation	0.1	0.2	0.7	0.6	0.5	0.4	0.3	0.2	0.1	0
	idéal	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
2	$x_{b,k}$	0	0	0	0	0	0	0	1	1	1
	déviation	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.2	0.1	0
	idéal	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
3	$x_{c,k}$	1	1	1	2	2	3	3	3	4	4
	déviation	0.6	0.2	0.2	0.4	0	0.6	0.2	0.2	0.4	0
	idéal	0.4	0.8	1.2	1.6	2	2.4	2.8	3.2	3.6	4
4	$x_{d,k}$	0	1	1	1	2	2	3	3	3	4
	déviation	0.4	0.2	0.2	0.6	0	0.4	0.2	0.2	0.6	0
	idéal	0.4	0.8	1.2	1.6	2	2.4	2.8	3.2	3.6	4

TAB. 5.1 – Un exemple de calcul des déviations $F_i(x) = |x|$ pour l'instance $n = 4$, $d_a = 1$, $d_b = 1$, $d_c = 4$, $d_d = 4$

La Table 5.1 donne un exemple de calcul des déviations pour max-abs ou sum-abs par rapport à la production idéale.

5.1.2 Formulations du problème d'ordonnancement juste-à-temps

Dans cette section, nous faisons le lien entre les problèmes JAT et un problème classique de théorie des votes (Section 5.1.2.1), puis nous décrivons les différentes formulations des problèmes max-abs (Section 5.1.2.2) et sum-abs (Section 5.1.2.3) et présentons un certificat d'optimalité des problèmes sum-abs (Section 5.1.2.4).

5.1.2.1 Théorie des votes

Le modèle d'ordonnancement juste-à-temps est lié au problème de *partage proportionnel en nombres entiers* [12, 11]. Ce problème est défini comme suit :

Partage proportionnel en nombres entiers
Question : Comment répartir h objets identiques entre n groupes de tailles distinctes de sorte que le partage soit à la fois proportionnel à la taille des groupes et que la part de chaque groupe soit en nombre entier ?

Le problème de partage proportionnel en nombres entiers a été largement étudié dans le cadre de la théorie des votes [10]. Par exemple, en France, l'élection des députés à l'assemblée nationale commence par la répartition des 577 sièges entre les différents départements. Ce partage doit être fait de manière proportionnelle au nombre d'habitants de ces départements. Il s'agit donc bien d'un problème de partage proportionnel en nombres entiers. Bien évidement, la solution idéale et parfaitement proportionnelle est très rarement entière et les méthodes employées ne peuvent qu'essayer de s'en approcher. Cela a conduit à un certain nombre de paradoxes historiques, dont le *Paradoxe d'Alabama* qui doit son nom à l'état d'Alabama qui avait vu son nombre de sièges au parlement américain diminuer alors que le nombre de sièges total augmentait. Pour éviter que ce paradoxe se reproduise, on a introduit le principe de *monotonie de la chambre* : si le nombre total de sièges augmente, le nombre de sièges de chacun des groupes ne doit pas diminuer.

Pour un problème d'ordonnancement juste-à-temps, on cherche à connaître à chaque instant h la production totale $x_{i,h}$ de pièces de type i. Si on dispose d'une méthode de partage proportionnel en nombres entiers, on pourra pour un instant h donné répartir les h instants de production écoulés (les sièges) entre les types de pièces (les états) en fonction du quotient de la demande en pièces de type i (la population) par la demande totale (la population totale). On peut donc appliquer cette méthode pour $h = 1$, puis $h = 2$, etc. jusqu'à $h = n$ pour obtenir une solution au problème d'ordonnancement juste-à-temps. Si on veut qu'elle soit réalisable, il faut toutefois que la méthode de théorie des votes choisie respecte le principe de monotonie de la chambre, pour que le nombre de pièces de chaque type produit soit bien croissant en fonction du temps.

Deux grandes catégories d'heuristiques ont été proposées pour le problème de partage proportionnel en nombres entiers : les *méthodes des restes* et les *méthodes de diviseur*.

Notons n le nombre de départements, p_i la population du département i, $P = \sum p_i$ la population totale et h le nombre de sièges à répartir. On appelle *quote-part*, la portion proportionnelle q_i des h sièges pour le département i. On a $q_i = h\frac{p_i}{P}$.

Les *méthodes des restes* attribuent à chaque département la partie entière de sa quote-part $\lfloor q_i \rfloor$ puis répartissent les sièges restants (qui sont au nombre de $h - \sum \lfloor q_i \rfloor$). Par exemple, la méthode des *Plus forts restes*, proposée par Hamilton (1792) ou Vinton (1852), affecte les sièges restants aux départements dont les restes $r_i = q_i - \lfloor q_i \rfloor$ sont les plus grands alors que la méthode de Lowndes (1822) les affecte aux départements pour lesquels les restes ajustés $r_i/\lfloor q_i \rfloor$ sont les plus grands. Ces méthodes ne respectent pas toujours la monotonie de la chambre et sont donc mal adaptées aux problèmes d'ordonnancement juste-à-temps, même si elles ont l'avantage de toujours donner aux départements la partie entière de leur quote-part.

Les *méthodes de diviseur* considèrent une liste d'entiers d_j, $j = 1, 2, \ldots, h$ qui placés entre chaque paire d'entiers consécutifs de $\{0, 1, \ldots, k\}$ vont permettre de décider comment arrondir les quotes-parts de chaque département. En effet, le *d-arrondi* de q_i est égal à m si q_i est compris entre d_{m-1} et d_m. Ce d-arrondi est unique sauf si $q_i = d_m$. Il faut donc que la méthode choisisse entre les valeurs $q_i = d_m$ et $q_i = d_{m+1}$ quel sera dans ce cas le d-arrondi de q_i. Le principe général des méthodes de diviseur est de choisir un diviseur commun x, par exemple $x = P/h$, et de diviser la population de chaque département par x pour obtenir les quotients $q_i = p_i/x$. Ensuite, on obtient le d-arrondi de ces quotients, x_i, et on attribue temporairement x_i sièges au département i. Si $\sum x_i = h$, alors on a fini. Si $\sum x_i > h$ alors, on augmente le diviseur x et si $\sum x_i < h$ on le diminue afin de recommencer l'étape précédente pour essayer d'obtenir un total de h. De nombreuses valeurs ont été essayées pour les d_i et ont donné lieu à des méthodes historiques (méthodes d'Adams (1832), de Dean (1832), du marquis de Condorcet (1792), etc.). Les méthodes de diviseur ont l'avantage de conserver la monotonie de la chambre, mais elles ne garantissent pas forcement que chaque département se voit attribuer un nombre de sièges au moins égal à la partie entière de sa quote-part.

Cela peut sembler un problème pour utiliser ces méthodes pour résoudre des problèmes juste-à-temps, mais si on est assuré que si $D = h$, le nombre de sièges attribués à chaque groupe est bien égal à sa population, on assure de trouver une solution réalisable pour le problème JAT en appliquant D fois cette méthode. Si on se propose d'utiliser comme premier diviseur D/h, ce sera le cas. Cependant, $D = \sum d_i$ n'est que pseudo-polynomial de la taille de l'instance et non polynomial. Or, le problème d'ordonnancement juste-à-temps peut être résolu de façon exacte en temps pseudo-polynomial si on minimise le maximum ou la somme des déviations. Ces méthodes heuristiques ne sont donc pas appliquées pour les problèmes d'ordonnancement juste-à-temps.

5.1.2.2 Déviation maximale et graphes bipartis

Le problème max-abs, aussi noté MDJIT (pour *maximum déviation Just In Time problem* en anglais) dans [18], a été analysé par Steiner et Yeomans [82] et Brauner et Crama [18]. Tous les résultats de cette section sont dérivés de ces deux papiers. Considérons le problème de décision associé au problème (5.1) avec la fonction objectif max-abs :

Problème de décision max-abs
Instance :
- $n \in \mathbb{N}$: nombre de types de pièces ;
- $d_i \in \mathbb{N}$: demande pour le type de pièces i, $i = 1, 2, \ldots, n$;
- $B \in \mathbb{Q}$: une borne.

Question : Existe-t-il une matrice $x = (x_{i,k})$ de taille $n \times D$ telle que :

$$\max_{1 \leq i \leq n,\, 1 \leq k \leq D} |x_{i,k} - kr_i| \leq B$$

$$\begin{array}{ll} \sum_{i=1}^{n} x_{i,k} = k, & k = 1, 2, \ldots, D \\ x_{i,D} = d_i, & i = 1, 2, \ldots, n \\ 0 \leq x_{i,k} - x_{i,k-1}, & i = 1, 2, \ldots, n;\ k = 2, 3, \ldots, D \\ x_{i,k} \in \mathbb{N}, & i = 1, 2, \ldots, n;\ k = 1, 2, \ldots, D \end{array} \tag{5.3}$$

Soit $(n,\, d_1,\, d_2, \ldots,\, d_n,\, B)$ une instance du problème de décision max-abs. On note (i,j) la $j^{ième}$ pièce de type i. Les instants de production au plus tôt et au plus tard de la pièce (i,j) sont définis par

$$E(i,j) = \left\lceil \frac{j-B}{r_i} \right\rceil \quad \text{et} \quad L(i,j) = \left\lfloor \frac{j-1+B}{r_i} + 1 \right\rfloor \tag{5.4}$$

Dans tout ordonnancement réalisable, la pièce (i,j) est produite dans l'intervalle $[E(i,j) \,..\, L(i,j)]$.

Le problème de décision max-abs peut être formulé comme la recherche d'un couplage parfait dans le graphe biparti $G = (V_1 \cup V_2, E)$. L'ensemble de sommets $V_1 = \{1,\, 2, \ldots,\, D\}$ représente les instants de production et V_2 l'ensemble des pièces (i,j). Une arête $(k,(i,j))$ est dans E si et seulement si la pièce (i,j) peut être produite à l'instant k, c'est-à-dire si et seulement si $k \in [E(i,j) \,..\, L(i,j)]$ (voir Figure 5.1).

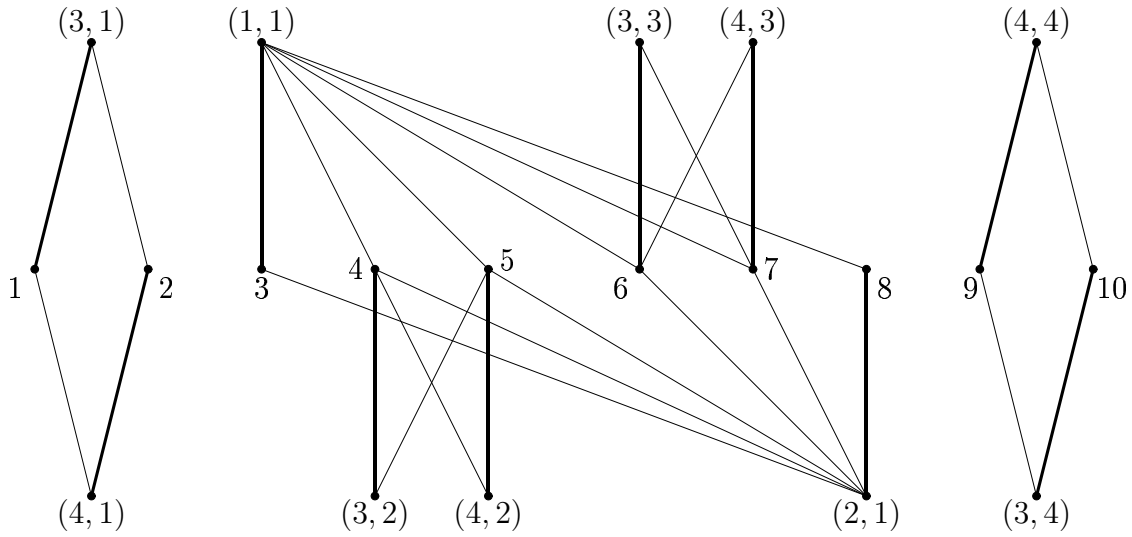


FIG. 5.1 – Graphe biparti pour $d_1 = d_2 = 1$, $d_3 = d_4 = 4$ et $B = \frac{7}{10}$

Le résultat suivant donne une condition nécessaire et suffisante pour qu'une solution soit réalisable.

Proposition 15 [18] *Le problème de décision max-abs a une solution réalisable si et seulement si le graphe G a un couplage parfait.*

De la proposition précédente, nous pouvons déduire :

Proposition 16 *La valeur optimale de la fonction objectif pour le problème max-abs est la plus petite borne B telle que le graphe G correspondant admet un couplage parfait.*

Ce résultat est utilisé pour montrer l'optimalité des solutions proposées pour les problèmes max-abs de la Section 5.2.2.3.

Remarque. Quand on cherche la valeur optimale pour la fonction objectif max-abs, on peut se restreindre aux valeurs $B = q/D$ avec $q \in \{1, 2, \ldots, D-1\}$ puisque les déviations sont des multiples de $\frac{1}{D}$ et que la valeur optimale du maximum des déviations est inférieure ou égale à $1 - 1/D$ [18].

Cette approche fournit un algorithme pseudo-polynomial exact pour le problème de décision max-abs. En effet, le graphe est biparti convexe, ce qui permet de trouver le couplage parfait, s'il existe, en appliquant la règle EDD (Earliest Due Date) avec $L(i,j)$ comme date d'échéance pour la pièce j. Cette règle consiste à placer, à un instant k, la pièce (i,j) non encore placée de plus petite date d'échéance telle que $E(i,j) \geq k$ puis à passer à l'instant $k+1$. Si on ne peut placer aucune pièce à un instant k par cette méthode, alors aucun couplage parfait n'existe. Il n'existe donc pas d'ordonnancement réalisable qui respecte la borne B.

5.1.2.3 Déviation totale et problème d'affectation

Cette section décrit la formulation des problèmes JAT de minimisation de la déviation totale (comme sum-abs et sum-sqr) en tant que problèmes d'affectation [59]. Cette formulation nous permet de trouver les solutions optimales de ces problèmes (toujours en temps pseudo-polynomial).

Dans [59], les auteurs montrent que, pour toutes fonctions convexes F_i, $i = 1, 2, \ldots, n$ vérifiant la condition (5.2), le problème (5.1) avec pour fonction objectif la déviation totale F_{sum} peut être formulé comme un problème d'affectation de la façon suivante.
On introduit la position idéale $Z^*_{i,j} = \lceil k_{i,j} \rceil$ de la pièce (i,j) dans la séquence des pièces usinées comme l'arrondi à l'entier supérieur de l'unique instant $k_{i,j}$ vérifiant

$$F_i\left(j - k_{i,j}\, r_i\right) = F_i\left(j - 1 - k_{i,j}\, r_i\right) \tag{5.5}$$

Si la position de la pièce (i,j) dans la séquence n'est pas $Z_{i,j}$, alors on associe à son usinage un coût non nul. Notons $C_{i,j,k}$ les coûts générés lorsque l'on usine (i,j) à la $k^{ième}$ position :

$$C_{i,j,k} = \begin{cases} \sum_{p=k}^{Z^*_{i,j}-1} \psi_{i,j,p} & \text{si } k < Z^*_{i,j} \\ 0 & \text{si } k = Z^*_{i,j} \\ \sum_{p=Z^*_{i,j}}^{k-1} \psi_{i,j,p} & \text{si } k > Z^*_{i,j} \end{cases} \tag{5.6}$$

où $\psi_{i,j,p}$ est défini par

$$\psi_{i,j,p} = \left|F_i\left(j - p\, r_i\right) - F_i\left(j - 1 - p\, r_i\right)\right| \tag{5.7}$$

Si la pièce (i,j) est usinée avant l'instant idéal de production $Z^*_{i,j}$, alors les $\psi_{i,j,p}$ représentent des *coûts de stockage* (*inventory costs* en anglais) et quand (i,j) est usinée après l'instant idéal $Z^*_{i,j}$, ils représentent des *pénalités de retard* (*shortage costs* en anglais).

Enfin, on définit les variables d'affectation par

$$y_{i,j,k} = \begin{cases} 1 & \text{si } (i,j) \text{ est produite à l'instant } k \\ 0 & \text{sinon} \end{cases}$$

Nous pouvons maintenant donner la formulation du problème d'affectation en tant que programme linéaire :

$$\begin{array}{lll} & \min \sum_{k=1}^{D} \sum_{i=1}^{n} \sum_{j=1}^{d_i} C_{i,j,k}\, y_{i,j,k} & \\ \text{s.c.} & \sum_{i=1}^{n} \sum_{j=1}^{d_i} y_{i,j,k} = 1, & k = 1, 2, \ldots, D \\ & \sum_{k=1}^{D} y_{i,j,k} = 1, & i = 1, 2, \ldots, n;\ \forall j = 1, 2, \ldots, d_i \\ & y_{i,j,k} \in \{0, 1\} & i = 1, 2, \ldots, n;\ \forall j = 1, 2, \ldots, d_i;\ \forall k = 1, 2, \ldots, D \end{array} \tag{5.8}$$

Les contraintes indiquent qu'une seule pièce est produite à l'instant k et que la pièce (i, j) est usinée une fois et une seule.

On peut adopter cette formulation du problème pour résoudre le problème de décision max-abs. En effet, pour une borne B, il suffit de choisir la valeur suivante pour les coûts :

$$C_{i,j,k} = \begin{cases} 0 & \text{si } k \in [E(i,j) \,..\, L(i,j)] \\ 1 & \text{sinon} \end{cases}$$

S'il existe une solution de coût nul, alors la réponse au problème de décision sera oui, et elle sera non sinon.

Cette formulation du problème ne permet pas plus que la précédente de placer dans $\mathcal{NPO}$ les problèmes max-abs ou sum-abs, puisque les solutions ne sont pas non plus de taille polynomiale par rapport à la taille de l'instance et que la fonction objectif ne peut toujours pas être calculée en temps polynomial.

Elle permet par contre d'obtenir en temps pseudo-polynomial la solution optimale pour les problèmes min-sum, puisqu'il est possible de calculer l'affectation de poids minimum en temps polynomial par rapport à la taille du graphe.

5.1.2.4 Certificat d'optimalité pour les problèmes min-sum

Pour montrer l'optimalité de solutions des problèmes min-sum, nous considérons la résolution en termes de graphes des problèmes d'affectation associés. L'objectif est de trouver le plus petit couplage parfait dans le graphe pondéré biparti $G_w = (V_1 \cup V_2, E)$ où, comme dans la Section 5.1.2.2, l'ensemble de sommets $V_1 = \{1, 2, \ldots, D\}$ représente les instants de production et V_2 l'ensemble des pièces à produire.

On peut remarquer que si on minimise la déviation totale, une pièce (i, j) peut être produite à n'importe quel instant k. Ainsi, G_w est un graphe biparti complet. Une arête $e = (k, (i, j)) \in E$ a pour poids $w_e = C_{i,j,k}$. Soit A la matrice d'incidence sommets-arêtes du graphe G_w (c'est-à-dire la matrice $2\,D \times D^2$ telle que $a_{l,m} = 1$ si le $l^{ième}$ sommet est incident à la $m^{ième}$ arête et 0 sinon).

Nous pouvons reformuler le problème par

$$\begin{array}{ll} & \min\ wy \\ s.c. & Ay = 1 \\ & y \text{ entier} \end{array} \tag{5.9}$$

La matrice d'incidence d'un graphe biparti est totalement unimodulaire. Ainsi, le polytope associé au problème (5.9) est entier. On peut donc résoudre la relaxation linéaire du programme en nombres entiers (5.9) pour obtenir la valeur optimale de la fonction objectif.

La relaxation linéaire duale de (5.9) est

$$\begin{array}{ll} & \max\ 1z \\ s.c. & zA \leq w \\ & z \text{ non contraint} \end{array} \tag{5.10}$$

Ce problème dual (5.10) peut être interprété comme la recherche d'une fonction de poids z définie sur l'ensemble des sommets telle que la somme des z_l pour tout $l \in V_1 \cup V_2$ est maximale et telle que pour toute arête $(k,(i,j))$, on a $z_k + z_{i,j} \leq C_{i,j,k}$. Ainsi, on obtient le résultat suivant :

Proposition 17 *Si on minimise la déviation totale, alors s est la valeur optimale de la fonction objectif si et seulement si il existe un couplage parfait M et une fonction de coût $z : V_1 \cup V_2 \to \mathbb{R}$ tels que*

$$\begin{cases} z_k + z_{i,j} \leq C_{i,j,k}, & \forall\,(k,(i,j)) \in E \\ s = \sum_{k \in V_1} z_k + \sum_{(i,j) \in V_2} z_{i,j} = \sum_{(k,(i,j)) \in M} C_{i,j,k} \end{cases} \tag{5.11}$$

Ce résultat nous permet de prouver de façon compacte qu'une solution est optimale pour un problème min-sum en exhibant le couplage parfait et la fonction de poids correspondants dans le graphe G_w.

5.1.2.5 Problèmes min-sum B-bornés

Dans le problème (5.8), il n'y a aucune contrainte concernant le maximum des déviations. Cependant, nous considérerons par la suite des problèmes min-sum dont la déviation maximale est bornée. Dans ce cas, l'ensemble des arêtes est celui décrit précédemment : une arête $(k,(i,j))$ est dans E si et seulement si la pièce (i,j) peut être produite à l'instant k, c'est-à-dire si et seulement si $k \in [E(i,j)\,..\,L(i,j)]$.

On dira qu'un problème min-sum dont le maximum des déviations est borné par B est un problème min-sum *B-borné*.

5.2 Optimisations simultanées de fonctions objectif

Nous souhaitons dans cette section étudier l'optimisation simultanée de plusieurs critères. Dans la mesure où une solution de maximum des déviations inférieure à 1 existe toujours, nous nous intéresserons en particulier aux problèmes 1-bornés.

5.2.1 Problèmes sum-abs et sum-sqr 1-bornés

L'ensemble des toutes les solutions d'un problème 1-borné est beaucoup plus petit que l'ensemble des solutions pour sa version non bornée. S'intéresser aux problèmes 1-bornés

plus tôt qu'aux problèmes non bornés permet donc des améliorations du temps de calcul pour des méthodes de résolutions exactes de min-sum. Par exemple, calculer le plus petit couplage parfait peut être fait en $O(nD^2 \log D)$ pour un problème min-sum 1-borné au lieu de $O(D^3 \log D)$ pour sa version non bornée avec $D >> n$ (voir [83]). Pour toutes les instances, il est possible de trouver une séquence dont le maximum des déviations est inférieur à 1 [82]. Ainsi, il est intéressant de savoir s'il existe toujours une séquence 1-bornée qui minimise la déviation totale, c'est-à-dire si tout problème min-sum 1-borné a la même valeur optimale que sa version non bornée.
Dans [56], les auteurs testent les deux questions suivantes sur $100,000$ instances choisies au hasard :
- Y-a-t-il une solution optimale pour sum-abs telle que le maximum des déviations est inférieur à 1 ?
- Y-a-t-il une séquence optimale pour sum-abs qui optimise sum-sqr ?

Pour toutes les instances testées, les réponses étaient "oui" dans les deux cas. De plus, dans [56], les auteurs ont remarqué que la séquence 1-bornée optimale pour sum-abs était toujours optimale pour sum-sqr. Des tests exhaustifs montrent qu'une telle séquence optimale 1-bornée n'existe pas pour toutes les instances (voir Section 5.2.2.3).

Le lemme suivant compare les coûts de toutes les arêtes $(k,(i,j))$ du graphe correspondant aux problèmes sum-abs et sum-sqr 1-bornés

Lemme 14 *Considérons la borne $B = 1$ des déviations. Pour tout $i = 1, 2, \ldots, n$; $j = 1, 2, \ldots, d_i$ et $k = 1, 2, \ldots, D$, on a*

$$k \in [E(i,j) \,..\, L(i,j)] \Rightarrow C^a_{i,j,k} = C^s_{i,j,k},$$

où C^a est la matrice des coûts calculée pour le problème sum-abs
et C^s est la matrice des coûts calculée pour le problème sum-sqr.

Démonstration. Notons $\lceil k^a_{i,j} \rceil$ et $\lceil k^s_{i,j} \rceil$ les instants de production idéaux de la pièce (i,j) pour les problèmes sum-abs et sum-sqr respectivement. Comme $|x| = |x-1|$ si et seulement si $x = \frac{1}{2}$ et $x^2 = (x-1)^2$ si et seulement si $x = \frac{1}{2}$, on a $k^a_{i,j} = k^s_{i,j}$. Ainsi, on peut noter $Z^*_{i,j} = \lceil k^a_{i,j} \rceil = \lceil k^s_{i,j} \rceil$ l'instant de production idéal de (i,j) pour les deux problèmes.

Soit $C^a_{i,j,k}$ et $C^s_{i,j,k}$ les coûts induits en plaçant (i,j) dans la $k^{ième}$ position et $\psi^a_{i,j,p}$ et $\psi^s_{i,j,p}$ les coûts de stockage et de retard pour chacun des deux problèmes. La fonction $f(x) = |x^2 - (x-1)^2| - ||x| - |x-1||$ est nulle si et seulement si $x \in [0,1]$. Ainsi, nous avons le résultat suivant :

$$\psi^a_{i,j,p} = \psi^s_{i,j,p} \qquad \text{si et seulement si} \qquad j - p\, r_i \in [0,1]$$

On considère une pièce (i,j) et un instant de production k et on suppose que $k \in [E(i,j) \,..\, L(i,j)]$, avec $E(i,j)$ et $L(i,j)$ les instants de production au plus tôt et au plus tard correspondants à la borne $B = 1$.
Comme $E(i,j) \leq k$, on a $\left\lceil \frac{j-1}{r_i} \right\rceil \leq k$. Ainsi, $r_i \left\lceil \frac{j-1}{r_i} \right\rceil \leq k r_i$.

On peut alors en déduire que $j - 1 \leq k r_i$ et donc que

$$j - k r_i \leq 1$$

Trois cas sont possibles en fonction des positions relatives de k et $Z^*_{i,j}$.

- Cas 1 : $k < Z^*_{i,j}$
  Pour tout $p = k,\, k+1, \ldots,\, Z^*_{i,j} - 1$, on a

$$j - p\, r_i > \frac{1}{2}$$

  puisque la fonction $g(x) = j - xr_i$ est décroissante et que $g(k_{i,j}) = \frac{1}{2}$. Ainsi, on a $0 \leq j - p\, r_i \leq 1$ et donc

$$\forall p = k,\, k+1, \ldots,\, Z^*_{i,j} - 1, \qquad \psi^a_{i,j,p} = \psi^s_{i,j,p}$$

- Cas 2 : $k > Z^*_{i,j}$
  On a $k \leq L(i,j)$, donc $k \leq \left\lfloor \frac{j}{r_i} \right\rfloor + 1$. Ainsi, $(k-1)r_i \leq \left\lfloor \frac{j}{r_i} \right\rfloor r_i$.
  On a alors $(k-1)r_i \leq j$. Nous obtenons donc

$$0 \leq j - (k-1)r_i$$

  Comme la fonction g est décroissante, pour tout $p = Z^*_{i,j},\, Z^*_{i,j} + 1, \ldots,\, k-1$, on a $j - pr_i \in [0,1]$. Ainsi,

$$\forall p = Z^*_{i,j},\, Z^*_{i,j} + 1, \ldots,\, k-1, \qquad \psi^a_{i,j,p} = \psi^s_{i,j,p}$$

- Cas 3 : $k = Z^*_{i,j}$
  Dans ce cas, les deux coûts sont nuls.

Ainsi, d'après la définition des coûts, on a dans les trois cas :

$$C^a_{i,j,k} = C^s_{i,j,k}$$

ce qui démontre le lemme. □

Le Théorème 13 suivant a été montré indépendamment dans [35]. Nous donnons ici une courte preuve alternative basée sur le Lemme 14.

Théorème 13 [35] *Soit y une séquence optimale pour sum-abs. Si son maximum des déviations est inférieur à 1, alors elle est aussi optimale pour le problème sum-sqr.*

Notre Démonstration du Théorème 13
La fonction $f(x) = |x^2 - (x-1)^2| - ||x| - |x-1||$ est positive. On a donc

$$\forall i, j, k, \qquad C^a_{i,j,k} \leq C^s_{i,j,k}$$

Ainsi, pour toute solution réalisable y du problème d'ordonnancement JAT,

$$\sum_{i=1}^{n} \sum_{j=1}^{d_i} \sum_{k=1}^{D} C^a_{i,j,k}\, y_{i,j,k} \leq \sum_{i=1}^{n} \sum_{j=1}^{d_i} \sum_{k=1}^{D} C^s_{i,j,k}\, y_{i,j,k} \tag{5.12}$$

Soit y une solution telle que le maximum de ses déviations est inférieur à 1. Supposons que $y_{i,j,k} = 1$. Comme le maximum des déviations pour y est inférieur à 1, on a $k \in [E(i,j) \,..\, L(i,j)]$ avec $E(i,j)$ et $L(i,j)$ correspondant à la borne $B = 1$. D'après le Lemme 14, on a $C^s_{i,j,k} = C^a_{i,j,k}$. Ainsi, pour tout y tel que le maximum des déviations est inférieur à 1,

$$y_{i,j,k} = 1 \Rightarrow C^s_{i,j,k} = C^a_{i,j,k}$$

Supposons que y^* est une solution optimale du problème sum-abs telle que le maximum des déviations est inférieur à 1. Alors, pour tout ordonnancement réalisable y, on a

$$\sum_{i=1}^{n}\sum_{j=1}^{d_i}\sum_{k=1}^{D} C^s_{i,j,k}\, y^*_{i,j,k} = \sum_{i=1}^{n}\sum_{j=1}^{d_i}\sum_{k=1}^{D} C^a_{i,j,k}\, y^*_{i,j,k} \leq \sum_{i=1}^{n}\sum_{j=1}^{d_i}\sum_{k=1}^{D} C^a_{i,j,k}\, y_{i,j,k} \qquad (5.13)$$

En combinant les équations (5.12) et (5.13), on obtient que pour toute solution réalisable y,

$$\sum_{i=1}^{n}\sum_{j=1}^{d_i}\sum_{k=1}^{D} C^s_{i,j,k}\, y^*_{i,j,k} \leq \sum_{i=1}^{n}\sum_{j=1}^{d_i}\sum_{k=1}^{D} C^s_{i,j,k}\, y_{i,j,k}$$

et donc y^* est une solution optimale du problème sum-sqr. □

Corollaire 7 *Si pour une instance, il existe une paire de solutions optimales (y^*, Y^*) des problèmes sum-abs et sum-sqr respectivement vérifiant*

$$\sum_{i=1}^{n}\sum_{j=1}^{d_i}\sum_{k=1}^{D} C^s_{i,j,k}\, Y^*_{i,j,k} > \sum_{i=1}^{n}\sum_{j=1}^{d_i}\sum_{k=1}^{D} C^a_{i,j,k}\, y^*_{i,j,k}$$

alors le problème sum-abs n'a aucune solution optimale telle que le maximum des déviations est inférieur ou égal à 1.

Démonstration. On obtient ce résultat directement à partir de la preuve du Théorème 13. □

On trouve un exemple d'utilisation de ce lemme dans l'Annexe A.4.1.

Le Théorème 13 soulève naturellement certaines questions :
- Une séquence optimisant sum-abs dont le maximum des déviations est inférieur à 1 est optimale pour sum-sqr, mais est-ce qu'une séquence optimale pour sum-sqr avec maximum des déviations inférieur à 1 est toujours optimale pour sum-abs ? C'est-à-dire, peut-on échanger sum-abs et sum-sqr dans le Théorème 13 ?
- Pour appliquer le Théorème 13, nous avons besoin de l'existence d'une séquence optimale pour sum-abs ayant un maximum des déviations inférieur à 1. Peut-on toujours trouver une telle séquence ? C'est-à-dire, existe-t-il pour toute instance une séquence qui optimise sum-abs et dont le maximum des déviations est inférieur à 1 ?
- Si une telle séquence n'existe pas, peut-on tout de même optimiser sum-abs et sum-sqr simultanément pour toute instance ?
- Peut-on trouver pour toute instance une séquence optimale 1-bornée pour sum-sqr ?

La Section 5.2.2 répond à ces questions en proposant des instances pour lesquelles il existe ou pas des séquences optimisant les différents critères simultanément.

5.2.2 Comparaison des fonctions objectif

5.2.2.1 Avec deux types de pièces

Pour le problème à deux types de pièces, les fonctions objectif max-abs, sum-abs et sum-sqr peuvent être optimisées simultanément, c'est-à-dire qu'il existe une solution qui est optimale pour les trois fonctions objectif. Afin d'être complet, nous prouvons ce résultat qui a déjà été mentionné dans [18]. On peut le déduire de la proposition suivante :

Proposition 18 *Pour le problème à deux types de pièces de taux de production r_1 et $r_2 = 1 - r_1$, la solution x^* du problème d'ordonnancement juste-à-temps définie par*

$$\begin{aligned} x^*_{1,k} &= [k\, r_1] \quad k = 1,\, 2, \ldots,\, D \\ x^*_{2,k} &= k - [k\, r_1] \quad k = 1,\, 2, \ldots,\, D \end{aligned}$$

est optimale pour toutes les fonctions F_{max} ou F_{sum} telles que F_1 et F_2 sont des fonctions positives et croissantes sur $[0, +\infty[$ vérifiant

$$\forall i = 1,\, 2, \quad \forall y \in \mathbb{R}, \qquad F_i\,(y) = F_i\,(-y)$$

et pour tout $y \in \mathbb{R}$, $[y]$ est l'arrondi au plus proche entier de y : $[y]$ est l'entier tel que $y - \frac{1}{2} \leq [y] < y + \frac{1}{2}$.

Démonstration. Soit x une solution réalisable du problème JAT.

$$\forall k = 1,\, 2, \ldots,\, D,\ x_{1,k} = j \Rightarrow x_{2,k} = k - j$$

Donc, si $x_{1,k} = j$ on a

$$\begin{aligned} x_{2,k} - kr_2 &= k - j - kr_2 \\ &= k\,(1 - r_2) - j \\ &= kr_1 - j \\ &= -\,(x_{1,k} - kr_1) \end{aligned}$$

Ainsi, $\forall k = 1,\, 2, \ldots,\, D,\ |x_{1,k} - kr_1| = |x_{2,k} - kr_2|$.
Par définition de $x^*_{i,k}$, on a $|x^*_{i,k} - kr_i| \leq \frac{1}{2}$ pour tout $i = 1, 2$ et tout $k = 1,\, 2, \ldots,\, D$.

Considérons maintenant une solution réalisable x du problème JAT qui diffère de x^* à l'instant l. Les auteurs montrent dans [18] qu'une telle solution vérifie $|x_{1,l} - lr_1| \geq \frac{1}{2}$.
Ainsi, $\forall k = 1,\, 2, \ldots,\, D, \forall i = 1, 2$, on a $|x_{i,k} - kr_i| \geq |x^*_{i,k} - kr_i|$.
Les fonctions F_1 et F_2 vérifient $\forall i = 1, 2, \forall y \in\,]-\infty, +\infty[,\ F_i\,(y) = F_i\,(|y|)$ et sont croissantes sur $[0, +\infty[$.
On a donc, $\forall k = 1,\, 2, \ldots,\, D,\ \forall i = 1, 2,\ F_i\,(x_{i,k} - kr_i) \geq F_i\left(x^*_{i,k} - kr_i\right)$.
Alors, pour toute fonction objectif, $F = F_{max}$ ou $F = F_{sum}$ telle que F_1 et F_2 sont des fonctions positives croissantes sur $[0, +\infty[$, x^* est une solution optimale pour le problème JAT associé à la fonction F. □

Remarque. Ce résultat reste valable si les fonctions F_1 et F_2 sont identiques, positives, décroissantes sur $]-\infty, 0]$ et croissantes sur $[0, +\infty[$. Il n'est plus valable si les fonctions sont différentes, positives, décroissantes sur $]-\infty, 0]$, croissantes sur $[0, +\infty[$ et non paires. Pour l'instance $d_a = 1$ et $d_b = 3$, si on minimise le total des déviations avec $F_a(-\frac{1}{2}) + F_b(\frac{1}{2}) < F_a(\frac{1}{2}) + F_b(-\frac{1}{2})$, alors x^* qui représente la séquence $(b,\, b,\, a,\, b)$ n'est pas optimale puisque la séquence $S = (b,\, a,\, b,\, b)$ est strictement meilleure.

Avec plus de deux types de pièces, il n'est plus possible pour toute instance d'optimiser toutes les fonctions objectifs simultanément. L'Annexe A.3 propose un exemple démontrant ce résultat.

5.2.2.2 Déviations pour la fonction objectif $Fmax$

On peut s'interroger sur le fait que le maximum des déviations est étudié dans la littérature uniquement pour les déviations $F_i(x) = |x|$.

Le résultat suivant montre que l'on peut en fait prendre n'importe quelles fonctions comme déviations, pourvue que toutes les fonctions F_i soient identiques et égales à une fonction F qui pénalise de façon égale le retard et l'avance.

Proposition 19 *Soit B une borne et $F : \mathbb{R} \longrightarrow \mathbb{R}^+$ une fonction paire strictement croissante sur $\mathbb{R}^+$. Supposons que toutes les déviations F_i sont égales à F. Alors, une solution $X = (x_{i,k})$ (pour le problème (5.1)) a un maximum des déviations inférieur à B si et seulement si*

$$\max_{i,k} F(x_{i,k}) \leq F(B)$$

En particulier, X est optimal pour $\max_{i,k} |x_{i,k} - kr_i|$ si et seulement si X est optimale pour $\max_{i,k} F(x_{i,k} - kr_i)$.

Idée de la preuve [52] :
Soit $F : \mathbb{R}^+ \longrightarrow \mathbb{R}^+$ une fonction croissante et soit A un sous-ensemble fini de $\mathbb{R}^+$. Alors

$$\max_{a \in A} \{F(a)\} = F\left(\max_{a \in A}\{a\}\right)$$

De plus, si F est strictement croissante, le membre de gauche est atteint uniquement par l'élément maximum $a \in A$. □

Ainsi, toutes les fonctions de déviations paires et convexes sont équivalentes si on optimise le maximum des déviations : elles ont le même ensemble de solutions optimales

5.2.2.3 Optimiser le maximum et la somme des déviations simultanément

Pour trouver les exemples de cette section, nous avons utilisé le logiciel CPLEX afin de résoudre les programmes linéaires décrivant les différents problèmes et nous avons adopté pour ceux-ci des formulations efficaces. Une description complète de ces formulations est donnée dans l'Annexe A.2.

Nos tests avaient pour but d'obtenir des instances pour lesquelles certaines séquences optimisent plusieurs critères simultanément, ou pour lesquelles aucune séquence n'optimise certains critères simultanément. Pour une instance donnée, nous avons introduit cinq variables booléennes notées AM, SM, $AM1$, $SM1$ et AS pour décrire quels critères sont optimisés par cette instance. Dans ces notations, A correspond à sum-abs, S à sum-sqr, M à max-abs, $M1$ à maximum des déviations inférieur à 1. Pour une instance donnée,

AM est vraie si et seulement si il existe une séquence qui optimise sum-abs et max-abs simultanément, SM est vraie si et seulement si une séquence optimise sum-sqr et max-abs simultanément, $AM1$ est vraie si et seulement si le maximum des déviations d'une séquence optimale de sum-abs est inférieur à 1, $SM1$ est vraie si et seulement si une séquence optimale de sum-sqr a son maximum des déviations inférieur à 1, et AS est vraie si et seulement si une séquence optimise sum-abs et sum-sqr simultanément.

On note V la valeur booléenne $Vrai$ et F la valeur booléenne $Faux$. Le quintuplet $(AM, SM, AM1, SM1, AS)$ ne peut pas prendre toutes les valeurs de $\{Vrai, Faux\}^5$. Remarquons tout d'abord que si AM est vraie, alors $AM1$ est vraie puisque la valeur optimale de max-abs est toujours inférieure à 1. De même, SM est vraie implique que $SM1$ est vraie. Les implications $AM \Rightarrow SM \wedge AS$ et $AM1 \Rightarrow SM1 \wedge AS$ sont des conséquences directes du Théorème 13. De plus, on a l'assertion suivante :

Proposition 20 $SM \wedge AM1 \Rightarrow AM$.

Démonstration. Considérons une instance du problème d'ordonnancement JAT et supposons que SM et $AM1$ sont vraies pour cette instance. Soient Y^* une séquence optimisant sum-sqr et max-abs simultanément et y^* une séquence optimale pour sum-abs dont le maximum des déviations est inférieur à 1.

La solution Y^* est optimale pour sum-sqr. Ainsi,

$$\sum_{i,j,k} C^s_{i,j,k} y^*_{i,j,k} \geq \sum_{i,j,k} C^s_{i,j,k} Y^*_{i,j,k}$$

Le maximum des déviations pour Y^* est égal à la valeur optimale de max-abs et est donc inférieur à 1. Ainsi, on peut déduire de la preuve du Théorème 13 que

$$\sum_{i,j,k} C^s_{i,j,k} Y^*_{i,j,k} = \sum_{i,j,k} C^a_{i,j,k} Y^*_{i,j,k}$$

De même, puisque la solution y^* a un maximum des déviations inférieur à 1, on a

$$\sum_{i,j,k} C^s_{i,j,k} y^*_{i,j,k} = \sum_{i,j,k} C^a_{i,j,k} y^*_{i,j,k}$$

On obtient alors l'inégalité suivante :

$$\sum_{i,j,k} C^a_{i,j,k} y^*_{i,j,k} \geq \sum_{i,j,k} C^a_{i,j,k} Y^*_{i,j,k}$$

Puisque la séquence y^* est minimale pour sum-abs, la séquence Y^* est aussi minimale pour sum-abs. Comme Y^* est optimale pour max-abs et sum-abs, AM est vraie. □

La Figure 5.2 donne une illustration des valeurs possibles du quintuplet (AM, SM, $AM1$, $SM1$, AS). Pour tout $X \in \{AM, SM, AM1, SM1, AS\}$, on note IX l'ensemble des instances telles que X est vraie. Comme $AM \Longleftrightarrow SM \wedge AM1$ (d'après le Théorème 13 et la Proposition 20), l'ensemble IAM est égal à l'intersection des ensembles $IAM1$ et ISM. Pour rendre la Figure 5.2 plus claire, on ne dessine donc pas l'ensemble IAM. Les nombres dans les ensembles représentent les sept exemples de la Table 5.2.

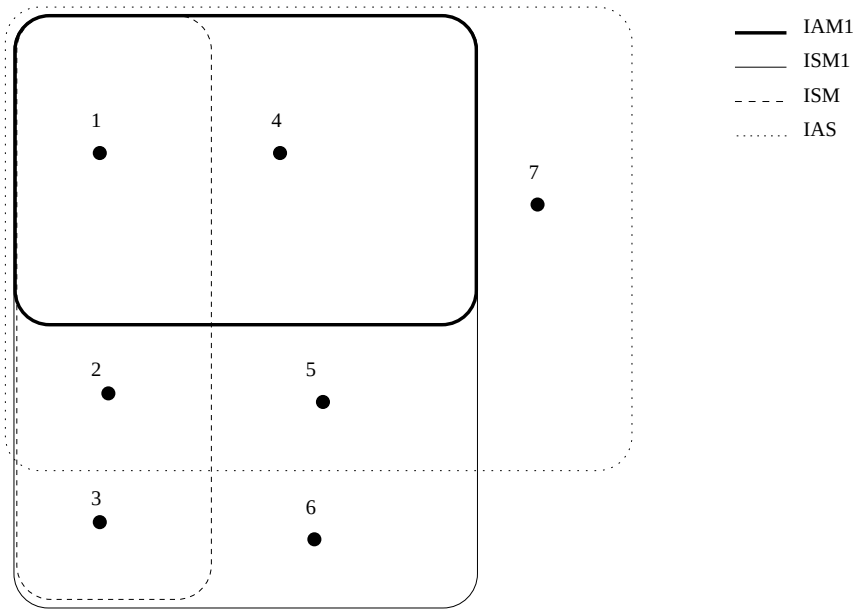


FIG. 5.2 – Représentation des ensembles d'instances correspondant aux différentes valeurs du quintuplet $(AM, SM, AM1, SM1, AS)$

La Table 5.2 propose une instance pour chaque valeur possible du quintuplet (AM, SM, $AM1$, $SM1$, AS) à l'exception du quintuplet (F, F, F, F, F). Ces instances sont données en utilisant une notation compacte pour le vecteur des demandes. On note $d = (d_a^p, d_b^q)$ le vecteur des demandes de l'instance à $n = p + q$ types de pièces dont les demandes des p premiers types sont égales à d_a et les q suivantes sont égales à d_b. Pour chacune des instances, si une des variables booléennes AM à AS est vraie, nous donnons une séquence qui le prouve. Cette séquence est aussi représentée dans une forme compacte. Si les types de pièces i et i' sont tels que $d_i = d_{i'}$, on peut échanger les pièces (i, j) et (i', j) sans modifier la somme ou le maximum des déviations. Puisque dans chacun des exemples, les types de pièces ont au plus deux valeurs différentes d_a et d_b, on note a_j, $j = 1, 2, \ldots,$ d_a l'ensemble des pièces (i, j) telles que $d_i = d_a$ et b_j, $j = 1, 2, \ldots,$ d_b toutes les pièces (i, j) telles que $d_i = d_b$. Par exemple, si $d = (1, 1, 4, 4) = (1^2, 4^2)$, la séquence de pièces $((3, 1), (4, 1), (3, 2), (4, 2), (1, 1), (2, 1), (3, 3), (4, 3), (3, 4), (4, 4))$ est notée $(b_1, b_1, b_2, b_2, a_1, a_1, b_3, b_3, b_4, b_4)$ ou $(b_1^2, b_2^2, a_1^2, b_3^2, b_4^2)$. Dans la Table 5.2, on note s-a la fonction objectif sum-abs, s-s la fonction objectif sum-sqr et m-a la fonction objectif max-abs.

Pour montrer que les variables booléennes AM à AS sont vraies pour une instance donnée, on peut exhiber une séquence avec les caractéristiques souhaitées. L'optimalité de cette séquence pour la fonction objectif sum-abs ou sum-sqr peut être montrée en utilisant la Proposition 17 : on construit le graphe biparti complet G_w (voir Section 5.1.2.4). La séquence est représentée par un couplage parfait. Donner les poids z_k et $z_{i,j}$ des sommets de G_w associés au couplage est suffisant pour montrer l'optimalité de la séquence.
Ainsi, lorsque AS n'est pas déjà impliquée par AM ou $AM1$, on peut montrer que AS est vraie en exhibant le couplage parfait représentant la solution optimale commune à sum-abs et sum-sqr et en proposant les poids des sommets associés à chacun des graphes bipartis (voir Section 5.1.2.4).

TAB. 5.2 – Exemple d'instances dont certaines séquences optimisent plusieurs critères

	Existence d'une séquence optimisant					Instance	Preuve
	AM	SM	AM1	SM1	AS		
	s-a	s-s	s-a	s-s	s-a		
	et	et	avec	avec	et		
	m-a	m-a	m-a≤ 1	m-a≤ 1	s-s		
1	T	T	T	T	T	$d = (1)$	$S = (a_1)$ est optimale pour tous les problèmes
2	F	T	F	T	T	$d = (1^9, 7^3)$	$S = (b_1^3, b_2^3, a_1^3, b_3^3, a_1, b_4^3, a_1^2, b_5^3, a_1^3, b_6^3, b_7^3)$ est optimale pour sum-abs et sum-sqr. $S' = (b_1^3, a_1, b_2^3, a_1^2, b_3^3, a_1, b_4^3, a_1^2, b_5^3, a_1^2, b_6^3, a_1, b_7^3)$ est optimale pour sum-sqr et max-abs
3	F	T	F	T	F	$d = (1^{12}, 10^2)$	$S = (b_1^2, a_1, b_2^2, a_1, b_3^2, a_1^2, b_4^2, a_1, b_5^2, a_1^2, b_6^2, a_1, b_7^2, a_1^2, b_8^2, a_1, b_9^2, a_1, b_{10}^2)$ est optimale pour sum-sqr et max-abs
4	F	F	T	T	T	$d = (1^2, 4^2)$	$S = (b_1^2, b_2^2, a_1^2, b_3^2, b_4^2)$ est optimale pour sum-abs avec un maximum des déviations inférieur à 1
5	F	F	F	T	T	$d = (1^{10}, 6^5)$	$S = (b_1^5, a_1, b_2^5, a_1^3, b_3^5, a_1^2, b_4^5, a_1^3, b_5^5, a_1, b_6^5)$ est optimale pour sum-sqr avec un maximum des déviations inférieur à 1 et $S' = (b_1^5, b_2^5, a_1^3, b_3^5, a_1^4, b_4^5, a_1^3, b_5^5, b_6^5)$ est optimale pour sum-abs et sum-sqr
6	F	F	F	T	F	$d = (1^9, 6^4)$	$S = (b_1^4, a_1, b_2^4, a_1^2, b_3^4, a_1^3, b_4^4, a_1^2, b_5^4, a_1, b_6^4)$ est optimale pour sum-sqr et est 1-bornée.
7	F	F	F	F	T	$d = (1^7, 6^4)$	$S = (b_1^4, b_2^4, a_1^2, b_3^4, a_1^3, b_4^4, a_1^2, b_5^4, b_6^4)$ est optimale pour sum-abs et sum-sqr

Pour montrer qu'une séquence optimale min-sum est B-bornée, on peut soit calculer le maximum des déviations pour la séquence et vérifier qu'il est bien inférieur à B, soit construire le graphe correspondant au problème B borné (voir Section 5.1.2.2) et vérifier que toutes les arêtes de la séquence (c'est-à-dire du couplage parfait optimal pour la fonction objectif min-sum) appartiennent au sous-graphe.

Trouver la valeur optimale de la fonction objectif max-abs peut être fait comme dans la Section 5.1.2.2.

Pour montrer que les variables AM à $SM1$ sont fausses, on doit comparer les valeurs optimales des fonctions objectif min-sum des problèmes bornés et non bornés. Cela signifie que l'on doit calculer le couplage parfait optimal et les poids des sommets pour les problèmes bornés et non bornés. Si les valeurs des deux couplages sont différentes, alors la variable correspondante est fausse. L'exemple suivant indique que AM est fausse pour l'instance $d = (1^2, 4^2)$. Pour plus de détails sur la simplification opérée sur les graphes de cet exemple, le lecteur peut se référer à l'Annexe A.1.1.

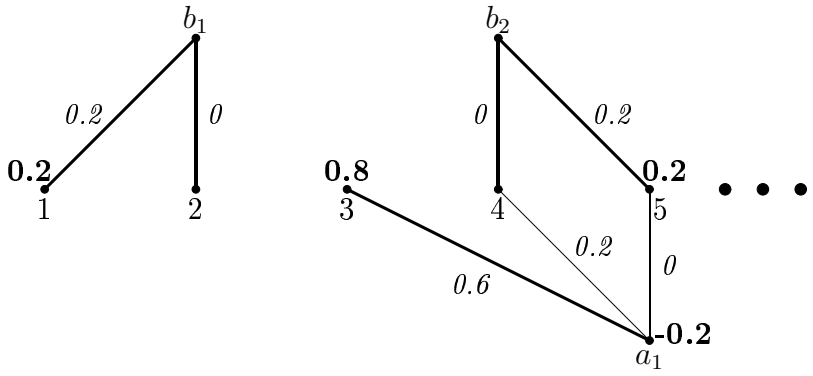


FIG. 5.3 – Graphe biparti pondéré pour $d_1 = d_2 = 1$, $d_3 = d_4 = 4$ et $B = \frac{7}{10}$

Le graphe de la Figure 5.3 représente les arêtes et leur poids (pour le problème sum-abs) du graphe G_w associé à l'instance $d = (1^2, 4^2)$ et à la borne $B = 0.7$, qui est la valeur optimale de la fonction objectif max-abs. L'Annexe A.1.2 d'écrit les méthodes de construction des graphes pour le juste-à-temps et les simplifications qui y sont apportées. Les arêtes en gras représentent un couplage parfait minimum noté M^*. Les poids duaux non nuls sont représentés en gras sur le graphe et en ajoutant zéro sur chacun des autres sommets, on obtient une solution duale optimale qui prouve que le couplage est optimal.

Le graphe de la Figure 5.4 représente un couplage du graphe biparti complet associé à G_w. Ce couplage est strictement meilleur que M^*. Ainsi, on peut en conclure que M^* n'est pas optimal pour le problème sum-abs non borné et que AM est fausse.

Remarque. $D = 10$ est le plus petit D tel qu'il existe une instance pour laquelle les fonctions objectif max-abs et sum-abs ne peuvent pas être optimisées simultanément.

La variable booléenne AS est fausse quand les problèmes sum-abs et sum-sqr n'ont aucune solution optimale commune, c'est-à-dire quand les ensembles des solutions optimales de sum-abs et sum-sqr sont disjoints. Pour montrer ce résultat, on peut pour chaque problème, générer le graphe biparti pondéré G_w et tous les couplages parfaits minimaux de

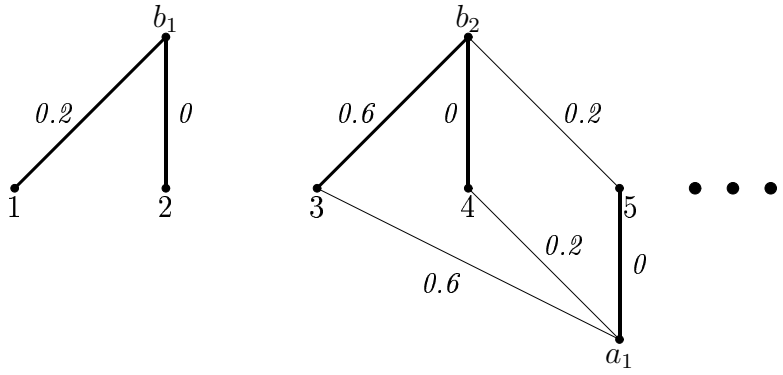


FIG. 5.4 – Un sous graphe du graphe biparti $d_1 = d_2 = 1$, $d_3 = d_4 = 4$

ce graphe. On peut le faire en utilisant l'algorithme proposé dans [20]. Il est parfois plus rapide d'utiliser le Corollaire 7 comme dans l'Annexe A.4.1.

5.3 Conclusion et extensions possibles

Nous avons étudié ici la possibilité d'optimiser deux critères simultanément pour le problème d'ordonnancement juste-à-temps. Nous nous sommes intéressés à trois fonctions objectif : le maximum des déviations (max-abs), la déviation totale (sum-abs) et la somme des déviations au carré (sum-sqr). Nous avons aussi étudié l'optimisation de sum-abs (resp. sum-sqr) en bornant le maximum des déviations par 1. Dans la plupart des cas, nous proposons des instances pour lesquelles il n'y a pas de solution qui optimise simultanément ces critères. Cependant, nous avons essayé de donner une vue d'ensemble détaillée de l'interaction entre ces critères.

La conjecture qui a donnée lieu à ces travaux était la suivante [58] :

Conjecture 1 [58] *Pour toute instance du problème juste-à-temps, il existe une solution qui optimise simultanément le maximum des déviations (max-abs) et la déviation totale (sum-abs).*

L'existence de contre-exemples avec une demande totale D supérieure à 100 avaient déjà été indiquée [56], mais aucun de ces contre-exemples n'avaient été fournis. Nous avons donné ici le contre-exemple de plus petite demande ($D = 10$) : l'instance $d = (1, 1, 4, 4)$ ainsi qu'une méthode pour montrer qu'il l'est bien.

Puisque pour toute instance il existe une solution dont le maximum des déviations est inférieur à 1 [82, 18] la conjecture suivante est plus faible.

Conjecture 2 [83] *Pour toute instance, il existe une séquence optimale pour sum-abs dont le maximum des déviations est inférieur à* 1.

Cette conjecture est aussi étudiée dans [56]. Les auteurs concluent alors que cette conjecture est vraie, mais des tests leurs ont permis plus tard de trouver un contre-exemple avec $D = 100$ (énoncé dans [35]). Dans la Figure 5.2 et la Table 5.2 nous proposons plu-

sieurs contre-exemples de demande totale inférieure que nous utilisons pour des résultats plus forts.

Le Théorème 13, déjà montré dans [35], est lié à la Conjecture 2. Il indique que *si* une solution avec max-abs inférieure à 1 est optimale pour sum-abs, *alors* elle est aussi optimale pour sum-sqr.

Les exemples $2, 3, 5, 6$ de la Table 5.2 montrent que les rôles de sum-abs et sum-sqr ne peuvent pas être inversés dans le Théorème 13.

La conjecture suivante, liée elle aussi à la Conjecture 2 est toujours ouverte, même si nous l'avons testée pour toutes les instance avec $D \leq 100$.

Conjecture. *Pour $n = 3$ il existe des instances telles que sum-abs n'a pas de solution optimale dont le maximum des déviations est inférieur à* 1.

La Section 5.2.2 ne présente aucun exemple avec $(AM, SM, AM1, SM1, AS) = (F, F, F, F, F)$. Nous n'en avons pas trouvé et on peut se demander si au moins une des variable $SM1$ et AS est toujours vraie.

Dans la Proposition 19 nous faisons remarquer que minimiser le maximum des déviations est équivalent à minimiser le maximum de fonctions plus générales, et en particulier équivalent à minimiser le carré des déviations.

Proposons maintenant des questions suggérées par notre travail pour lesquelles nous ne donnons pas directement de réponses.

Il est probable que si une solution du problème sum-abs a un maximum des déviations inférieur à 1, cela n'impliquera pas nécessairement que toutes les solutions optimales de sum-abs auront cette propriété.

Comme les problèmes sum-abs et sum-sqr peuvent n'avoir aucune solution 1-bornée optimale, les améliorations proposées par Steiner et Yeomans dans [83] ne peuvent pas être appliquées dans le cas général, mais l'étude des solutions 1-bornées peut montrer qu'elles peuvent approximer les solutions non bornées avec un faible ratio.

Un autre problème intéressant est de borner le maximum des déviations sous la contrainte que la déviation totale reste minimale : nous savons qu'il n'est pas possible d'imposer au maximum des déviations d'être inférieur à 1 mais est-il possible de remplacer 1 par une autre constante ?

L'Annexe A.1.2 propose des heuristiques polynomiales basées sur la structure des graphes associés au problème. L'étude de leurs performances n'a pas pu être menée par manque de temps mais pourrait être envisagée pour de futures recherches, ainsi que la recherche d'améliorations.

Chapitre 6

Tâches couplées

La notion de tâches couplées a été introduite par Shapiro [81] en 1980 pour décrire l'ordonnancement des opérations d'un radar. Celui-ci émet des signaux électromagnétiques qui sont réfléchis par les cibles potentielles avant d'être reçus par le radar qui doit alors analyser les informations recueillies. Le radar effectue donc deux opérations par tâche ou signal : l'émission et la réception. Ces deux opérations sont nécessairement séparées par un laps de temps puisque le signal parcourt deux fois la distance entre le radar et la cible avant de revenir à son point de départ.

Une tâche couplée est donc une tâche composée de deux opérations qui sont réalisées sur la même machine mais pas immédiatement l'une après l'autre puisqu'elles sont séparées par un temps d'attente fixé.

On souhaite minimiser le makespan ou la durée moyenne du cycle de production dans le cas cyclique. Nous notons (TC) ces deux problèmes et préciserons si besoin le critère concerné.

Dans un premier temps, nous présentons l'état de l'art relatif à l'ordonnancement des tâches couplées (Section 6.1), puis l'équivalence avec un problème particulier de cellules robotisées (Section 6.2). Nous nous intéressons au cas où les tâches couplées sont identiques et nous étudions l'algorithme proposé par Ahr et al. [6] pour lequel nous proposons des améliorations (Section 6.3). Enfin, nous examinons le cas cyclique et proposons une conjecture concernant sa complexité (Section 6.4).

6.1 État de l'art et notations du problème

Une contrainte de précédence classique entre deux opérations impose que l'une ne puisse commencer avant que l'autre ne soit terminée. Si on note $R(o_k)$ l'instant de départ de l'opération o_k, on peut traduire le fait que o_i précède o_j par l'équation suivante :

$$R(o_i) + p_i \leq R(o_j)$$

Roy [77] introduit une généralisation des contraintes de précédence en proposant de remplacer p_i par un entier quelconque $l_{i,j}$.

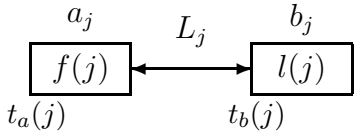
$$R(o_i) + l_{i,j} \leq R(o_j) \tag{6.1}$$

On appelle la durée $l_{i,j}$ *time-lag* en anglais. Si $l_{i,j}$ est positif, o_j ne peut pas commencer moins de $l_{i,j}$ unités de temps après le départ de o_i. On retrouve ce type de contraintes par exemple dans la métallurgie où on attend le refroidissement des pièces avant d'effectuer l'opération suivante. Si $l_{i,j}$ est négatif, l'opération o_j ne peut pas commencer plus tôt que $|l_{i,j}|$ unités de temps avant le début de o_i. Cette contrainte intervient dans l'industrie alimentaire où les biens périssables doivent être acheminés aux distributeurs avant qu'un délais trop important ne soit écoulé. Brucker et al. [28] et Brucker et Knust [29] ont montré que de nombreux problèmes d'ordonnancement pouvaient se réduire vers des problèmes à une unique machine et time-lags. Une conséquence de ce résultat est que la plupart des problèmes avec contraintes de précédence généralisées sont $\mathcal{NP}$-Difficiles, même sur une seule machine (voir [84, 29] pour la complexité de ces problèmes).

Dans ce chapitre, nous étudions un problème particulier à une machine avec contraintes de précédence généralisées : le problème des tâches couplées. Chaque tâche est composées de deux opérations séparées par un laps de temps fixé. Ce problème trouve son origine dans la planification des opérations d'un radar [81]. La première opération de chaque tâche correspond à l'émission du signal, la seconde à sa réception, le laps de temps qui les sépare représente la durée mise par le signal pour atteindre la cible du radar et être réfléchi. Orman et al. [72] présentent une étude de cas plus détaillée qui considère un système radar multifonction. Ce système est utilisé dans un contexte militaire pour permettre le guidage des armes, rechercher des cibles potentielles ou surveiller des espaces prédéfinis.

Gupta [44] décrit plusieurs autres applications potentielles de ce modèle. Dans l'industrie chimique, on peut utiliser la même machine pour réaliser plusieurs opérations de la même tâche, le laps de temps entre ces opérations correspondant à la durée des réactions chimiques. Dans un atelier flexible, où une station peut effectuer différents types d'opérations, certains réajustements de la pièce peuvent être nécessaires entre deux opérations successives. Pendant que ces ajustements sont effectués sur une station dédiée à cette effet, la machine elle-même peut être utilisée pour réaliser d'autres opérations.

Nous utilisons pour ce problème les notations suivantes :
- n est le nombre de tâches à effectuer dans le cas non cyclique ;
- $f(j)$ est la première opération de la tâche j et $l(j)$ sa seconde opération ;
- a_j est la durée de $f(j)$ et b_j la durée de $l(j)$;
- $t_a(j)$ est l'instant de départ de $f(j)$ et $t_b(j)$ l'instant de départ de $l(j)$;
- L_j, le laps de temps séparant la fin de $f(j)$ et le début de $l(j)$.



La relation de précédence entre les opérations $f(j)$ et $l(j)$ de la tâche j s'écrit alors

$$\begin{cases} t_a(j) + (a_j + L_j) \leq t_b(j) \\ t_b(j) - (a_j + L_j) \leq t_a(j) \end{cases}$$

si la durée entre les deux opérations doit être exactement de taille L_j, et

$$\begin{cases} t_a(j) + (a_j + u_j) \leq t_b(j) \\ t_b(j) - (a_j + v_j) \leq t_a(j) \end{cases}$$

si la durée entre les deux opérations doit appartenir à l'intervalle $[u_j, v_j]$ (voir [28, 49, 75] pour des heuristiques pour ce type de problèmes).

Le problème des tâches couplées a été largement étudié en tant que tel pour ses nombreuses applications, et un nombre important de résultats de complexité ont été obtenus [71]. Nous nous intéressons dans ce chapitre au problème de tâches couplées pour lesquelles l'intervalle entre les deux opérations de la tâche j est exactement de L_j. Remarquons qu'il est possible d'inverser l'ordre de toutes les opérations des tâches pour obtenir un problème équivalent pour le critère $Cmax$ (ou pour la maximisation du taux de production). Par exemple, le problème avec $a_j = a; L_j; b_j$ est équivalent au problème avec $a_j; L_j; b_j = b$ et la Table 6.1 ne fait apparaître qu'un de ces problèmes symétriques.

Si les tâches couplées sont identiques ($a_j = a$, $L_j = L$, $b_j = b$), la complexité du problème est toujours ouverte (voir Table 6.1), bien que les auteurs aient montré dans [14] qu'ajouter des contraintes de précédence classiques entre les tâches rendait le problème difficile, et ce même si les durées d'usinage des opérations sont unitaires.

Le meilleur algorithme pour n tâches identiques, à notre connaissance, a pour complexité $O(nr^{2L})$, où $r \leq \sqrt[a-1]{a}$ [6]. Cet algorithme n'est pas polynomial de la taille de l'instance. Il devrait pour cela être de l'ordre de $O(\log(\max(n, a, L, b)))$ ou juste $O(\log(\max(a, L, b)))$. Dans la Section 6.3, nous proposons des améliorations substantielles de cet algorithme.

fortement $\mathcal{NP}$-Complet	a_j ; L_j ; b_j
	$a_j = L_j = b_j$
	$a_j = a$; L_j ; $b_j = b$
	$a_j = a$; $L_j = L$; b_j
ouvert	$a_j = a$; $L_j = L$; $b_j = b$
polynomial	$a_j = L_j = p$; b_j
	$a_j = b_j = p$; $L_j = L$

TAB. 6.1 – Complexité des problèmes d'ordonnancement de tâches couplées avec durée fixe L_j entre les opérations de la tâche j [71]

6.2 Des cellules robotisées aux tâches couplées

Nous montrons dans cette section l'équivalence entre le problème des tâches couplées et un problème de cellules robotisées à une machine de capacité infinie.

6.2.1 Configuration de la cellule

Nous considérons ici une cellule robotisée composée d'une station d'entrée (notée IN), d'une station de sortie (notée OUT), et d'une machine (notée M). La machine M peut traiter un nombre quelconque de pièces simultanément (il peut par exemple s'agir d'un bain chimique dans lequel elles seraient trempées). La Figure 6.1 représente cette configuration.

La Section 6.2 a fait l'objet d'une publication [23]

Nous supposons que la capacité du robot est unitaire et que les durées de transport sont additives. On considère le problème sans attente : une fois son usinage terminé sur M, la pièce doit être immédiatement retirée de la machine par le robot. Les durées de chargement et déchargement des pièces sont nulles.

Les notations sont les suivantes :
- A_j est la durée du transport de la pièce j de IN à M ;
- B_j est la durée du transport de la pièce j de M à OUT ;
- p_j est la durée d'usinage de la pièce j sur la machine M ;
- α et β sont les durées de retour à vide du robot entre M et IN et OUT et M respectivement.

On souhaite produire n tâches en minimisant le makespan $Cmax$ ou maximiser le taux de production dans le cas cyclique.

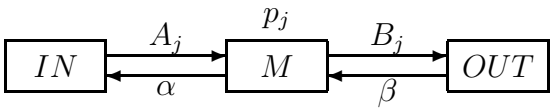


FIG. 6.1 – Cellule robotisée à une machine

6.2.2 Équivalence

Le problème défini à la Section 6.2.1 a été étudié indépendamment du problème des tâches couplées. Cependant, les deux problèmes sont en fait équivalents.

Théorème 14 *Le problème des tâches couplées est équivalent au problème de cellule robotisée sans attente à une machine de capacité infinie.*

Démonstration. On définit les deux activités du robot suivantes :
- U_j : le robot vide se déplace de M à IN (α unités de temps), retire la pièce j de IN, la transporte et la charge sur la machine M (A_j unités de temps) ;
- V_j : le robot retire la pièce j de la machine M et la transporte jusqu'à OUT où il la décharge (B_j unités de temps). Le robot qui est maintenant vide retourne à M (β unités de temps).

Sans perte de généralité, on peut supposer que si le robot attend, alors c'est uniquement à la machine. Sous cette hypothèse, aucun temps d'attente n'intervient durant les activités U_j et V_j. Une fois qu'il se trouve à la machine M, le robot peut uniquement effectuer l'une de ces deux activités. On suppose que les durées de transport sont additives et donc que le robot repasse par la machine M pour se rendre de OUT à IN. Le problème étant sans attente, le temps écoulé entre les activités U_j et V_j est exactement p_j. En fixant $a_j = \alpha + A_j$, $b_j = \beta + B_j$ et $L_j = p_j$, on obtient une instance de (TC). On ordonnance les tâches couplées pour cette instance exactement dans le même ordre que celui des pièces correspondantes de la cellule robotisée.

Inversement, si on part d'une instance $\{a_j, L_j, b_j\}$ de (TC), on considère une cellule robotisée où les retours à vide sont de durées nulles. On peut en fait utiliser n'importe quelles durées $\alpha \in [0, \min a_j]$ et $\beta \in [0, \min b_j]$ pour ces déplacements et poser $A_j = a_j - \alpha$, $B_j = b_j - \beta$. Une fois de plus, les deux problèmes sont identiques. □

Cette équivalence permet de déduire des résultats de la Table 6.1 la complexité des problèmes de cellules robotisées correspondants. Ils sont résumés dans la Table 6.2.

fortement $\mathcal{NP}$-Difficile	$A_j\,;\,p_j\,;\,B_j$	pièces différentes, durées de transport dépendantes des pièces
	$A_j+\alpha=p_j=B_j+\beta$	pièces différentes, durées de transport et d'usinage liées
	$A_j=A\,;\,p_j\,;\,B_j=B$	pièces différentes, durées de transport constantes
	$A_j=A\,;\,p_j=p\,;\,B_j$	durées d'usinage identiques
ouvert	$A_j=A\,;\,p_j=p\,;\,B_j=B$	pièces identiques
polynomial	$A_j+\alpha=p_j=p\,;\,B_j$	durées d'usinage identiques
	$A_j+\alpha=B_j+\beta=p_j=p$	durées d'usinage et de transport liées et identiques

TAB. 6.2 – Complexité pour les cellules robotisées no-wait à une machine de capacité infinie

6.3 Algorithme exact pour les tâches couplées identiques

Nous nous intéressons dans cette section au problème avec tâches identiques et à l'algorithme proposé par Ahr et al. [6] pour le cas non cyclique.

Pour ce problème, nous simplifions les notations de la façon suivante.
– n est le nombre de tâches ;
– a est la durée de leur première opération ;
– b est la durée de leur seconde opération ;
– L est le temps qui les sépare.
On peut supposer que $a \geq b$, sinon, il suffit de considérer le problème symétrique. Nous décrivons brièvement l'algorithme de [6], proposons des améliorations et l'adaptons pour traiter le problème cyclique.

6.3.1 Description

Étant donnés (a, L, b) et une séquence des pièces, il est possible d'associer à chaque tâche un *motif* de longueur L. La Figure 6.2 présente l'exemple d'une séquence de 4 tâches et les motifs associés sous la forme de suites de '0' et de '1'.

Un $'0'$ indique que cette position de L n'est pas occupée et un bloc de $'1'$ de longueur b signifie que cette portion de L contient la seconde opération d'une des tâches qui précèdent la tâche courante dans la séquence. Puisque le laps de temps entre deux opérations d'une même tâche est exactement de longueur L, on peut déduire la position de la première

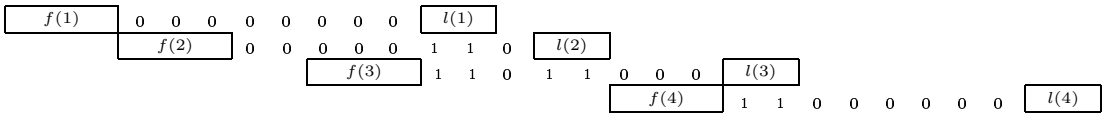


FIG. 6.2 – Séquence de motifs

opération de la présence de ces blocs de '1'. Dans la Figure 6.2 les deux tâches du milieu, la tâche 2 et la tâche 3, ont les motifs suivants :

$m_2 = 00000110$ et $m_3 = 11011000$

Le motif m_3 appartient au successeur immédiat de la tâche 2. Ainsi, on sait que le dernier bloc de '1' dans m_3 doit nécessairement indiquer la position de l'opération $l(2)$. Par conséquent, on a en déduit aussi l'intervalle qui sépare le début des deux tâches 2 et 3. Une séquence de motifs donnée définit donc un unique ordonnancement des pièces et réciproquement, à chaque ordonnancement correspond une séquence de motifs.

On introduit le graphe $G = (V, A)$ suivant. L'ensemble V des sommets est composé de tous les motifs possibles. L'arc (m_i, m_j) de poids w appartient à l'ensemble A des arcs de G si en plaçant une tâche à distance w d'une tâche dont le motif serait m_i, on obtient une tâche de motif m_j (comme expliqué pour les motifs m_2 et m_3 de la Figure 6.2 ; le poids de l'arc (m_2, m_3) est 5). Dans [6], les auteurs introduisent le lemme suivant pour réduire le nombre de sommets dans le graphe dans le cas non cyclique. Leur démonstration est basée sur un argument d'échange.

Lemme 15 *On peut considérer qu'on a au plus $a+b-2$ temps morts entre deux opérations consécutives.*

Afin d'évaluer la rapidité de leur algorithme, les auteurs proposent une valeur approximative du nombre de sommets ainsi obtenus dans V, qui intervient dans la complexité de leur algorithme. Ce nombre est en fait donné par la formule suivante :

$$|V| = 1 + \sum_{i=1}^{\lfloor \frac{L}{a} \rfloor} \prod_{k=1}^{i} \frac{L - ia + k}{k} \tag{6.2}$$

Trouver la position optimale de n tâches couplées correspond à trouver le plus court chemin $\mathcal{L}(n)$ passant par n sommets distincts du graphe $G(V, A)$ et commençant par le sommet $m_0 = 0$. La durée optimale de l'ordonnancement est alors donnée par $|\mathcal{L}(n)| + (a + L + b)$. Dans [6], la détermination du plus court chemin est faite d'une façon peu économe, puisque pour chaque sommet, presque tous les successeurs et les poids des arcs sont mémorisés. La section suivante montre qu'on peut en fait améliorer cette structure en faisant un choix plus judicieux concernant le stockage des poids et en limitant le nombre de sommets utiles.

6.3.2 Améliorations

Afin de trouver une structure appropriée pour l'implémentation de l'algorithme, nous faisons les observations suivantes. On considère un motif m_i. Alors le dernier bloc de $'1'$ de m_i appartient à la seconde opération de **tout prédécesseur** m_j. Ainsi, tous les arcs arrivant en m_i ont le même poids w. On peut remarquer que, pour le cas $m_i = 0$, une nouvelle tâche est placée sans enchevêtrement des opérations et $w = a + L + b$. Il est donc bien plus efficace de placer le poids de tous arcs entrant dans un sommet sur ce sommet plutôt que sur les arcs. Avec cette structure, on peut éliminer un grand nombre de sommets et d'arcs comme le montre le lemme suivant.

Lemme 16 *Que l'on soit dans le cas cyclique ou non cyclique, il est suffisant de considérer les arcs de poids w tel que $w = a$ ou $w \geq a + b$.*

Démonstration. Supposons que $b \geq 2$. Soit i une tâche telle que $a < w_i < a + b$. On ne peut placer aucune autre tâche entre $f(i)$ et $f(i-1)$ ou entre $l(i)$ et $l(i-1)$ (voir Figure 6.3). On obtient alors un meilleur ordonnancement en décalant le sommet i vers la gauche jusqu'à obtenir $w_i = a$.

FIG. 6.3 – Simplification du graphe pour $a < w < a + b$

□

Ce lemme permet donc d'éliminer tous les sommets de poids $w \in \{a+1, a+2, \ldots, a+b-1\}$ lorsque $b \geq 2$.

Le nombre de sommets du graphe (voir équation (6.2)) montre que le degré de difficulté du problème est influencé par la taille de $\lfloor \frac{L}{a} \rfloor$. La taille du graphe explose lorsque $\lfloor \frac{L}{a} \rfloor$ augmente. La Table 6.3 nous donne une indication sur la possibilité de résoudre de façon exacte le problème cyclique pour lequel nous adaptons l'algorithme à la section suivante (sur un Pentium 4, CPU 2.53GHz, RAM, 1Go).

6.3.3 Adaptation au cas cyclique

Lorsque l'on produit un seul type de pièces dans un atelier de production, il est courant d'essayer de maximiser le taux de production de la cellule plutôt que le makespan. Le nombre de pièces est alors très grand et non connu à l'avance. L'objectif est donc de déterminer le cycle de production qui maximise le taux de production.

Ce problème sera étudié en détail dans la section suivante où une conjecture est proposée pour le résoudre en temps polynomial. Il est possible de modifier simplement l'algorithme de [6] pour qu'il résolve de façon exacte le cas cyclique, ce qui nous a permis de tester cette conjecture.

L	$\lfloor L/a \rfloor$	Nb de sommets	Nb de sommets pour le graphe simplifié	Temps pour obtenir une solution
10	2	8	6	$<$ 1 sec
12	2	15	8	$<$ 1 sec
14	2	26	14	$<$ 1 sec
16	3	45	22	$<$ 1 sec
18	3	80	36	$<$ 1 sec
20	4	140	58	$<$ 1 sec
22	4	245	92	$<$ 1 sec
24	4	431	151	$<$ 1 sec
26	5	756	241	$<$ 1 sec
28	5	1326	391	$<$ 1 sec
30	6	2328	627	$<$ 1 sec
32	6	4085	1013	$<$ 1 sec
33	6	5441	1288	$<$ 1 min
34	6	7168	1634	$<$ 1 min
35	7	9496	2071	$<$ 1 min
36	7	12580	2632	$<$ 1 min
37	7	16665	3344	$<$ 1 min
38	7	22076	4246	$<$ 1 min
39	7	29244	5389	$<$ 1 heure
40	8	38740	6839	$<$ 1 heure
41	8	51320	8688	$<$ 1 heure
42	8	67985	11034	$<$ 1 heure
43	8	90061	14007	non résolu

TAB. 6.3 – Résultats pour $a = 5, b = 3$

Nous décrivons ici brièvement la méthode employée afin d'expliquer les modifications qui seront nécessaires. Considérons l'instance $a = 3$, $L = 8$ et $b = 2$ du problème cyclique. Les Figures 6.4, 6.5 et 6.6 proposent trois cycles de production différents. La Figure 6.4 représente la position la plus simple pour les pièces, de la forme *aaabbb* avec une durée moyenne du cycle de $\frac{19}{3} = 6.33\ldots$. Dans la Figure 6.5 le cycle *abab* a une durée moyenne de $\frac{13}{2} = 6.5$. Le cycle optimal est en fait *aababbab* et est représenté sur la Figure 6.6. Il a une durée moyenne de $21/4 = 5.25$.

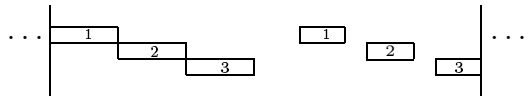


FIG. 6.4 – $L = 8$, $a = 3$, $b = 2$, solution ...*aaabbb*..., cycle de trois pièces, durée moyenne 19/3

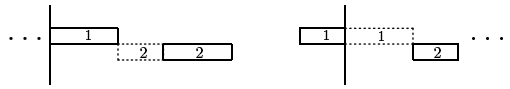


FIG. 6.5 – $L = 8$, $a = 3$, $b = 2$, solution ...*abab*..., cycle de deux pièces, durée moyenne 13/2

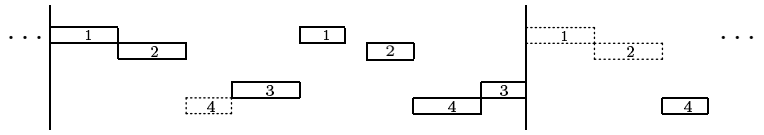


FIG. 6.6 – $L = 8$, $a = 3$, $b = 2$, solution ...*aababbab*..., cycle de quatre pièces, durée moyenne 21/4

Un cycle de production correspond à un circuit dans le graphe G. Cela signifie que sa longueur moyenne λ est définie comme la somme des poids des arcs du circuit divisée par le nombre de sommets. Le problème est donc de trouver le circuit tel que λ soit minimale. Ce problème déjà été longuement étudié et résolu efficacement. On le trouve dans la littérature sous le nom de PROBLÈME DU PLUS COURT CYCLE MOYEN [53] (SHORTEST MEAN CYCLE PROBLEM en anglais).

Il suffit donc d'appliquer l'algorithme de [53] au graphe G définit par Ahr et al. [6], en gardant les mêmes modifications que précédemment, pour obtenir la solution optimale au problème. Il n'est malheureusement pas possible de traiter des instances de très grande taille puisque le graphe, même réduit, n'est pas polynomial de la taille de l'instance associée.

6.4 Conjecture pour le cas cyclique avec tâches identiques

La conjecture présentée ici a été testée en utilisant l'algorithme décrit à la section précédente pour plus de 10000 instances telles que $70 \geq L > a + b$ et n'a pas été invalidée. Cependant, elle n'a pas pu être démontrée, et ce malgré les propriétés qui abondent dans son sens.

6.4.1 Notations et définitions

Dans le cas cyclique, l'instance consiste en trois entiers positifs a, b et L. Comme nous l'avons dit précédemment, on peut considérer que $a \geq b$ et nous travaillerons sous cette hypothèse. De plus, afin de nous restreindre à des cas non triviaux, nous supposons que $a \neq b$ et $L > a + b$.

Définition 17 (Fenêtre d'une tâche) *La* **fenêtre** $W(j)$ **de la tâche** j *est l'intervalle* $\{t_a(j) + a, t_a(j) + a + 1, \ldots, t_b(j) - 1\}$.

Définition 18 *On dit qu'une opération de la tâche* j, $f(j)$ *ou* $l(j)$, *est* **suivie** *de l'opération* $f(k)$ *ou* $l(k)$ *d'une tâche* k *si aucune autre opération n'est effectuée sur la machine entre ces deux opérations. On dit que les deux opérations sont* **consécutives**. *L'opération qui suit une opération* o *est notée* $s(o)$.

Définition 19 *Soit* i *une tâche.*
On dit que l'opération $f(i)$ **appartient à l'intervalle** I, *et on note* $f(i) \in I$ *si* ($t_a(i) \in I$ *et* $t_a(i) + a - 1 \in I$).
On dit que l'opération $l(i)$ **appartient à l'intervalle** I, *et on note* $l(i) \in I$ *si* ($t_b(i) \in I$ *et* $t_b(i) + b - 1 \in I$).

On note $n_a(i)$ le cardinal de l'ensemble $\{f(j) | f(j) \in W(i)\}$, et $n_b(i)$ le cardinal de l'ensemble $\{l(j) | l(j) \in W(i)\}$ ($n_a(i)$ *est le nombre de 'a' produits dans l'intervalle de temps* $W(i)$ *et* $n_b(i)$ *le nombre de 'b' produits dans cet intervalle*).

Pour une séquence S de 'a' et de 'b', on notera S^k la séquence S répétée k fois et S^* la séquence S répétée à l'infini.

6.4.2 Écriture de L et cycle associé

Nous introduisons la notion *d'écriture* pour la durée L qui sépare deux opérations d'une même tâche de la façon suivante.

Définition 20 *Soient* α *et* β *des entiers positifs ou nuls. On dit qu'une écriture de* L *sous la forme* $L = \alpha\, a + \beta(a+b) + \gamma$ *est* **admissible** *si* $\gamma \geq 0$. *On appelle* $L = \alpha\, a + \beta(a+b) + \gamma$ *l'***écriture** *ou la* **forme** *de* L.
On dit qu'une forme est r**-admissible** *si* $0 \leq \gamma < b$ *ou* ($\alpha = 0$ *et* $0 \leq \gamma < a$).

Remarque. Il existe toujours une écriture r-admissible de L. Notons β le quotient de la division euclidienne de L par $(a+b)$ et γ son reste. Si $\gamma \geq a$, la forme $L = a + \beta(a+b) + (\gamma - a)$ est r-admissible et si $\gamma < a$, la forme $L = \beta(a+b) + \gamma$ est r-admissible.

Si L admet l'écriture admissible $L = \alpha\, a + \beta(a+b) + \gamma$ avec $\alpha + \beta \geq 1$, on peut faire correspondre à cette écriture le cycle $\mathcal{C}(\alpha, \beta) = ((a^{(\alpha+1)}\,(ba)^{\beta} b^{(\alpha+1)} (ab)^{\beta})^{\Gamma})^*$ si $\alpha \neq 0$ et $((ab)^{\Gamma})^*$ sinon, où Γ est entièrement déterminé par la valeur de γ lorsque les tâches sont placées au plus tôt. Si γ est nul, $\Gamma = 1$. Si γ est non nul, $\Gamma = \beta + 1$.

Deux exemples de cycles associés à une écriture de L sont représentés pour $\gamma = 0$ et $L = a + (a+b)$ sur la Figure 6.7 et pour $\gamma = 1$ et $L = a + (a+b) + 1$ sur la Figure 6.8.

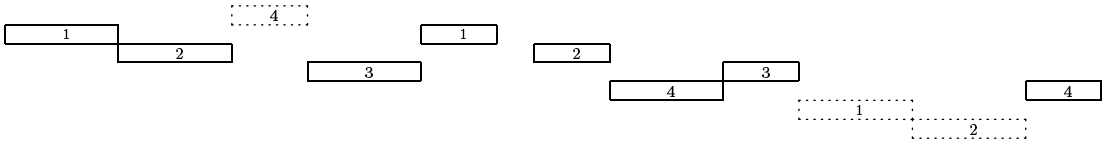


FIG. 6.7 – $\gamma = 0$, $L = a + (a+b)$, $\mathcal{C}(1,1) = (a^2bab^2ab)^*$

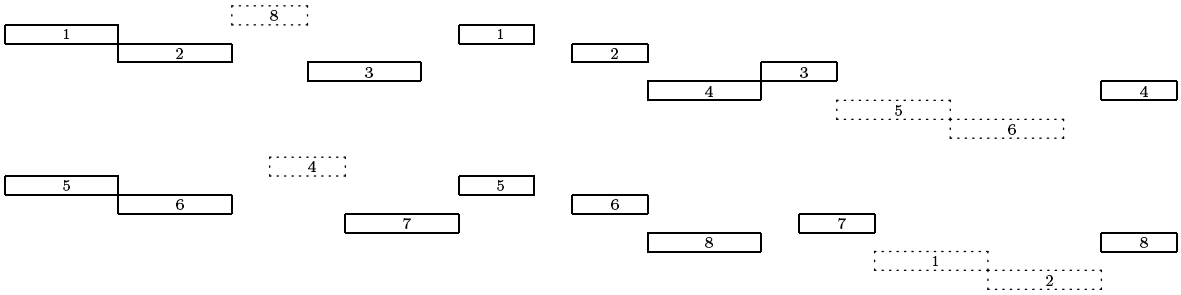


FIG. 6.8 – $\gamma = 1$, $L = a + (a+b) + 1$, $\mathcal{C}(1,1) = ((a^2bab^2ab)^2)^*$

Remarquons que si L admet l'écriture admissible $L = \alpha a + \beta(a+b) + \gamma$ avec $\gamma > 0$, dans un cycle de la forme $\mathcal{C} = ((a^{\alpha+1}(ba)^{\beta} b^{\alpha+1}(ab)^{\beta})^{\delta})^*$ avec $1 \leq \delta$, $\delta \neq \beta + 1$, toutes les tâches ne sont pas placées au plus tôt. Son taux de production ne sera donc pas meilleur que celui du cycle $\mathcal{C}(\alpha, \beta)$.

Nous conjecturons qu'il existe une écriture admissible de L telle que le cycle associé soit optimal.

Conjecture 1 *Pour tout instance (a, b, L) il existe une écriture admissible $L = \alpha\, a + \beta(a+b) + \gamma$ de L telle que le cycle $\mathcal{C}(\alpha, \beta)$ est optimal.*

On peut dès à présent formuler plusieurs remarques sur cette conjecture. La première est que le nombre d'écritures admissibles de L est supérieur à $\lceil L/a \rceil + 1$, ce qui n'est pas polynomial par rapport à la taille de l'instance. Il est donc nécessaire, même si la conjecture est admise, de déterminer quelles sont parmi ces écritures celles qui peuvent correspondre à un cycle optimal. La seconde est que si la conjecture est vérifiée, il nous faut encore connaître la durée moyenne du cycle pour répondre entièrement au problème.

Nous répondons tout d'abord à la seconde de ces deux interrogations.

Si on considère une écriture admissible pour L et le cycle associé $\mathcal{C}(\alpha, \beta)$, on peut calculer en temps polynomial les durée moyenne et durée moyenne d'inactivité de $\mathcal{C}(\alpha, \beta)$.

En effet, la durée de ce cycle est donnée par l'équation suivante :

$$T(\mathcal{C}(\alpha,\beta)) = \begin{cases} (\beta+1)(2L+a+b)-\gamma & \text{si } \gamma > 0 \text{ et } \alpha > 0 \\ L+a+b & \text{si } \gamma > 0 \text{ et } \alpha = 0 \\ 2L+a+b & \text{si } \gamma = 0 \text{ et } \alpha > 0 \\ a+b & \text{si } \gamma = 0 \text{ et } \alpha = 0 \end{cases} \tag{6.3}$$

Le nombre de tâches du cycle est :

$$N(\mathcal{C}(\alpha,\beta)) = \begin{cases} (\beta+1)(1+\alpha+2\beta) & \text{si } \gamma > 0 \text{ et } \alpha > 0 \\ 1+\beta & \text{si } \gamma > 0 \text{ et } \alpha = 0 \\ 1+\alpha+2\beta & \text{si } \gamma = 0 \text{ et } \alpha > 0 \\ 1 & \text{si } \gamma = 0 \text{ et } \alpha = 0 \end{cases} \tag{6.4}$$

La somme des temps morts survenus durant le cycle est :

$$TM(\mathcal{C}(\alpha,\beta)) = \begin{cases} (2\beta+1)\gamma + \alpha(1+\beta)(a-b) & \text{si } \alpha > 0 \text{ et } \gamma \neq 0 \\ \alpha(a-b) & \text{si } \alpha > 0 \text{ et } \gamma = 0 \\ \gamma & \text{si } \alpha = 0 \end{cases} \tag{6.5}$$

La durée moyenne du cycle est :

$$\lambda(\mathcal{C}(\alpha,\beta)) = \frac{(\beta+1)(2L+a+b)-\gamma}{(\beta+1)(1+\alpha+2\beta)} \tag{6.6}$$

Et la durée moyenne d'inactivité est :

$$\mu(\mathcal{C}(\alpha,\beta)) = \frac{(2\beta+1)\gamma + \alpha(1+\beta)(a-b)}{(\beta+1)(1+\alpha+2\beta)} \tag{6.7}$$

On peut donc, en temps polynomial par rapport à la taille de l'instance, connaître, pour un des cycles proposés par la conjecture, la durée moyenne du cycle et une représentation compacte.

Afin de comparer les différentes écritures de L dans la section suivante, nous introduisons la notion de dominance pour les cycles.

Définition 21 (Dominance d'un cycle par rapport à un autre) *On dit que le cycle $\mathcal{C}_1$* **domine** *le cycle $\mathcal{C}_2$ si la durée moyenne du cycle $\mathcal{C}_1$ est plus courte que celle du cycle $\mathcal{C}_2$, ou de façon équivalente si la durée moyenne des périodes d'inactivité dans $\mathcal{C}_1$ est plus courte que celle de $\mathcal{C}_2$. Cela signifie que l'une des deux équations équivalentes suivantes est vérifiée :*
- $\lambda(\mathcal{C}_1) \leq \lambda(\mathcal{C}_2)$
- $\mu(\mathcal{C}_1) \leq \mu(\mathcal{C}_2)$

La section suivante nous permet de choisir parmi les écritures de L celles qui peuvent correspondre au cycle optimal.

6.4.3 Bornes sur γ pour que l'écriture de L puisse être optimale

Dans une écriture de L, la valeur de γ correspond à la durée minimum d'inactivité dans la fenêtre de chaque tâche du cycle. Elle est bornée par les lemmes suivants dans le cas où le cycle considéré serait optimal.

Lemme 17 *L'écriture $L = \alpha\, a + \beta\,(a+b) + \gamma$ n'est pas optimale si $\gamma \geq a$.*

Démonstration. Montrons que si $\gamma \geq a$, alors $\mathcal{C}(\alpha, \beta)$ est dominé par $\mathcal{C}(\alpha+1, \beta)$.

Supposons $\gamma \geq a$. On compare les écritures $L = \alpha\, a + \beta\,(a+b) + \gamma$ et $L = (\alpha+1)\, a + \beta\,(a+b) + (\gamma - a)$. Soit Δ la différence entre les durées moyennes des deux cycles associés à ces écritures.

$$
\begin{aligned}
\Delta &= \lambda(\mathcal{C}(\alpha,\beta)) - \lambda(\mathcal{C}(\alpha+1,\beta)) \\
&= \frac{(\beta+1)(2L+a+b)-\gamma}{(\beta+1)(1+\alpha+2\beta)} - \frac{(\beta+1)(2L+a+b)-(\gamma-a)}{(\beta+1)(1+(\alpha+1)+2\beta)} \\
&= \frac{(\beta+1)(2L+a+b)[(2+\alpha+2\beta)-(1+\alpha+2\beta)] - \gamma(2+\alpha+2\beta) + (\gamma-a)(1+\alpha+2\beta)}{(\beta+1)(1+\alpha+2\beta)(2+\alpha+2\beta)} \\
&= \frac{(\beta+1)(2L+a+b) - \gamma(2+\alpha+2\beta) + (\gamma-a)(1+\alpha+2\beta)}{(\beta+1)(1+\alpha+2\beta)(2+\alpha+2\beta)} \\
&= \frac{\beta(2L+a+b) + (2\alpha\, a + 2\beta\,(a+b) + 2\gamma) - \gamma(2+\alpha+2\beta) + (\gamma-a)(1+\alpha+2\beta) + a + b}{(\beta+1)(1+\alpha+2\beta)(2+\alpha+2\beta)} \\
&= \frac{\beta(2L+a+b) + (\alpha+2\beta)(a-\gamma) + (\gamma-a)(1+\alpha+2\beta) + (\alpha+1)\, a + (2\beta+1)\, b}{(\beta+1)(1+\alpha+2\beta)(2+\alpha+2\beta)} \\
&= \frac{\beta(2L+a+b) + (\gamma-a) + (\alpha+1)\, a + (2\beta+1)\, b}{(\beta+1)(1+\alpha+2\beta)(2+\alpha+2\beta)} \\
&> 0
\end{aligned}
$$

puisque $\gamma - a \geq 0$.
Le cycle $\mathcal{C}(\alpha, \beta)$ n'est donc pas optimal : il est strictement dominé par $\mathcal{C}(\alpha+1, \beta)$. □

Lemme 18 *L'écriture $L = \alpha\, a + \beta\,(a+b) + \gamma$ n'est pas optimale si $\gamma \geq b$ et $\alpha \geq 1$.*

Démonstration. Si $\gamma \geq a$, le Lemme 17 donne le résultat.

Supposons que $a > \gamma \geq b$ et $\alpha \geq 1$. Montrons que $\mathcal{C}(\alpha, \beta)$ est dominé par $\mathcal{C}(\alpha-1, \beta+1)$.

On compare la représentation proposée avec $L = (\alpha-1)\, a + (\beta+1)(a+b) + (\gamma - b)$.
Soit Δ la différence entre les durées moyennes des deux cycles associés à $\mathcal{C}(\alpha, \beta)$ et $\mathcal{C}(\alpha -$

$1, \beta+1)$.

$$\begin{aligned}
\Delta &= \lambda(\mathcal{C}(\alpha,\beta)) - \lambda(\mathcal{C}(\alpha-1,\beta+1)) \\
&= \frac{(\beta+1)(2L+a+b)-\gamma}{(\beta+1)(1+\alpha+2\beta)} - \frac{(\beta+2)(2L+a+b)-(\gamma-b)}{(\beta+2)(1+(\alpha-1)+2(\beta+1))} \\
&= \frac{(\beta+1)(\beta+2)(2L+a+b)-\gamma(\beta+2)(2+\alpha+2\beta)+(\gamma-b)(\beta+1)(1+\alpha+2\beta)}{(\beta+1)(\beta+2)(1+\alpha+2\beta)(2+\alpha+2\beta)} \\
&= \frac{(\beta+2)[\beta(2L+a+b)+2L+a+b-\gamma(2+\alpha+2\beta)]+(\gamma-b)(\beta+1)(1+\alpha+2\beta)}{(\beta+1)(\beta+2)(1+\alpha+2\beta)(2+\alpha+2\beta)} \\
&= \frac{(\beta+2)[\beta(2L+a+b)+2\alpha a+2\beta a+2\beta b+a+b-\gamma(\alpha+2\beta)]}{(\beta+1)(\beta+2)(1+\alpha+2\beta)(2+\alpha+2\beta)} \\
&\quad + \frac{(\gamma-b)(\beta+1)(1+\alpha+2\beta)}{(\beta+1)(\beta+2)(1+\alpha+2\beta)(2+\alpha+2\beta)} \\
&= \frac{(\beta+2)[\beta(2L+a+b)+(a-\gamma)(\alpha+2\beta)+(\alpha+1)a+(2\beta+1)b]}{(\beta+1)(\beta+2)(1+\alpha+2\beta)(2+\alpha+2\beta)} \\
&\quad + \frac{(\gamma-b)(\beta+1)(1+\alpha+2\beta)}{(\beta+1)(\beta+2)(1+\alpha+2\beta)(2+\alpha+2\beta)} \\
&> 0
\end{aligned}$$

car $\gamma - b \geq 0$ et $a - \gamma > 0$.
Donc le cycle $\mathcal{C}(\alpha,\beta)$ n'est pas optimal : il est strictement dominé par $\mathcal{C}(\alpha-1,\beta+1)$. □

On peut déduire de ces deux lemmes que si la conjecture est vérifiée, le cycle optimal correspondra à une écriture r-admissible.

6.4.4 Deux cas résolus

Nous illustrons à présent la Conjecture 1 par deux exemples pour lesquels il est possible de démontrer de façon relativement concise qu'un des cycles proposés par la conjecture est optimal.

Proposition 21 *Dans le cas où L s'écrit sous la forme $L = \beta(a+b)$, le cycle $\mathcal{C}(0,\beta)$ est le seul cycle optimal.*

Démonstration. Le cycle $\mathcal{C}(0,\beta)$ n'a pas de temps mort, il est donc optimal. De plus, c'est le seul cycle dans ce cas. En effet, si dans un cycle $\mathcal{C}$ il existe une tâche j telle que $f(j)$ et $f(j+1)$ sont consécutives et non séparées par un temps mort, on aura une période d'inactivité de durée $a-b$ entre $l(j)$ et $l(j+1)$, donc le cycle n'est pas optimal. Ainsi, dans un cycle optimal, toute opération $f(j)$ est suivie d'une opération $l(k)$ qui est elle-même suivie de $f(j+1)$. De plus on a pas de temps morts entre ces opérations. Il s'agit donc du cycle $\mathcal{C}(0,\beta)$. □

Théorème 15 *Si L peut s'écrire $L = a + \beta(a+b)$ alors le cycle $\mathcal{C}(1,\beta)$ est optimal.*

Démonstration. Supposons que $L = a + \beta\,(a + b)$ avec $\beta \geq 1$.

Pour prouver ce résultat, on montre que pour toute autre solution, dans tout intervalle de durée $T(\mathcal{C}(1, \beta)) = 2L+a+b$, la somme des temps morts est au moins égale à $TM(\mathcal{C}(1, \beta))$, c'est-à-dire à $a - b$.

On considère un cycle admissible $\mathcal{C}$ de durée $T(\mathcal{C})$.
Soit $t \in \{0, 1, \ldots, T(\mathcal{C}) - 1\}$ et S la séquence de production obtenue en reproduisant $\mathcal{C}$ suffisamment de fois pour qu'en partant de l'instant t après le début du cycle, on obtienne une durée de production de $2L + a + b$.

On note I l'intervalle $\{t, t + 1, \ldots, t + 2L + a + b - 1\}$. Soit i l'indice tel que $t_a(i) = \min\{t_a(j) | t_a(j) + a \in I\}$.

Supposons que $\mathbf{t_a(i) \geq t + a}$. Dans ce cas
- Soit aucune opération ne termine sa production dans l'intervalle $\{t, t + 1, \ldots, t + a - 1\}$ et cet intervalle est alors une période d'inactivité de durée $a > a - b$.
- Soit une ou plusieurs opérations terminent dans l'intervalle $\{t, t + 1, \ldots, t + a - 1\}$. Alors, d'après la définition de l'indice i, il s'agit de 'b' (voir Figure 6.9) et on note $l(k)$ la première de ces opérations. Si aucune autre opération ne commence sa production dans l'intervalle $\{t, t + 1, \ldots, t + a - 1\}$, alors on a au moins $a-b$ temps morts puisque l'opération $l(k)$ dure b unités de temps.
  Sinon, $l(k + 1) \in I$ et $l(k + 1) = s(l_k)$. Donc on a au moins un temps mort de durée $a - b$ entre ces deux 'b'.

On a dans les deux cas $a - b$ temps morts dans l'intervalle $\{t, t + 1, \ldots, t + a - 1\}$. Or $\{t, t + 1, \ldots, t + a - 1\} \subseteq I$, donc on a au moins $a - b$ temps morts dans I.

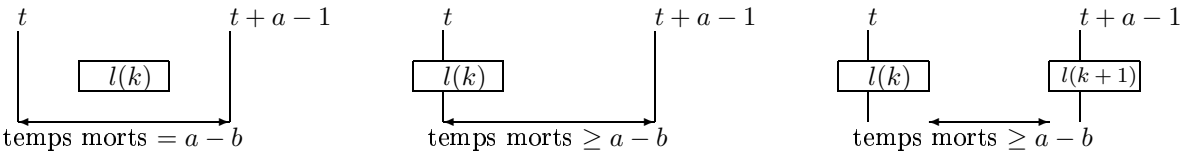


FIG. 6.9 – Temps morts dans un intervalle de taille a dans lequel aucune opération $f(j)$ ne commence ni ne finit

Supposons que $\mathbf{t_a(i) < t + a}$. La fenêtre $W(i)$ de la tâche i commence à l'instant $t_a(i) + a \in I$, par définition de i. Elle finit à l'instant $t_b(i) - 1 = t_a(i) + a + L - 1 < t + 2\,a + L - 1$. Or $L \geq a$ donc $t_b(i) - 1 < t + a + b + 2L - 1$. Ainsi, $t_b(i) - 1 \in I$ et $W(i) \subseteq I$.

S'il existe j tel que $l(j) \in W(i)$, $l(j + 1) \in W(i)$ et $s(l(j)) = l(j + 1)$ (c'est-à-dire que la fenêtre de i contient au moins deux 'b' consécutifs), alors on a un temps mort de durée supérieure ou égale à $a-b$ dans $W(i)$. Comme $W(i) \subseteq I$, on a un une période d'inactivité de durée supérieure ou égale à $a - b$ dans I.

Sinon, $n_a(i) \geq n_b(i)$. Si $n_a(i) = n_b(i)$ ou si $\forall j, \quad f(j) \in W(i) \Rightarrow w(i) \geq a + b$ (tout 'a' consécutif à un autre 'a' est séparé de celui-ci par au moins b temps morts), alors on a au moins a temps morts dans $W(i)$ et donc dans I car $L = a + \beta\,(a + b)$ (pour chaque opération $f(j)$ qui est de durée a, au moins b unités de temps sont utilisées, soit

pour produire une seconde opération $l(k)$, qui lui est consécutive, soit pour la séparer de l'opération $f(j+1)$ qui la suit).

Sinon, il existe k tel que $s(f(k)) = f(k+1)$ où $f(k)$ et $f(k+1)$ sont produites dans l'intervalle $\{t_a(i), t_a(i)+1, \ldots, t_b(i)-1\}$ et $w(k+1) = a$ (c'est-à-dire, il n'y a pas de temps morts entre les 2 'a'). Ainsi, entre $l(k)$ et $l(k+1)$, on a $a-b$ temps morts. Or $l(k) \in \{t_b(i), t_b(i)+1, \ldots, t_b(i)+L+b-1\}$ et $l(k+1) \in \{t_b(i), t_b(i)+1, \ldots, t_b(i)+L+b-1\}$. Donc l'intervalle $\{t_b(i), t_b(i)+1, \ldots, t_b(i)+L-1\}$ contient au moins $a-b$ temps morts.

Soit τ la durée des temps morts de l'intervalle $\{t, t+1, \ldots, t_a(i)-1\}$ si $t_a(i) \geq t$ et 0 sinon.
On a $t_a(i) \leq t+\tau+b$ donc $t_b(i) \leq t+\tau+a+b+L$. Ainsi, $t_b(i)+L-1 \leq t+a+b+2L+\tau$. Donc l'intervalle $\{t_b(i), t_b(i)+1, \ldots, t_b(i)+L-1\} \cap I$ contient au moins $a-b-\tau$ temps morts. Donc le temps mort total contenu dans I est supérieur ou égal à $\tau+a-b-\tau = a-b$.

Dans tous les cas, la durée totale d'inactivité contenue dans l'intervalle I est supérieure ou égale à $a-b$, c'est-à-dire supérieure ou égale au temps mort total qui survient pendant le cycle $\mathcal{C}(1, \beta)$, qui est de même durée que I. Le cycle $\mathcal{C}(1, \beta)$ est donc optimal. □

6.4.5 Précisions de la conjecture

Comme nous l'avons montré à la Section 6.4.3, il n'est pas nécessaire de considérer toutes les écritures admissibles de L puisque seules les formes r-admissibles ont la possiblibité d'être optimales.

Définition 22 *On appelle* **forme suivante** *d'une une forme r-admissible $L = \alpha a+\beta(a+b)+\gamma$, la forme r-admissible $L = \alpha' a + \beta'(a+b) + \gamma'$ ayant le plus petit $\alpha' > \alpha$.*

Définition 23 *On appelle* **forme précédente** *d'une forme r-admissible $L = \alpha a+\beta(a+b)+\gamma$, la forme r-admissible $L = \alpha' a + \beta'(a+b) + \gamma'$ ayant le plus grand $\alpha' < \alpha$.*

> **Remarque.** Dans les deux cas, les formes suivante et précédente sont uniques quand elles existent et on a $|\alpha' - \alpha| \geq 2$.

Nous montrons dans cette section qu'il est en fait possible de se restreindre pour chaque instance à au plus deux formes r-admissibles qui dominent toutes les autres. Nous utilisons plusieurs lemmes pour parvenir à ce résultat.

Lemme 19 *On considère une écriture de L sous la forme r-admissible $L = \alpha a + \beta(a+b)+\gamma$.*
Si $\gamma < (a-b)(\beta+1)$, alors pour toute forme r-admissible $L = \alpha' a + \beta'(a+b) + \gamma'$,

$$\alpha' \leq \alpha \Rightarrow \gamma' < (a-b)(\beta'+1)$$

Démonstration. Supposons que

$$\gamma < (a-b)(\beta+1) \tag{6.8}$$

Si $\alpha \leq 1$, la forme $L = \alpha a + \beta(a+b) + \gamma$ n'a pas de forme précédente, comme l'indique la remarque qui précède ce lemme, et le lemme est vrai.

Supposons $\alpha \geq 2$. Si $\gamma + (a-b) < b$, la forme $L = (\alpha-2)a + (\beta+1)(a+b) + \gamma + (a-b)$ est r-admissible et d'après (6.8), $\gamma + (a-b) < (\beta+2)(a-b)$.

Supposons que $\gamma + (a-b) \geq b$. Soit $k > 0$ tel que

$$\gamma + (a-b) - kb < b$$
$$\gamma + (a-b) - (k-1)b \geq b$$

- Si $\alpha < 2+k$, on considère la forme $L = (\beta+\alpha-1)(a+b) + \gamma + (a-b) - (\alpha-2)b$. On a $\gamma < b$ donc $\gamma + (a-b) - (\alpha-2)b < a$ et $L = (\beta+\alpha-1)(a+b) + \gamma + (a-b) - (\alpha-2)b$ est r-admissible. Il s'agit donc de la forme précédente de $L = \alpha a + \beta(a+b) + \gamma$ d'après la définition de k.
- Si $\alpha \geq 2+k$, on considère la forme $L = (\alpha-2-k)a + (\beta+1+k)(a+b) + \gamma + (a-b) - kb$ qui est r-admissible et est la forme précédente de $L = \alpha a + \beta(a+b) + \gamma$ d'après la définition de k.

La forme précédente de L sécrit donc $L = (\alpha-2-q)a + (\beta+1+q)(a+b) + \gamma + (a-b) - qb$ avec $q = \min\{\alpha-2, k\}$. On a $\gamma+(a-b)-qb < \gamma+(a-b) < (a-b)(\beta+2) < (a-b)(\beta+2+q)$ d'après (6.8).

On conclue en répétant le même raisonnement avec la forme précédente de $L = \alpha a + \beta(a+b) + \gamma$ lorsqu'elle existe. □

Le lemme suivant, qui ne sera pas utilisé par la suite, énonce le résultat symétrique pour les formes suivantes d'une forme r-admissible.

Lemme 20 *On considère une écriture de L r-admissible $L = \alpha a + \beta(a+b) + \gamma$.*
Si $\gamma > (a-b)(\beta+1)$, alors pour toute forme r-admissibles $L = \alpha' a + \beta'(a+b) + \gamma'$,

$$\alpha' > \alpha \Rightarrow \gamma' > (a-b)(\beta'+1)$$

Démonstration. Supposons que

$$\gamma > (a-b)(\beta+1) \tag{6.9}$$

Si $\beta = 0$, il n'existe aucune forme avec $\alpha' > \alpha$, donc le lemme est vrai.

Si $\beta \geq 1$, on cherche le plus petit α' tel que : $L = \alpha' a + \beta'(a+b) + \gamma'$ est une forme r-admissible et $\alpha' > \alpha$.
$L = (\alpha+1)a + (\beta-1)(a+b) + \gamma + b$ n'est pas r-admissible puisque $\gamma + b \geq b$.

Dans le cas où $\alpha \geq 1$, $L = (\alpha+2)a + (\beta-1)(a+b) + \gamma - (a-b)$ est une forme r-admissible car (6.9) implique que $\gamma > a-b$ et $\gamma < b$ donc $0 \leq \gamma - (a-b) < b$.

Dans le cas où $\alpha = 0$ si $\gamma - (a-b) < b$, la forme est r-admissible comme précédemment.

Dans les 2 cas, (6.9) implique que $\gamma - (a-b) > ((\beta-1)+1)(a-b)$.

Si $\alpha = 0$ et $\gamma - (a-b) \geq b$, la forme $L = (\alpha+2)a + (\beta-1)(a+b) + \gamma - (a-b)$ n'est pas r-admissible.

On considère l'entier $k > 0$ tel que

$$\begin{aligned} \gamma - (k+1)(a-b) &< b \\ \gamma - k(a-b) &\geq b \end{aligned}$$

La forme suivante de $L = \alpha a + \beta(a+b) + \gamma$ est donc $L = (\alpha + 2(1+k)) + (\beta - 1 - k)(a+b) + (\gamma - (1+k)(a-b))$ si $\beta \geq k+1$ et n'existe pas sinon.

Si $\beta \geq k+1$, la forme est bien admissible car (6.9) implique $\gamma - (k+1)(a-b) > ((\beta - (1+k)) + 1)(a-b)$.

On conclue en répétant ce raisonnement avec la forme suivante de $L = \alpha a + \beta(a+b) + \gamma$ lorsqu'elle existe. □

Le théorème suivant implique que si la Conjecture 1 est avérée, il se sera possible de déterminer en temps polynomial la solution optimale et la valeur optimale du taux de production.

Théorème 16 *Toute forme r-admissible de L est dominée par l'un des deux cycles associés aux écritures suivantes*
- $L = \alpha a + \gamma, \quad 0 \leq \gamma < b$
- $L = \varepsilon a + \beta(a+b) + \gamma, \quad \varepsilon \in \{0,1\},\ 0 \leq \gamma < a$ *si* $\epsilon = 0$ *et* $0 \leq \gamma < b$ *si* $\varepsilon = 1$

Démonstration.
- Si la forme $L = \alpha a + \gamma$ est r-admissible, on montre qu'elle domine toutes les formes r-admissibles $L = \alpha' a + \beta'(a+b) + \gamma'$ telles que $\gamma' \geq (a-b)(\beta'+1)$.
  Supposons qu'il existe une forme r-admissible $\mathcal{C} : L = \alpha' a + \beta'(a+b) + \gamma'$ telle que $\gamma' \geq (a-b)(\beta'+1)$. Si cette forme est $L = \alpha a + \gamma$ on a fini.
  Sinon, comparons-la avec les formes $\mathcal{C}_k : L = (\alpha' + 2k)a + (\beta' - k)(a+b) + (\gamma' - (a-b)k)$ telles que $k \leq \beta'$. Ces formes sont toutes admissibles car $\gamma' - (a-b)(\beta'+1) \geq 0$.

$$\begin{aligned} \Delta_k &= \lambda(\mathcal{C}) - \lambda(\mathcal{C}_k) \\ &= \frac{(\beta'+1)(2L+a+b) - \gamma'}{(\beta'+1)(1+\alpha'+2\beta')} - \frac{(\beta'-k+1)(2L+a+b) - (\gamma' - (a-b)k)}{(\beta'-k+1)(1+\alpha'+2k+2\beta'-2k)} \\ &= \frac{(\gamma' - (a-b)k)(\beta'+1) - \gamma'(\beta'-k+1)}{(\beta'+1)(\beta'-k+1)(1+\alpha'+2\beta')} \\ &= \frac{k(\gamma' - (a-b)(\beta'+1))}{(\beta'+1)(\beta'-k+1)(1+\alpha'+2\beta')} \\ &\geq 0 \end{aligned}$$

  En effet, on a $\gamma' \geq (a-b)(\beta'+1)$.
  Ainsi, $\mathcal{C}$ est dominé par $\mathcal{C}_k$ pour tout k, et en particulier par $\mathcal{C}(\alpha, 0)$.
- Si la forme $L = \alpha a + \gamma$, où $0 \leq \gamma < a$, n'est pas r-admissible, on considère l'entier k tel que

$$\begin{cases} \gamma - (k-1)b \geq b \\ \gamma - k\,b < b \end{cases} \tag{6.10}$$

Si $k > \alpha$, alors la seule forme r-admissible est $L = \alpha(a+b)+\gamma-\alpha b$ où $0 \leq \gamma - \alpha b < a$. Montrons que dans ce cas, elle domine toutes les autres formes admissibles $L = \alpha' a + \beta'(a+b)+\gamma'$. On remarque que pour toute forme $L = \alpha' a + \beta'(a+b)+\gamma'$, telle que $\alpha' \geq 0$, $\beta' \geq 0$ et $\beta' + \alpha' \leq \alpha$, on a $\gamma' = \gamma + (\alpha - (\alpha' + \beta'))a - \beta' b$ et que la condition $\beta' + \alpha' \leq \alpha$ est nécessaire pour qu'une écriture soit admissible. On a alors

$$
\begin{aligned}
\Delta &= \lambda(\mathcal{C}(\alpha',\beta')) - \lambda(\mathcal{C}(0,\beta)) \\
&= \frac{(\beta'+1)(2L+a+b)-\gamma'}{(\beta'+1)(1+\alpha'+2\beta')} - \frac{(\alpha+1)(2L+a+b)-(\gamma-\alpha b)}{(\alpha+1)(1+2\alpha)} \\
&= \frac{(\beta'+1)(\alpha+1)(2L+a+b)[2\alpha-(\alpha'+2\beta')]-\gamma'(\alpha+1)(1+2\alpha)}{(\beta'+1)(1+\alpha'+2\beta')(\alpha+1)(1+2\alpha)} \\
&\quad + \frac{(\gamma-\alpha b)(\beta'+1)(1+\alpha'+2\beta')}{(\beta'+1)(1+\alpha'+2\beta')(\alpha+1)(1+2\alpha)} \\
&\geq \frac{(\alpha+1)\{(\beta'+1)[2\alpha-(\alpha'+2\beta')](2\alpha+1)(a+b)-\gamma'(1+2\alpha)\}}{(\beta'+1)(1+\alpha'+2\beta')(\alpha+1)(1+2\alpha)} \\
&\geq \frac{(\alpha+1)(2\alpha+1)\{[2\alpha-(\alpha'+2\beta')](a+b)-\gamma-(\alpha-(\alpha'+\beta'))a\}}{(\beta'+1)(1+\alpha'+2\beta')(\alpha+1)(1+2\alpha)} \\
&\geq \frac{(\alpha+1)(2\alpha+1)\{[\alpha'+\alpha-(\alpha'+\beta')]a-\gamma\}}{(\beta'+1)(1+\alpha'+2\beta')(\alpha+1)(1+2\alpha)} \\
&\geq 0
\end{aligned}
$$

car $\gamma < a$ et soit $\alpha' > 0$, soit $\alpha - (\alpha' + \beta') > 0$ (pour que la forme $L = \alpha' a + \beta'(a+b)+\gamma'$ soit différente de $L = \alpha(a+b)+\gamma-\alpha b$). Le cycle $\mathcal{C}(\alpha',\beta')$ est donc dominé par $\mathcal{C}(0,\beta)$. Si $k \leq \alpha$, la forme r-admissible ayant le plus grand α est $L = (\alpha-k)a + k(a+b)+\gamma-kb$. Comme $\gamma < a$, on a $\gamma - b < a - b < (k+1)(a-b)$. Ainsi, d'après le Lemme 19 toutes les formes r-admissibles $L = \alpha' a + \beta'(a+b)+\gamma'$ vérifient

$$\gamma' < (\beta'+1)(a-b)$$

– Montrons que la forme r-admissible $L = \varepsilon a + \beta(a+b)+\gamma$ avec $\varepsilon \in \{0,1\}$, ($0 \leq \gamma < b$ si $\varepsilon = 1$) et ($0 \leq \gamma < a$ si $\varepsilon = 0$) domine toutes les formes r-admissibles $L = \alpha' a + \beta'(a+b)+\gamma'$ telles que $\gamma' < (a-b)(\beta+1)$.
Supposons qu'il existe une forme r-admissible $\mathcal{C}$: $L = \alpha' a + \beta'(a+b)+\gamma'$ telle que $\gamma' < (a-b)(\beta'+1)$. Si cette forme est $L = \varepsilon a + \beta(a+b)+\gamma$, on a fini.
On se place dans le cas où $\alpha' > 1$.
Soit $p > 0$ tel que $\alpha' = 2p + \varepsilon'$ avec $\varepsilon' \in \{0,1\}$. Soit q tel que $p \geq q > 0$, on compare les formes $L = \alpha' a + \beta'(a+b)+\gamma'$ et $L = (2(p-q)+\epsilon')a + (\beta'+q)(a+b)+\gamma'+q(a-b)$.
Soit $q' > 0$ l'entier tel que $\gamma' + q'(a-b) \geq b$ et $\gamma' + (q'-1)(a-b) < b$ (comme on a supposé que $L = \alpha' a + \beta'(a+b)+\gamma'$ est une forme r-admissible, on a $\gamma' < b$). Si $q' > 1$, alors toutes les formes $L = (2(p-q)+\epsilon')a + (\beta'+q)(a+b)+\gamma'+q(a-b)$ avec $q < q'$ sont r-admissibles. En particulier, la forme $L = (2(p-1)+\epsilon')a + (\beta'+1)(a+b)+\gamma'+(a-b)$ est r-admissible et est donc la forme précédente de $L = \alpha' a + \beta'(a+b)+\gamma'$.

$$\begin{aligned}
\Delta &= \lambda(\mathcal{C}) - \lambda(\mathcal{C}(2(p-1)+\epsilon', \beta'+1)) \\
&= \frac{(\beta'+1)(2L+a+b)-\gamma'}{(\beta'+1)(1+\alpha'+2\beta')} - \frac{(\beta'+2)(2L+a+b)-(\gamma'+(a-b))}{(\beta'+2)(1+\alpha'-2+2\beta'+2)} \\
&= \frac{\gamma'(\beta'+1)+(a-b)(\beta'+1)-\gamma'(\beta'+2)}{(\beta'+1)(\beta'+2)(1+\alpha'+2\beta')} \\
&= \frac{(a-b)(\beta'+1)-\gamma'}{(\beta'+1)(\beta'+2)(1+\alpha'+2\beta')} \\
&> 0
\end{aligned}$$

Si $q'=1$, $L=(2(p-1)+\epsilon')a+(\beta'+1)(a+b)+\gamma'+(a-b)$ n'est pas la forme précédente de $L=\alpha' a+\beta'(a+b)+\gamma'$ car elle n'est pas r-admissible. On définit l'entier k tel que

$$\begin{cases} \gamma+a-b-kb < b \\ \gamma+a-b-(k-1)b \geq b \end{cases}$$

comme dans la démonstration du Lemme 19. Alors, la forme précédente de $L=\alpha' a+\beta'(a+b)+\gamma'$ est définie par $\mathcal{C}_k : L=(\alpha'-2-k)a+(\beta'+1+k)(a+b)+\gamma'+a-b-kb$ si $\alpha' \geq 2+k$ et $\mathcal{C}_{\alpha'-2} : L=(\beta'+\alpha'-1)(a+b)+\gamma'+a-b-(\alpha'-2)b$ sinon. Dans ce dernier cas il s'agit donc de la forme $L=\varepsilon a+\beta(a+b)+\gamma$.
Pour tout $0<r\leq\alpha'-2$,

$$\begin{aligned}
\Delta_r &= \lambda(\mathcal{C}) - \lambda(\mathcal{C}_r) \\
&= \frac{(\beta'+1)(2L+a+b)-\gamma'}{(\beta'+1)(1+\alpha'+2\beta')} - \frac{(\beta'+2+r)(2L+a+b)-(\gamma'+(a-b)-rb)}{(\beta'+2+r)(1+\alpha'-2-r+2\beta'+2r+2)} \\
&= \frac{(\beta'+2+r)[r(1+\beta')(2L+a+b)-\gamma'(1+r+\alpha'+2\beta')]}{(\beta'+1)(1+\alpha'+2\beta')(\beta'+2+r)(1+r+\alpha'+2\beta')} \\
&\quad + \frac{(\gamma'+(a-b)-kb)(\beta'+1)(1+\alpha'+2\beta')}{(\beta'+1)(1+\alpha'+2\beta')(\beta'+2+r)(1+r+\alpha'+2\beta')}
\end{aligned}$$

Or $2r\,L=2r\,\alpha' a+2r\,\beta'(a+b)+2r\,\gamma' > (1+r+\alpha'+2\beta')\gamma'$ car $\gamma'<b$ et $r\geq 1$.
De plus on a $\gamma'+(a-b)-rb\geq 0$, donc $\Delta_r>0$. Ainsi, dans les deux cas sités précédement, la forme précédente de $\mathcal{C}$ domine $\mathcal{C}$.
Si la forme précédente $L=\alpha'' a+\beta''(a+b)+\gamma''$ de $L=\alpha' a+\beta'(a+b)+\gamma'$ n'est pas $L=\varepsilon a+\beta(a+b)+\gamma$, elle domine $L=\alpha' a+\beta'(a+b)+\gamma'$ et vérifie $\gamma''<(a-b)(\beta''+1)$ d'après le Lemme 19. On peut donc répéter le même raisonnement avec l'écriture $L=\alpha'' a+\beta''(a+b)+\gamma''$ pour conclure.

□

Corollaire 8 *Si la Conjecture 1 est vérifiée, alors il est suffisant de calculer les durées moyennes des deux cycles associés aux formes*
– $L=\alpha a+\delta$ *si* $0\leq\delta<b$;
– $L=\epsilon a+\beta(a+b)+\gamma$, *avec* $\epsilon\in\{0,1\}$, $0\leq\gamma<a$ *si* $\epsilon=0$ *et* $0\leq\gamma<b$ *si* $\epsilon=1$.
pour obtenir la solution optimale au problème.

Démonstration. Directement des Lemmes 17 et 18 et du Théorème 16. □

Remarque. On dispose alors en temps polynomial par rapport à la taille de l'instance du cycle optimal et du taux de production maximal si la conjecture est vérifiée.

On peut en fait déterminer, sans calculer les durées moyennes de leurs cycles associés, la meilleure des deux écritures $L = \alpha a + \delta$ et $L = \epsilon a + \beta(a+b) + \gamma$.

Proposition 22 *Considérons les deux écritures admissibles suivantes* $L = \alpha a + \delta$, $0 \leq \delta < a$ *et* $L = \epsilon a + \beta(a+b) + \gamma$, $\epsilon \in \{0,1\}$, $0 \leq \gamma < a$ *si* $\epsilon = 0$, $0 \leq \gamma < b$ *si* $\epsilon = 1$. *Le cycle* $\mathcal{C}(\alpha, 0)$ *domine le cycle* $\mathcal{C}(\epsilon, \beta)$ *si et seulement si*

$$\begin{cases} \alpha = \epsilon + 2\beta \\ a - b \leq \delta \end{cases}$$

Démonstration. Remarquons tout d'abord que $L = (\epsilon + \beta)a + \beta b + \gamma$ donc comme $b < a$, on a $\alpha \leq \epsilon + 2\beta$.

Le cycle $\mathcal{C}(\alpha, 0)$ domine le cycle $\mathcal{C}(\epsilon, \beta)$ si et seulement si

$$\begin{aligned} \lambda(\mathcal{C}(\alpha,0)) - \lambda(\mathcal{C}(\epsilon,\beta)) &\leq 0 \\ \frac{2L + a + b - \delta}{\alpha + 1} - \frac{(\beta+1)(2L+a+b) - \gamma}{(\beta+1)(1+\epsilon+2\beta)} &\leq 0 \\ \frac{(1+\beta)(\epsilon+2\beta-\alpha)(2L+a+b) + \gamma(\alpha+1) - \delta(1+\beta)(1+\epsilon+2\beta)}{(\alpha+1)(1+\beta)(1+\epsilon+2\beta)} &\leq 0 \end{aligned}$$

or $2L + a + b = 2\epsilon a + (2\beta+1)(a+b) + 2\gamma \geq (1 + \epsilon + 2\beta)a$ et $\alpha \leq \epsilon + 2\beta$ donc $\Delta \leq 0$ si et seulement si

$$\begin{cases} \alpha = \epsilon + 2\beta \\ \gamma \leq \delta(1+\beta) \end{cases}$$

or $\alpha = \epsilon + 2\beta$ est équivalent à $\gamma = \delta + \beta(a-b)$ donc $\Delta \leq 0$ si et seulement si

$$\begin{cases} \alpha = \epsilon + 2\beta \\ a - b \leq \delta \end{cases}$$

□

6.4.6 Remarques et propriétés

La Conjecture 1, bien qu'elle n'ait pas pu être démontrée, est encouragée par les propriétés que nous énonçons dans cette section.

Tout d'abord, remarquons que tout cycle calé à gauche qui ne possède aucune tâche de poids a est dominé.

Proposition 23 *Soit* (a, b, L) *une instance et* $L = \beta(a+b) + \gamma$, $0 \leq \gamma < a + b$ *une écriture admissible de* L. *Si* $\mathcal{C}$ *est un cycle tel que pour toute tâche* j *de* $\mathcal{C}$, $w(j) \geq a + b$, *alors* $\mathcal{C}(0, \beta)$ *domine* $\mathcal{C}$ *si* $\gamma < a$ *et* $\mathcal{C}(1, \beta)$ *domine* $\mathcal{C}$ *sinon.*

Démonstration. Dans cette démonstration, nous utiliserons les remarques suivantes. Tout d'abord, dans chaque fenêtre des tâches de $\mathcal{C}$, on a un temps mort total supérieur ou égal à γ puisque toutes les tâches sont de poids supérieurs à $a+b$. Ensuite deux 'a' consécutifs dans $\mathcal{C}$ sont séparés par un temps mort de durée supérieure ou égale à b et deux 'b' consécutifs par un temps mort supérieur ou égal à a.

- Supposons que $\gamma < a$. Considérons un intervalle I de longueur $T(\mathcal{C}(0,\beta)) = a+b+L$ qui commence à un instant t du cycle $\mathcal{C}$ (on peut répéter le cycle $\mathcal{C}$ si nécessaire). On note $TM(I)$ la somme des temps morts qu'il contient. Si I commence par une période d'inactivité, on peut considérer à la place de I l'intervalle de même longueur que I qui commence à l'instant de début de la première opération de I et qui ne peut contenir que moins de temps mort que I.
  Soit $f(i)$ tel que $t_a(i) = \min\{t_a(k) |\, \{t_a(k), t_a(k)+1, \ldots, t_a(k)+a-1\} \cap I \neq \emptyset\}$ ($f(i)$ est le premier 'a' de l'intervalle I).
  Si $t_a(i) \leq t+b-1$, alors la fenêtre de i, $W(i)$, est incluse intégralement dans I. Or, elle contient au moins γ temps morts donc $TM(I) \geq \gamma$.
  Si $t_a(i) > t+b-1$, par hypothèse, l'intervalle commence par une opération et non par du temps mort. Il s'agit donc d'un 'b'. On note $l(j)$ cette première opération.
  Si $s(l(j)) = f(i)$, alors I contient la fenêtre $W(i)$ privée d'une longueur égale au temps mort entre $l(j)$ et $f(i)$. On a donc encore une fois $TM(I) \geq \gamma$.
  Si $s(l(j)) = l(j+1)$, alors on a un temps mort entre $l(j)$ et $l(j+1)$ supérieur ou égal à a donc $TM(I) \geq \gamma$.
  Ainsi, dans tous les cas, $TM(I) \geq \gamma$ et $\mathcal{C}$ est dominé par $\mathcal{C}(0,\beta)$.
- Supposons que $\gamma \geq a$. On considère le cycle $\mathcal{C}_1 = (a^2\,(ba)^\beta\, b^2\,(ab)^\beta)^*$ calé à gauche. Ce cycle a un temps mort de $2(\gamma-a)+a-b = \gamma-(a+b-\gamma) < \gamma$ pour une longueur de $2L+a+b$ (il est donc dominé par $\mathcal{C}(1,\beta)$).
  Considérons un intervalle I de longueur $T(\mathcal{C}_1) = a+b+2L$ qui commence à un instant t du cycle $\mathcal{C}$ (on peut répéter le cycle $\mathcal{C}$ si nécessaire). On note $TM(I)$ la somme des temps morts qu'il contient. De même que précédemment, on peut supposer que I ne commence pas par du temps mort. On note encore $f(i)$ l'opération telle que $t_a(i) = \min\{t_a(k) |\, \{t_a(k), t_a(k)+1, \ldots, t_a(k)+a-1\} \cap I \neq \emptyset\}$.
  Si I commence par deux 'b', alors la durée de la période d'inactivité qui les sépare est supérieure à a et $TM(I) > \gamma > TM(\mathcal{C}_1)$.
  Si I ne commence pas par deux 'b', on montre comme précédemment, qu'il contient cette fois l'intégralité de la fenêtre $W(i)$ ou plus de γ temps morts et donc $TM(I) \geq \gamma > TM(\mathcal{C}_1)$.
  Dans les deux cas, le cycle est dominé par $\mathcal{C}_1$ et donc par $\mathcal{C}(1,\beta)$.

Le cycle $\mathcal{C}$ est donc dominé par $\mathcal{C}(0,\beta)$ si $\gamma < a$ et par $\mathcal{C}(1,\beta)$ sinon. □

Puisque les cycles ne possédant pas de tâche de poids a sont dominés par un des cycles proposés par la Conjecture 1, nous nous tournons vers les cycles contenant une tâche de poids a.

Par convention, la seconde opération d'un tel cycle $\mathcal{C}$ sera la première opération d'une tâche de poids a et la dernière sera un 'b' ($w(2) = a$ et $\mathcal{C} = (aa\mathcal{S}b)^*$ pour une certaine séquence $\mathcal{S}$ de 'a' et de 'b').

Nous nous intéressons au contenu de la fenêtre $W(1)$ de la première tâche du cycle. Pour

le décrire, nous introduisons les notations suivantes.

- Pour toute tâche i, on notera $S(i)$ la séquence de 'a' et de 'b' de la fenêtre $W(i)$;
- Pour toute séquence S de 'a' et de 'b', on note S^{-1} son inverse, c'est-à-dire la séquence de même longueur que S dans laquelle les 'a' sont remplacés par des 'b' et réciproquement.

On remarque que, pour toute écriture admissible $L = \alpha a + \beta(a+b) + \gamma$ telle que $\alpha \geq 1$, on a $S(i) = S(i + \alpha + 2\beta)$ et que le cycle associé s'écrit sous la forme $((aS(1)bS^{-1}(1))^{\Gamma})^*$. Cette propriété sera en fait retrouvée pour tout cycle $\mathcal{C}$ contenant une tâche de poids a si on limite la durée des temps morts dans chaque fenêtre des tâches de $\mathcal{C}$. Pour prouver cette affirmation, nous utilisons la définition suivante et plusieurs lemmes.

Définition 24 (Forme d'intervalle de longueur L'). *On considère un intervalle I qui ne contient que des opérations complètes. On écrit la longueur L' de l'intervalle I sous la forme $L' = \alpha' a + \beta'(a+b) + \gamma'$ où*

- *γ' est la somme des temps morts qui ne sont pas engendrés par une tâche de poids a, c'est-à-dire que si $s(l(j)) = l(j+1)$, on compte dans γ' une durée $w(i+1) - a$. Tout se passe comme si $l(j)$ était un 'a' (voir Figure 6.10) ;*
- *α' est le cardinal de l'ensemble $\{j | (s(f(j-1)) = f(j) \wedge f(j) \in I) \vee (s(l(j)) = l(j+1) \wedge l(j) \in I)\}$;*
- *β' est le cardinal de l'ensemble $\{j | s(l(k)) = f(j) \wedge f(j) \in I\}$.*

$L' = \alpha' a + \beta'(a+b) + \gamma'$ est alors la forme de l'intervalle *I.*

Cette notion de forme d'intervalle est utilisée dans le reste de la section pour décrire les formes des fenêtres des tâches. La Figure 6.10 donne un exemple de forme de fenêtre possible pour $L = 23$, $a = 3$ et $b = 2$.

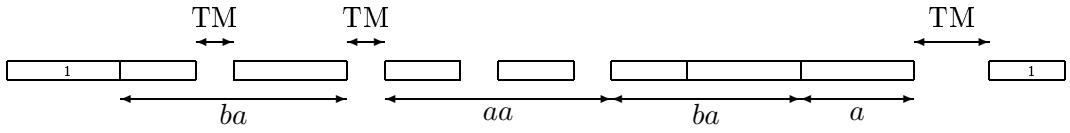


FIG. 6.10 – Forme de la fenêtre d'une tâche : L de la forme $L = 3\,a + 2\,(a+b) + 4$

On peut maintenant énoncer le lemme suivant.

Lemme 21 *On considère un cycle $\mathcal{C}$ tel que $TM(W(j)) \leq a-1$ pour toute tâche j dans $\mathcal{C}$ (où $TM(W(j))$ est la somme des temps morts dans la fenêtre de j). Soit i une tâche de $\mathcal{C}$. Si $W(i)$ admet la forme $L = \alpha' a + \beta'(a+b) + \gamma'$ avec $\gamma' < b$, alors la fenêtre $W(i+1)$ admet la même forme que $W(i)$.*

Démonstration. On note $F(i)$ l'ensemble des tâches j telles que $s(f(j-1)) = f(j)$ avec $f(j) \in W(i)$ et $L(i)$ l'ensemble des tâches j telles que $s(l(j)) = l(j+1)$ avec $l(j) \in W(i)$.

Remarquons que lorsque les temps morts totaux de $W(i)$ et $W(i+1)$ sont inférieurs à a, il suffit de montrer que

$$|F(i)| + |L(i)| = |F(i+1)| + |L(i+1)|$$

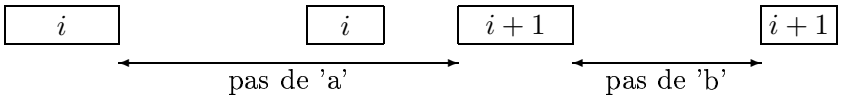


FIG. 6.11 – $f(i+1) \notin W(i)$

pour que $W(i)$ et $W(i+1)$ aient la même forme.

Supposons que $f(i+1) \notin W(i)$ (voir Figure 6.11).
Dans ce cas, $W(i)$ ne contient aucun 'a' et la forme de $W(i)$ est $L = \alpha' a + \gamma'$ avec $0 \leq \gamma' < b$. L'intervalle $\{t_b(i)+b,\, t_b(i)+b+1, \ldots,\, t_b(i)+L+a\}$ ne peut donc contenir que des 'a'. Ainsi, comme on ne peut pas avoir de période d'inactivité de durée supérieure à a dans aucune fenêtre, $W(i+1)$ contient exactement α' 'a'.

On a donc $|F(i)| = |L(i+1)| = 0$ et $|L(i)| = |F(i+1)| = \alpha$, et $|F(i)| + |L(i)| = |F(i+1)| + |L(i+1)|$.

Supposons que $f(i+1) \in W(i)$ (voir Figure 6.12).

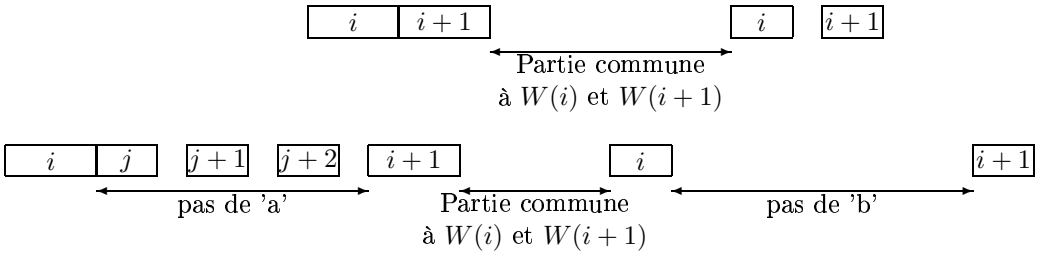


FIG. 6.12 – $f(i+1) \in W(i)$: $s(f(i)) = f(i+1)$ ou $s(f(i)) = l(j)$

– Supposons $s(f(i)) = f(i+1)$.
Dans ce cas, $s(l(i)) = l(i+1)$ donc $F(i+1) = F(i) \setminus \{i+1\}$ et $L(i+1) = L(i) \cup \{i\}$. Ainsi,
$$|F(i+1)| + |L(i+1)| = |F(i)| - 1 + |L(i)| + 1 = |F(i)| + |L(i)|$$
– Supposons $s(f(i)) = l(j)$, $s(l(j)) = l(j+1)$, …, $s(l(j+q)) = f(i+1)$.
Dans ce cas, on a un nombre $q+1 \geq 2$ de 'b' dans l'intervalle $\{t_a(i)+a,\, t_a(i)+a+1, \ldots,\, t_a(i+1)-1\}$. Il est donc de longueur $b + q\,a \leq l < 2\,b + q\,a$. Ainsi, l'intervalle $\{t_b(i)+b,\, t_b(i)+b+1, \ldots,\, tb(i+1)-1\}$ ne contient que des 'a' et est de longueur $l + a - b$. Or $(1+q)\,a \leq l + (a-b) < b + (1+q)\,a$ et $\gamma' < b$ donc l'intervalle $\{t_b(i)+b,\, t_b(i)+b+1, \ldots,\, tb(i+1)-1\}$ contient exactement $q+1$ 'a'. Ainsi, $|F(i+1)| = |F(i)| + q + 1$ et $|L(i+1)| = |L(i)| - (q+1)$. Donc
$$|F(i+1)| + |L(i+1)| = |F(i)| + |L(i)|$$

Pour toute tâche i de $\mathcal{C}$, on a $|F(i+1)| + |L(i+1)| = |F(i)| + |L(i)| = \alpha'$. Le fait que le temps mort total de $W(i+1)$ soit inférieur à a permet de conclure quand à la forme de $W(i+1)$. □

Lemme 22 *Soit $\mathcal{C}$ un cycle tel que $a-1$ est une borne supérieure du temps mort contenu dans la fenêtre de toute tâche de $\mathcal{C}$. Si $\mathcal{C}$ contient au moins une tâche de poids a, alors toutes les fenêtres des tâches de $\mathcal{C}$ ont la même forme.*

Démonstration. Sans perte de généralité, on peut considérer que la première tâche du cycle est telle que $s(f(1)) = f(2)$ et $w(2) = a$. On suppose que $W(1)$ est de la forme $L = \alpha' a + \beta'(a+b) + \gamma'$.

Comme la tâche 2 est de poids a, il n'y a pas de temps mort entre $f(1)$ et $f(2)$ et il y a une période d'inactivité de durée $a-b$ entre $l(1)$ et $l(2)$. Ainsi, le temps mort total $TM(W(2))$ inclus dans $W(2)$ est tel que $TM(W(2)) = a - b + TM(W(1))$. Or $TM(W(1)) \geq \gamma'$ donc $\gamma' < b$ puisque le temps mort total dans chaque fenêtre est inférieur à a.

On peut donc utiliser le Lemme 21 pour compléter la preuve. □

Lemme 23 *Soit $\mathcal{C}$ un cycle tel que $a-1$ est une borne supérieure du temps mort contenu dans la fenêtre de toute tâche de $\mathcal{C}$. On suppose que $\mathcal{C}$ contient au moins une tâche de poids a. Pour toute tâche j de $\mathcal{C}$, si $L = \alpha' a + \beta'(a+b) + \gamma'$ est la forme de $W(j)$, alors $S(j) = S(j + \alpha' + 2\beta')$ et la séquence S des opérations qui sépare $f(j)$ et $f(j + \alpha' + 2\beta')$ est $S = S(j)bS^{-1}(j)$.*

Démonstration. Remarquons tout d'abord que le Lemme 22 et sa démonstration impliquent que $\gamma' < b$ et que le nombre d'opérations dans chaque fenêtre est constant et égal à $\alpha' + 2\beta'$. Toute opération $f(i)$ est donc séparée de $l(i)$ par exactement $\alpha' + 2\beta'$ opérations.

On considère une tâche j du cycle $\mathcal{C}$. La preuve du Lemme 21, implique que
- Si $f(j+1) \notin W(j)$, alors $S(j) = b^{\alpha'}$ et $S(j+1) = a^{\alpha'}$ donc $S = S(j)baa^{\alpha'-1} = S(j)bS^{-1}(j)$.
- Si $f(j+1) \in W(j)$ et $f(j)$ est suivie de q 'b' puis de $f(j+1)$, alors $S(j)$ est de la forme $b^q a S_1$ et $S(j+1)$ est de la forme $S_1 b a^q$.

On en déduit que si la $k^{ième}$ opération du cycle est un 'b', l'opération $k + \alpha' + 2\beta'$ est un 'a', puisque chaque fenêtre contient exactement $\alpha' + 2\beta'$ opérations.

Comme, si la $k^{ième}$ opération du cycle est un 'a', l'opération $k + \alpha' + 2\beta'$ est un 'b', on peut conclure. □

On considère la forme r-admissible $L = \epsilon a + \beta(a+b) + \gamma$ avec $\epsilon \in \{0, 1\}$. Le cycle $\mathcal{C}(\epsilon, \beta)$ est tel que chaque fenêtre contient au plus $a-1$ temps morts. On a de plus au moins β 'a' et β 'b' dans chaque fenêtre, donc la durée d'inactivité moyenne d'un cycle ne peut pas dépacer a pour $\beta + 1$ tâche. On peut remarquer aussi que $2\beta + \epsilon$ est le nombre maximal d'opérations que peut contenir une fenêtre puisque $\lfloor L/a \rfloor \leq 2\beta + \epsilon$ et que $L - \lfloor L/a \rfloor < b$ lorsque $\lfloor L/a \rfloor = 2\beta + \epsilon$ (voir démonstration de la Proposition 22). Ce nombre d'opérations est celui inclus dans toute fenêtre du cycle $\mathcal{C}(\epsilon, \beta)$ et du cycle $\mathcal{C}(\lfloor L/a \rfloor, 0)$ si celui-ci le domine (Proposition 22).

Un cycle qui possède une tâche de poids a et qui ne conserve pas la même forme de fenêtre créera un temps mort de durée supérieure ou égale à a dans au moins une de ses fenêtres et l'empêchera donc de contenir $2\beta + \epsilon$ opérations.

On peut donc s'attendre à ce que de tels cycles soient dominés par ceux qui garderaient une fenêtre constante. Si cette fenêtre est de la forme $L = \alpha' a + \beta'(a+b) + \gamma$, on conjecture alors que le cycle est dominé par $\mathcal{C}(\alpha', \beta')$.

6.5 Conclusion

Dans ce chapitre, nous avons étudié le problème des tâches couplées. Nous montrons son équivalence avec un problème de cellules robotisées sans attente à une machine, puis nous nous intéressons au problème à un unique type de tâches et établissons la conjecture suivante.

Conjecture. *Si L s'écrit sous les formes admissibles $L = \alpha a + \delta$, $0 \leq \delta < a$ et $L = \epsilon a + \beta(a+b) + \gamma$ avec $\epsilon \in \{0, 1\}$ et $0 \leq \gamma < a$ si $\epsilon = 0$ et $0 \leq \gamma < b$ si $\epsilon = 1$, alors l'un des deux cycles $\mathcal{C}(\alpha, 0)$ et $\mathcal{C}(\epsilon, \beta)$ est optimal.*

Cette conjecture a été testée pour un nombre important d'instances et n'a pas été invalidée. Si elle s'avère exacte, maximiser le taux de production dans le cas de pièces identiques se ferait en temps polynomial. On montre dans ce chapitre que le meilleur de ces deux cycles dominent l'ensemble des cycles associés aux formes admissibles de L et sont optimaux pour certaines classes d'instances particulières. Nous espérons pouvoir prochainement conclure quand à la validité de cette conjecture.

Conclusion

Dans ce manuscrit, nous abordons différents problèmes, théoriques et pratiques, d'ordonnancement dans les ateliers de production. Dans une première partie, nous considérons différents modèles de cellules de production automatisées, jusqu'ici étudiés indépendamment dans la littérature classique. Nous insistons sur les liens et distinctions entre ces différents modèles et les hypothèses qui rendent certains d'entre eux équivalents. Nous démontrons l'appartenance à $\mathcal{P}$ de plusieurs problèmes de flowshops robotisés et flowshops avec serveur à deux machines, avant de nous intéresser à un problème industriel de planification d'expérience, pour lequel un logiciel est implémenté. L'étude de ses différents aspects nous conduit à introduire une nouvelle notion en ordonnancement avec ressources d'entrée/sortie : l'indisponibilité des opérateurs. Dans un environnement mono-machine, nous étudions la complexité de la minimisation du makespan en présence de périodes d'indisponibilité des opérateurs. Ce problème est d'ores et déjà $\mathcal{NP}$-Difficile pour une période d'indisponibilité dans le cas général, mais admet un algorithme polynomial, que nous explicitons, si la période est plus courte que la durée d'usinage de n'importe quelle pièce. Pour les autres cas, nous proposons des heuristiques polynomiales efficaces dont nous établissons les performances.

Dans une seconde partie, nous nous tournons vers des problèmes *high multiplicity*, pour lesquels les tâches ne possèdent plus chacune leurs spécificités, mais sont agrégées en catégories, les tâches d'un même type étant toutes identiques. Le nombre total de tâches n'étant plus polynomial de la taille de l'instance, il est difficile de déterminer à quelles classes de complexité appartiennent ces problèmes. Nous en étudions deux exemples. Le premier concerne l'ordonnancement juste-à-temps des lignes de production flexibles, pour lequel de nombreuse fonctions objectif ont été introduites dans la littérature. Nous comparons les plus étudiées et nous focalisons sur les problèmes multicritères faisant intervenir plusieurs d'entre elles. Nous montrons que leur optimisation simultanée n'est pas toujours possible et examinons plusieurs propriétés d'instances pour lesquelles l'optimisation de plusieurs critères peut être faite par la même solution. Le second problème abordé est l'ordonnancement de tâches couplées identiques avec comme critère le makespan ou le taux de production maximal. Nous montrons qu'il est équivalent à un problème classique de cellules robotisées et proposons une conjecture pour le cas high multiplicity des tâches couplées identiques. Si elle s'avère exacte, le problème pourra se résoudre en temps polynomial dans le cas cyclique, et ce malgré la taille très réduite de l'instance.

Cette thèse ouvre de nombreuses perspectives de recherches futures. Dans le Chapitre 2, nous étudions des problèmes de flowshop robotisés à deux machines. Les complexités de leurs versions à trois ou quatre machines sont en l'état de nos connaissances encore ouvertes et pourront donc être examinées. Les problèmes d'ordonnancement avec périodes

d'indisponibilité des opérateurs du Chapitre 4 n'ont été étudiés que pour le critère $Cmax$. Le problème industriel de planification dont il est dérivé faisait apparaître un autre critère, la somme des dates d'achèvement des tâches, qui fera l'objet d'une thèse qui commencera l'an prochain. L'analyse de la structure des solutions du problème d'ordonnancement juste-à-temps du Chapitre 5 laisse de nombreuses questions en suspend qui sont décrites dans la conclusion de ce chapitre, et bien entendu, les heuristiques polynomiales proposées pour ce problèmes mériteraient d'être améliorées et leurs performances établies. Le Chapitre 6 laisse en suspend sa conjecture des tâches couplées identiques qui à mon grand regret n'a pas pu être démontrée. J'espère que les propriétés introduites dans ce chapitre me permettrons de conclure quand à son exactitude.

Bibliographie

[1] A.H. Abdekhodaee and A. Wirth. Scheduling parallel machines with a single server : some solvable cases and heuristics. *Computers and Operations Research*, 29 :295–315, 2002.

[2] A.H. Abdekhodaee, A. Wirth, and H.S. Gan. Equal processing and equal setup time cases of scheduling parallel machines with a single server. *Computers and Operations Research*, 31 :1867–1889, 2004.

[3] A.H. Abdekhodaee, A. Wirth, and H.S. Gan. Scheduling two parallel machines with a single server : the general case. *Computers and Operations Research*, 33 :994–1009, 2006.

[4] A. Agnetis. Scheduling no-wait robotic cells with two and three machines. *European Journal of Operational Research*, 123(2) :303–314, 2000.

[5] A. Agnetis and D. Pacciarelli. Part sequencing in three-machine no-wait robotic cells. *Operations Research Letters*, 27 :185–192, 2000.

[6] D. Ahr, J. Békési, G. Galambos, M. Oswald, and G. Reinelt. An exact algorithm for scheduling identical coupled tasks. *Mathematical Methods of Operations Research (ZOR)*, 59 :193–203, 2004.

[7] C.R. Asfahl. *Robots and manufacturing automation.* John Wiley & Sons, New York, 1985.

[8] A. Asratian, T.M.J. Denley, and R. Hăggkvist. *Bipartite graphs and their applications.* University Press, Cambridge, 1998.

[9] G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, and M. Protasi. *Complexity and Approximation, Combinatorial optimization problems and their approximability properties.* Springer Verlag, 1999.

[10] B. Balinski. Le scrutin. *Pour La Science*, 294 :46–69, 2002.

[11] M. Balinski and N. Shahidi. A simple approach to the product rate variation problem via axiomatics. *Operations Research Letters*, 22 :129–135, 1998.

[12] J. Bautista, R. Companys, and A. Corominas. A note on the relation between the product rate variation (prv) problem and the apportionment problem. *Journal of the Operational Research Society*, 47 :1410–1414, 1996.

[13] J.C. Billaut, V.A. Strusevich, P.H. Maugiere, and J.L. Bouquard. A single operator two-machine flow shop scheduling problem with makespan minimization. In *Models and Algorithms for Planning and Scheduling Problems(MAPSP)*, Siena, Italy, 2005.

[14] J. Blazewicz, K. Ecker, T. Kis, and M. Tanas. A note on the complexity of scheduling coupled tasks on a single processor. *Journal of the Brazilian Computer Society*, 7(3) :23–26, 2001.

[15] J. Blazewicz, K.H. Ecker, E. Pesch, G. Schmidt, and J. Weglarz. *Scheduling computer and manufacturing processes.* Springer-Verlag, Berlin, Heidelberg, 1996.

[16] M. Boudhar and G. Finke. Scheduling on a batch machine with job compatibilities. *Belgian Journal of Operations Research, Statistics and Computer Science - JORBEL*, 40(1-2) :69–80, 2000.

[17] N. Brauner. Identical part production in cyclic robotic cells : a state of the art. *Les cahiers du laboratoire Leibniz*, 144, 2006.

[18] N. Brauner and Y. Crama. The maximum deviation just-in-time scheduling problem. *Discrete Applied Mathematics*, 134 :25–50, 2004.

[19] N. Brauner, Y. Crama, A. Grigoriev, and J. van de Klundert. A framework for the complexity of high-multiplicity scheduling problems. *Journal of Combinatorial Optimization*, 9 :313–323, 2005.

[20] N. Brauner, R. Echahed, G. Finke, H. Gregor, and F. Prost. A complete assignment algorithm and its application in constraint declative languages. *Les cahiers du laboratoire Leibniz*, 111, 2004.

[21] N. Brauner, G. Finke, and C. Gueguen. Flow-shop robotisé monoproduit avec stockage. *European Journal of Automation / Journal Européen des Systèmes Automatisés - APII-JESA*, 32 :875–891, 1998.

[22] N. Brauner, G. Finke, and W. Kubiak. Complexity of one-cycle robotic flow-shops. *Journal of Scheduling*, 6(4) :355–371, 2003.

[23] N. Brauner, G. Finke, V. Lebacque, C.N. Potts, and J. Whitehead. Scheduling of coupled tasks and one-machine no-wait robotic cells. In *Information Control Pproblems in Manufacturing 2006, A Proceedings volume from the 12th IFAC International Symposium*, volume 3, pages 141–146, Saint-Etienne, France, 2006.

[24] N. Brauner and G. Naves. Scheduling chains of operations on a batching machine with task compatibility. Communication privée, 2007.

[25] J. Breit, G. Schmidt, and V. A. Strusevich. Non-preemptive two-machine open shop scheduling with non-availability constraints. *Mathematical Methods of Operations Research*, 57 :217–234, 2003.

[26] P. Brucker, C. Dhaenens-Flipo, S. Knust, S.A. Kravchenko, and F. Werner. Complexity results for parallel machine problems with a single server. *Journal of Scheduling*, 5 :429–457, 2002.

[27] P. Brucker, A. Gladky, H. Hoogeveen, M.Y. Kovalyov, C.N. Potts, T. Tautenhahn, and S. Van De Velde. Scheduling a batching machine. *Journal of Scheduling*, 1 :31–54, 1998.

[28] P. Brucker, T. Hilbig, and J. Hurink. A branch and bound algorithm for a single-machine scheduling problem with positive and negative time-lags. *Discrete Applied Mathematics*, 94 :77–99, 1999.

[29] P. Brucker and S. Knust. Complexity results for single-machine problems with positive finish-start time-lags. *Computing*, 63 :299–316, 1999.

[30] P. Brucker, S. Knust, and G. Wang. Complexity results for flow-shop problems with a single server. *European Journal of Operational Research*, 165 :398–407, 2005.

[31] J. Bruno, E.G. Coffman Jr., and R. Sethi. Scheduling independant tasks to reduce mean finishing time. *ACM*, 17 :382–387, 1974.

[32] T.C.E Cheng, C. Sriskandarajah, and G. Wang. Two- and three-stage flowshop scheduling with no-wait in process. *Production and Operations Management*, 9(4) :367–378, 2000.

[33] T.C.E. Cheng, G. Wang, and C. Sriskandarajah. One-operator-two-machine flowshop scheduling with setup and dismounting times. *Computers and Operations Research*, 26 :715–730, 1999.

[34] S.A. Cook. The complexity of theorem proving procedures. In *Proceedings of the third annual ACM symposium on Theory of computing*, pages 151–158, 1971.

[35] A. Corominas and N. Moreno. On the relations between optimal solutions for different types of min-sum balanced jit optimization problems. *INFOR*, 41 :333–339, 2003.

[36] Y. Crama, V. Kats, J. van de Klundert, and E. Levner. Cyclic scheduling in robotic flowshops. *Annals of Operations Research : Mathematics of Industrial Systems*, 96 :97–124, 2000.

[37] Y. Crama and J. van de Klundert. Cyclic scheduling in 3-machine robotic flow shops. *Journal of Scheduling*, 2(1) :35–54, 1999.

[38] M.W. Dawande, H.N. Geismar, S.P. Sethi, and C. Sriskandarajah. Sequencing and scheduling in robotic cells : Recent developments. *Journal of Scheduling*, 8(5) :387–426, 2005.

[39] M.W. Dawande, H.N. Geismar, S.P. Sethi, and C. Sriskandarajah. *Throughput Optimization in Robotic Cells*, volume 101 of *International Series in Operations Research and Management Science*. Springer, 2007.

[40] G. Finke, C. Gueguen, and N. Brauner. Robotic cells with buffer space. Dublin, Ireland, 1996. Proceedings Conference of the European Chapter on Combinatorial Optimization (ECCO IX).

[41] R.L. Graham. Bounds on multiprocessing timing anomalies. *SIAM Journal of Applied Mathematics*, 17(2) :416–429, 1969.

[42] R.L. Graham, E.L. Lawler, J.K. Lenstra, and H.G. Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling : A survey. *Annals of Discrete Mathematics*, 5 :287–326, 1979.

[43] A. Grigoriev. *High multiplicity scheduling problems*. PhD thesis, Univerteit Maastricht, Maastricht, Netherlands, 2003.

[44] J. N. D. Gupta. Comparative evaluation of heuristic algorithms for the single machine scheduling problem with two operations per job and time-lags. *Journal of Global Optimization*, 9 :239–250, 1996.

[45] N. G. Hall, H. Kamoun, and C. Sriskandarajah. Scheduling in robotic cells : classification, two and three machine cells. *Operations Research*, 45 :421–439, 1997.

[46] N.G. Hall, C. Potts, and C. Sriskandarajah. Parallel machine scheduling with a common server. *Discrete Applied Mathematics*, 102 :223–243, 2000.

[47] N.G. Hall, C.N. Potts, and C. Sriskandarajah. Parallel machine scheduling with a common server. In *Fifth International Workshop on Project Management and Scheduling*, pages 102–106, 1996.

[48] C. Hanen and A. Munier. Ordonnancement cyclique d'un robot sur une ligne de galvanoplastie : modèles et algorithmes. Technical report, Institut Blaise Pascal, 1993.

[49] J. Hurink and J. Keuchel. Local search algorithms for a single machine scheduling problem with positive and negative time-lags. *Discrete Applied Mathematics*, 112 :179–197, 2001.

[50] J. Hurink and S. Knust. Makespan minimization for flow-shop problems with transportation times and a single robot. *Discrete Applied Mathematics*, 112 :199–216, 2001.

[51] R.R. Inman and R.L. Bulfin. Sequencing JIT mixed-model assembly lines. *Management Science*, 37 :901–904, 1991.

[52] V. Jost. *Deux problèmes d'approximation diophantienne : Le partage proportionnel en nombre entiers et Les pavages équilibrés de* $\mathbb{Z}$. Mémoire de DEA en Recherche Operationnelle, INPG, Grenoble, France, 2002.

[53] M. Karp. A characterization of the minimum cycle mean in a digraph. *Discrete Mathematics*, 23 :309–311, 1978.

[54] H. Kise. On an automated two-machine flowshop scheduling problem with infinite buffer. *Journal of the Operations Research Society of Japan*, 34 :354–361, 1991.

[55] H. Kise, H. Matsuo, and R.S. Sullivan. Optimal cyclic scheduling for automated two-machine flowshops with no buffer. *Symposium on Flexible Automation*, pages 1179–1183, 1990.

[56] M. Y. Kovalyov, W. Kubiak, and J. S. Yeomans. A computational analysis of balanced JIT optimization algorithms. *Journal of Information Systems and Operational Research - INFOR*, 39 :299–315, 2001.

[57] S.A. Kravchenko and F. Werner. Parallel machine scheduling problems with a single server. *Mathematical Computer Modelling*, 26(12) :1–11, 1997.

[58] W. Kubiak. Minimizing variation of production rates in just-in-time systems : a survey. *European Journal of Operational Research*, 66 :259–271, 1993.

[59] W. Kubiak and S. Sethi. Optimal just-in-time schedules for flexible transfer lines. *The International Journal of Flexible Manufacturing Systems*, 6 :137–154, 1994.

[60] V. Lebacque and N. Brauner. Flowshops with material handling : a critical comparison of different existing models. Soumis à *Computers and Operations Research*, 2007.

[61] V. Lebacque, V. Jost, and N. Brauner. Simultaneous optimization of classical objectives in JIT scheduling. *European Journal of Operational Research*, 182 :29–39, 2007.

[62] C.-Y. Lee. Machine scheduling with an availability constraint. *Journal of Global Optimization*, 9 :395–416, 1996.

[63] L. Lei and Q. Liu. An $o(n^2)$ algorithm for cyclic scheduling of a two-machine no-wait robotic flowshop with multiple products. *Private communication*, 2004.

[64] J.K. Lenstra, A.H.G Rinnoy Kan, and P. Brucker. Complexity of machine scheduling problems. *Annals of Discrete Mathematics*, 1 :343–362, 1977.

[65] J.Y-T. Leung. *Handbook of scheduling.* Chaman and Hall/CRC, 2004.

[66] J.Y-T. Leung, H. Li, and M. Pinedo. Order scheduling models : an overview. In *First Multidisciplinary International Conference on Scheduling : Theory and Applications (MISTA 2003)*, pages 37–53, Nottingham, UK, 2005.

[67] L. Lovasz and M.D. Plummer. *Matching theory.* Akademiai Kiado, Budapest, 1986.

[68] M.Garey and D. Johnson. *Computers and intractability : A guide to the theory of NP-completeness.* Freeman and Company, New York, 1979.

[69] J. Miltenburg. Level schedules for mixed-model assembly lines in just-in-time production systems. *Management Science*, 35 :192–207, 1989.

[70] Y. Monden. *Toyota production system.* Institute of Industrial Engineers Press, 1983.

[71] A. J. Orman and C. N. Potts. On the complexity of coupled-task scheduling. *Discrete Applied Mathematics*, 72 :141–154, 1997.

[72] A. J. Orman, C. N. Potts, A. K. Shahani, and A. R. Moore. Scheduling for a multifunction phased array radar system. *European Journal of Operational Research*, 90 :13–25, 1996.

[73] M. Pinedo. *Scheduling : Theory, Algorithms, and Systems.* Prentice Hall, Englewood Cliffs, NJ, 1995.

[74] C.N. Potts and M.Y. Kovalyov. Scheduling with batching : A review. *EJOR*, 120 :228–249, 2000.

[75] C.N. Potts and J.D. Whitehead. Heuristics for coupled-operation scheduling problem. *Journal of the Operational Research Society*, pages 1–14, 2006.

[76] H. Röck. The three-machine no-wait flow shop is np-complete. *Journal of the ACM*, 31(2) :336–345, 1984.

[77] B. Roy. Contribution de la théorie des graphes à l'étude de certains problèmes linéaires. *Extrait des comptes rendus de l'Académie des Sciences*, 248 :2437–2439, 1959.

[78] E. Sanlaville and G. Schmidt. Machine scheduling with availability constraints. *Acta Informatica*, 35 :795–811, 1998.

[79] G. Schmidt. Scheduling with limited machine availability. *European Journal of Operational Research*, 121 :1–15, 2000.

[80] S.P. Sethi, C. Sriskandarajah, G. Sorger, J. Blazewicz, and W. Kubiak. Sequencing of parts and robot moves in a robotic cell. *International Journal of Flexible Manufacturing Systems*, 4 :331–358, 1992.

[81] R. D. Shapiro. Scheduling coupled tasks. *Naval Research Logistics Quarterly*, 27(2) :489–497, 1980.

[82] G. Steiner and J. S. Yeomans. Level schedules for mixed-model, just-in-time processes. *Management Science*, 39 :728–735, 1993.

[83] G. Steiner and J. S. Yeomans. A bicriterion objective for levelling the schedule of a mixed-model, just-in-time assembly processes. *Mathematical and Computer Modelling*, 20 :123–134, 1994.

[84] E.D. Wikum, D.C. Llewellyn, and G.L. Nemhauser. One-machine generalized precedence constrained scheduling problems. *Operations Research Letters*, 16(2) :87–99, 1994.

Annexes

Annexe A

Compléments pour l'ordonnancement Juste-À-Temps

Cette annexe contient plusieurs résultats théoriques dont les preuves laborieuses n'ont pas été incluses dans le Chapitre 5 dans un soucis de clarté (Section A.1.1). Ces résultats s'avèrent cependant utiles pour représenter les exemples de ce chapitre (dont la plupart se trouvent dans cette annexe, Section A.4) et ont permis de proposer une heuristique polynomiale (Section A.1.2). L'étude de ses performances n'ayant pas pu être réalisée, elle se trouve elle aussi dans cette annexe. Le détail des formulations en tant que programme linéaire en nombre entiers des différents problèmes évoqués dans le Chapitre 5 est aussi inclus dans cette partie (Section A.2.1).

A.1 Symétrie du problème et heuristique polynomiale

Dans cette section, nous exploitons les formulations décrites dans la Section 5.1.2 p 94 pour proposer des méthodes qui créent en temps polynomial des solutions pour le problème d'ordonnancement juste-à-temps.

A.1.1 Structure des graphes et représentation

Les graphes qui correspondent aux formulations du problème juste-à-temps en tant que problème de couplage parfait ou d'affectation de poids minimal présentent des propriétés de régularités intéressantes. Elles permettent de simplifier la représentation de ces graphes pour rendre les exemples proposés à la Section A.4 plus lisibles et posent les fondements de l'heuristique polynomiale proposée à la Section A.1.2.

A.1.1.1 Représentation par les f-facteurs

Lorsque deux types de pièces i_1 et i_2 sont de même demande d, les sommets (i_1, j) et (i_2, j) ($j \in \{1, 2, \ldots, d\}$) ont les mêmes voisins. Afin de simplifier le graphe, nous proposons dans ce cas, d'agréger ces sommets. Dans le graphe transformé, comme nous le verrons dans

cette section, nous ne chercherons plus un couplage, mais un *f-facteur* [67, 8] (*f-factor* en anglais). Considérons un graphe $G = (V, E)$ et une fonction $f : V \longrightarrow \mathbb{N}$.

Définition 25 (f-facteur) *Un* f-facteur *est un sous-graphe* H *de* G *tel que* $deg_H(v) = f(v), \forall v \in V$.

On déduit directement de cette définition la proposition suivante :

Proposition 24 *Un graphe a un* 1*-facteur si et seulement si il a un couplage parfait.*

On considère une instance $(n, d_1, d_2, \ldots, d_n, B)$ du problème de décision max-abs et le graphe $G = (V_1 \cup V_2, E)$ associé. Soit D la somme de toutes les demandes. Supposons que $d_{i_1} = d_{i_2}$ pour deux types de pièces distincts i_1 et i_2 et considérons un indice $j \in \{1, 2, \ldots, d_{i_1}\}$. Par définition (5.4) p 97 des instants de production au plus tôt et au plus tard, on a $E(i_1, j) = E(i_2, j)$ et $L(i_1, j) = L(i_2, j)$. Ainsi, dans le graphe G on aura $\delta((i_1, j)) = \delta((i_2, j))$ (avec $\delta(v)$ l'ensemble des arêtes incidentes au sommet v). D'où la remarque suivante :

Lemme 24 *Soit* G *le graphe associé à l'instance* $(n, d_1, d_2, \ldots, d_n, B)$*. Si deux types de pièces* i_1 *et* i_2 *ont même demande* $d = d_{i_1} = d_{i_2}$ *alors pour tout* $j \in \{1, 2, \ldots, d\}$*, les sommets* (i_1, j) *et* (i_2, j) *ont les mêmes voisins.*

Nous agrégerons alors les sommets (i_1, j) et (i_2, j), si les types i_1 et i_2 sont de même demande, de la façon décrite par l'Algorithme 3. Les Figures A.1 et A.2 illustrent sur un exemple la transformation du graphe opérée.

Algorithme 3 Simplification

Initialisation
$L = \{1, 2, \ldots, n\}$
$G = (V_1 \cup V_2, E)$
$f : V = V_1 \cup V_2 \longrightarrow \mathbb{N}$, $f(v) = 1 \quad \forall v \in V$
Tant que il existe $i_1, i_2 \in L$ tels que $d_{i_1} = d_{i_2}$ **Faire**
  Pour tout $j \in \{1, 2, \ldots, d_{i_1}\}$ **Faire**
    $V_2 = V_2 \setminus (i_2, j)$
    $f((i_1, j)) = f((i_1, j)) + f((i_2, j))$
    $E = E \setminus \{((i_2, j), k) \in \delta((i_2, j))\}$
  Fin Pour
  $L = L \setminus \{i_2\}$
Fin Tant que

La preuve de la proposition suivante est facile mais laborieuse et n'est donnée ici que pour être complet.

Proposition 25 *Le graphe* G *a un couplage parfait si et seulement si le graphe* G' *obtenu par l'algorithme Simplification a un* f*-facteur.*

Démonstration. On note $G^p = (V_1 \cup V_2^p, E^p)$ le graphe et f^p la fonction au début de la $p^{ième}$ itération.

On remarque tout d'abord que l'algorithme ne modifie pas l'ensemble V_1 ni les valeurs de f sur cet ensemble, c'est-à-dire que

$$\forall p, \quad V_1^p = V_1 \text{ et } (\forall k \in V_1, \quad f^p(k) = f(k) = 1) \tag{A.1}$$

De plus, on a $\forall p,\ V_2^{p+1} \subseteq V_2^p$.

Le graphe G admet un couplage parfait si et seulement si il a un 1-facteur. On raisonne donc par récurrence pour montrer que pour tout $p \geq 1$, G' a un f-facteur si et seulement si G a un couplage parfait.

Implication. Supposons que ce graphe G^p admet un f^p-facteur H^p et que L^p contient les deux indices i_1 et i_2 (tels que $d_{i_1} = d_{i_2}$) qui sont utilisés à l'itération p de l'algorithme pour construire le graphe G^{p+1} et la fonction f^{p+1}.

Montrons que G^{p+1} admet un f^{p+1}-facteur.
Considérons le sous-graphe H^{p+1} de G^{p+1} tel que $V(H^{p+1}) = V(G^{p+1})$ et $((i,j),k)$ est une arête de H^{p+1} si et seulement si une des deux assertions suivantes est vérifiée :
– $(i,j) \in V_2^{p+1}$ et $((i,j),k)$ est une arête de H^p ;
– $i = i_1$ et $((i_2,j),k)$ est une arête de H^p.
Toutes les arêtes de H^{p+1} sont donc bien incluses dans E^{p+1} puisque qu'aucune arête de la forme $((i_2,j),k)$ n'appartient à $E(H^{p+1})$. On peut remarquer que H^p et H^{p+1} ont le même nombre d'arêtes.

– Soit $k \in V_1$. D'après l'équation (A.1), on a $f^{p+1}(k) = f^p(k)$. Or $deg_{H^p}(k) = f^p(k)$ car H^p est un f^p-facteur et $deg_{H^{p+1}}(k) = deg_{H^p}(k)$ d'après la définition des arêtes de H^{p+1}. On peut donc en déduire que

$$deg_{H^{p+1}}(k) = f^{p+1}(k)$$

– Soit $(i,j) \in V_2^{p+1}$ tel que $i \neq i_1$. D'après la définition des arêtes de H^{p+1}, $deg_{H^{p+1}}((i,j)) = deg_{H^p}((i,j))$. H^p étant un f^p-facteur, $deg_{H^p}((i,j)) = f^p((i,j))$ et donc $deg_{H^{p+1}}((i,j)) = f^p((i,j))$.
f^p et f^{p+1} diffèrent sur V_2^{p+1} seulement pour les couples (i_1,l), $l \in \{1, 2, \ldots, d_{i_1}\}$. Ainsi, $f^{p+1}((i,j)) = f^p((i,j))$. On a donc

$$deg_{H^{p+1}}((i,j)) = f^{p+1}((i,j))$$

– Soit $j \in \{1, 2, \ldots, d_{i_1}\}$. Comme H^p est un f^p-facteur, $deg_{H^p}((i_1,j)) = f^p((i_1,j))$ et $deg_{H^p}((i_2,j)) = f^p((i_2,j))$. D'après la définition de H^{p+1}, $deg_{H^{p+1}}((i_1,j)) = deg_{H^p}((i_1,j)) + deg_{H^p}((i_2,j))$. Ainsi, $deg_{H^{p+1}}((i_1,j)) = f^p((i_1,j)) + f^p((i_2,j))$. Comme $f^{p+1}((i_1,j)) = f^p((i_1,j)) + f^p((i_2,j))$ nous avons

$$deg_{H^{p+1}}((i_1,j)) = f^{p+1}((i_1,j))$$

Ainsi, pour tout sommet v de H^{p+1} on a $deg_{H^{p+1}}(v) = f^{p+1}(v)$. On en déduit donc que H^{p+1} est un f^{p+1}-facteur de G^{p+1}.

Réciproque. Supposons que L^p contient les deux indices i_1 et i_2 (tels que $d_{i_1} = d_{i_2}$) qui sont utilisés à l'itération p de l'algorithme pour construire le graphe G^{p+1} et la fonction f^{p+1}. Supposons que G^{p+1} admet un f^{p+1}-facteur H^{p+1}.

Notons F^{p+1} l'ensemble des arêtes de H^{p+1}. Pour tout $j = 1, 2, \ldots, d_{i_1}$ on considère une partition de l'ensemble $\{((i_1, j), k) \in F^{p+1}\}$ en 2 sous-ensembles $F^1_{j,k}$ et $F^2_{j,k}$ de tailles respectives $f^p((i_1, j))$ et $f^p((i_2, j))$. On a

$$|\left\{((i_1, j), k) \in F^{p+1}\right\}| = f^{p+1}((i_1, j))$$

puisque $deg_{H^{p+1}}((i_1, j)) = f^{p+1}((i_1, j))$ et ainsi,

$$|\left\{((i_1, j), k) \in F^{p+1}\right\}| = f^p((i_1, j)) + f^p((i_2, j))$$

ce qui implique que la partition est réalisable.
On définit le sous-graphe H^p de G^p suivant. Tous les sommets de G^p sont dans H^p et une arête $((i, j), k)$ de G^p est dans l'ensemble des arêtes du sous-graphe H^p si et seulement si
– $i \notin \{i_1, i_2\}$ et $((i, j), k) \in E^{p+1}$;
– $i = i_1$ et $((i, j), k) \in F^1_{j,k}$;
– $i = i_2$ et $((i, j), k) \in F^2_{j,k}$.
De la même façon que précédemment, on peut montrer que pour tout sommet $k \in V^p_1$, on a $deg_{H^p}(k) = f^p(k)$ et pour tout sommet (i, j) tel que $i \notin \{i_1, i_2\}$, on a $deg_{H^p}((i, j)) = f^p((i, j))$.

Par définition du sous-graphe H^p, pour tout $j \in \{1, 2, \ldots, d_{i_1}\}$, on a $deg_{H^p}((i_1, j)) = f^p((i_1, j))$ et $deg_{H^p}((i_2, j)) = f^p((i_2, j))$.

Ainsi, H^p est un f^p-facteur de G^p.

Donc G^p admet un f^p-facteur si et seulement si G^{p+1} a un f^{p+1}-facteur, ce qui permet de conclure. □

Remarque. On peut déduire de la démonstration de la Proposition 25 comment obtenir un f-facteur du graphe une fois transformé à partir d'un couplage du graphe initial et réciproquement.

Exemple 1. On considère l'instance $d = (1, 1, 4, 4)$ avec $d_1 = d_2 = 1$ et $d_3 = d_4 = 4$. La Figure A.1 représente le graphe correspondant pour la borne $B = \frac{7}{10}$. Les arêtes en gras forment un couplage parfait du graphe.

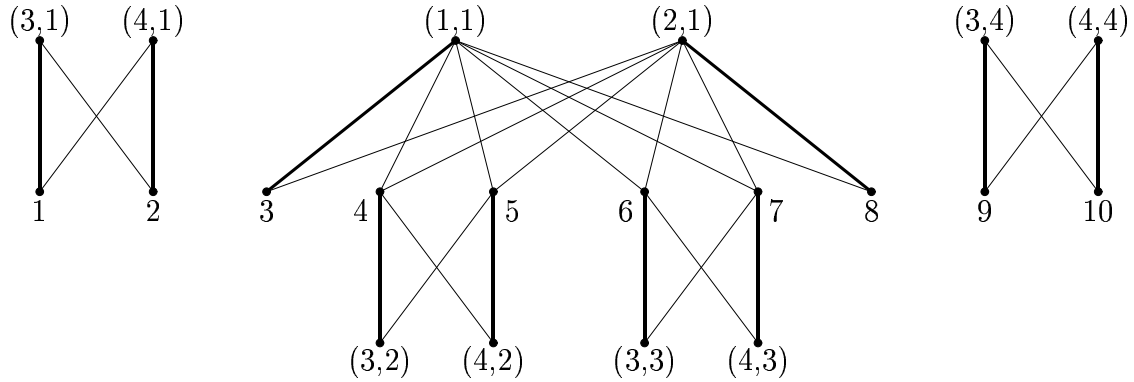


FIG. A.1 – Graphe biparti pour $d_1 = d_2 = 1$, $d_3 = d_4 = 4$ et $B = \frac{7}{10}$

Quand nous regroupons les types de pièces de demandes identiques, nous obtenons le graphe de la Figure A.2.

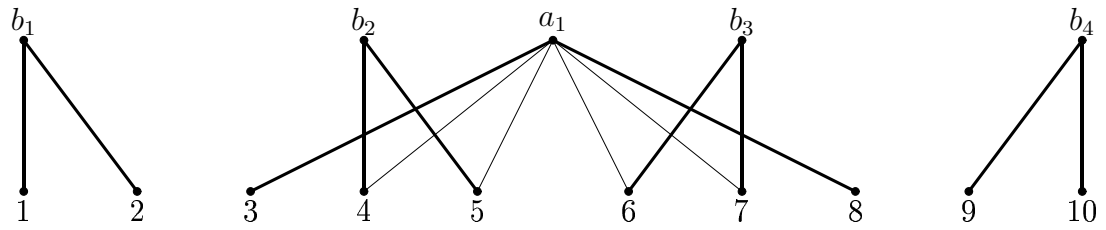


FIG. A.2 – Graphe biparti simplifié pour $d_1 = d_2 = 1$, $d_4 = d_4 = 4$ et $B = \frac{7}{10}$

Remarque. Les arêtes en gras de la Figure A.2 sont un f-facteur obtenu en appliquant l'algorithme Simplification au sous-graphe de la Figure A.1 induit par son couplage parfait.

A.1.1.2 Symétrie du graphe

Nous ajoutons une seconde transformation à celle fournie par l'Algorithme Simplification : si la solution est symétrique, alors seule la première moitié du graphe est représentée. Les Propositions 26 et 27 impliquent que le graphe est toujours symétrique par rapport au temps (mais ce n'est pas le cas selon la définition classique des graphes symétriques) et certains des exemples proposés ont des couplages présentant la même symétrie.

Proposition 26 *On considère un problème $B-$borné. Alors pour tout $(i,j) \in V_2$ on a*

$$E(i, d_i - j + 1) = D - L(i,j) + 1 \quad (A.2)$$
$$L(i, d_i - j + 1) = D - E(i,j) + 1 \quad (A.3)$$

Démonstration. À partir de l'équation (5.4) p 97 on obtient

$$\begin{aligned} E(i, d_i - j + 1) &= \left\lceil \frac{d_i - j + 1 - B}{r_i} \right\rceil \\ &= \left\lceil D - \frac{j - 1 + B}{r_i} \right\rceil \\ &= D - \left\lfloor \frac{j - 1 + B}{r_i} \right\rfloor \\ &= D - \left\lfloor \frac{j - 1 + B}{r_i} + 1 \right\rfloor + 1 \\ &= D - L(i,j) + 1 \end{aligned}$$

L'égalité (A.3) est obtenue en remplaçant j par $d_i - j + 1$ dans (A.2). □

Les arêtes du graphe sont donc symétriques et nous prouvons maintenant que leurs coûts le sont aussi. Considérons les deux lemmes suivants.

Lemme 25 *Soit F_i une fonction paire et convexe vérifiant la condition (5.2) p 94. L'instant idéal de production $Z^*_{i,j}$ de la pièce (i,j) est tel que*

$$Z^*_{i,j} = \begin{cases} D - Z^*_{i,d_i-j+1} & \text{si } k_{i,j} \text{ est entier} \\ D - Z^*_{i,d_i-j+1} + 1 & \text{sinon} \end{cases}$$

Démonstration. Pour toute fonction paire et convexe F_i vérifiant la condition (5.2) p 94, on a $F_i(x) = F_i(x-1)$ si et seulement si $x = \frac{1}{2}$. Ainsi, d'après la définition (5.5) p 98 de l'instant idéal de production $k_{i,j}$ de la pièce (i,j), nous avons $k_{i,j} = \frac{2j-1}{2r_i}$.

On considère un triplet (i,j,k). Soient $q \in \mathbb{N}$ et $\alpha \in \mathbb{Q}$ tels que $k_{i,j} = q + \alpha$ et $\alpha \in [0,1[$. On a

$$\begin{aligned} Z^*_{i,d_i-j+1} &= \lceil k_{i,d_i-j+1} \rceil \\ &= \left\lceil \frac{2(d_i - j + 1) - 1}{2r_i} \right\rceil \\ &= \left\lceil D - \frac{2j-1}{2r_i} \right\rceil \\ &= \lceil D - q - \alpha \rceil \\ &= D - q \end{aligned}$$

Ainsi, si $k_{i,j}$ est entier, c'est-à-dire si $Z^*_{i,j} = k_{i,j} = q$, on a $Z^*_{i,d_i-j+1} = D - Z^*_{i,j}$ et si $k_{i,j}$ n'est pas entier, c'est-à-dire si $Z^*_{i,j} = q+1$, on a $Z^*_{i,d_i-j+1} = D - Z^*_{i,j} + 1$. D'où le résultat. □

Lemme 26 *Soit F_i une fonction paire et convexe vérifiant la condition (5.2) p 94. Pour tout entier p de $\{1, 2, \ldots, D-1\}$ on a*

$$\psi_{i,j,p} = \psi_{i,d_i-j+1,D-p}$$

Démonstration. Soit m un entier. D'après (5.7) p 98, on a

$$\begin{aligned} \psi_{i,d_i-j+1,D-p} &= |F_i(d_i - j + 1 - (D-p)r_i) \\ &\quad -F_i(d_i - j + 1 - (D-p)r_i - 1)| \\ &= |F_i(-(-d_i + j - 1 + d_i - pr_i)) \\ &\quad -F_i(-(-d_i + j + d_i - pr_i))| \\ &= |F_i(-(j - 1 - pr_i)) - F_i(-(j - pr_i))| \end{aligned}$$

Comme F_i est paire, on a $\psi_{i,d_i-j+1,D-p} = |F_i(j-1-pr_i) - F_i(j-pr_i)|$ et donc

$$\psi_{i,d_i-j+1,D-p} = \psi_{i,j,p}$$

□

Proposition 27 *Soit F_i une fonction paire et convexe vérifiant la condition (5.2) p 94. Pour tout triplet (i,j,k), on a*

$$C_{i,j,k} = C_{i,d_i-j+1,D-k+1}$$

Démonstration. On considère le triplet (i,j,k). Montrons que $C_{i,j,k} = C_{i,d_i-j+1,D-k+1}$. D'après le Lemme 25, on a $Z^*_{i,j} = D - Z^*_{i,d_i-j+1} + \epsilon$ avec $\epsilon \in \{0,1\}$. On note $\overline{\epsilon}$ l'entier $1-\epsilon$.

– Supposons que $k < Z^*_{i,j}$. Le Lemme 26 implique que :

$$\begin{aligned}\sum_{p=k}^{Z^*_{i,j}-1} \psi_{i,j,p} &= \sum_{p=k}^{D-Z^*_{i,d_i-j+1}-\overline{\epsilon}} \psi_{i,d_i-j+1,D-p} \\ &= \sum_{p=Z^*_{i,d_i-j+1}+\overline{\epsilon}}^{D-k} \psi_{i,d_i-j+1,p}\end{aligned}$$

Si $\overline{\epsilon} = 1$, alors $k_{i,j}$ est entier et donc on peut en déduire d'après la preuve du Lemme 25 que k_{i,d_i-j+1} est aussi un entier. De ce résultat et de la définition (5.5) p 98 de k_{i,d_i-j+1} on obtient $\psi_{i,d_i-j+1,Z^*_{i,d_i-j+1}} = 0$. Ainsi,

$$\sum_{p=k}^{Z^*_{i,j}-1} \psi_{i,j,p} = \sum_{p=Z^*_{i,d_i-j+1}}^{(D-k+1)-1} \psi_{i,d_i-j+1,p}$$

Donc, d'après définition (5.6) p 98 des coûts, on a

$$C_{i,j,k} = C_{i,d_i-j+1,D-k+1}$$

– Supposons que $k = Z^*_{i,j}$. Si $k_{i,j}$ est entier, k_{i,d_i-j+1} est aussi entier et égal à $D-k$ (d'après le Lemme 25). Ainsi, on a $C_{i,d_i-j+1,D-k+1} = \sum_{p=Z^*_{i,d_i-j+1}}^{D-k+1-1} \psi_{i,d_i-j+1,p} = \psi_{i,j,Z^*_{i,d_i-j+1}} = 0$. Comme $k = Z^*_{i,j}$, on a $C_{i,j,k} = 0$. On obtient donc,

$$C_{i,j,k} = C_{i,d_i-j+1,D-k+1}$$

– Supposons que $k > Z^*_{i,j}$. Le Lemme 26 implique

$$\begin{aligned}\textstyle\sum_{p=Z^*_{i,j}}^{k-1} \psi_{i,j,p} &= \textstyle\sum_{p=D-Z^*_{i,d_i-j+1}+\epsilon}^{k-1} \psi_{i,d_i-j+1,D-p} \\ &= \textstyle\sum_{p=D-k+1}^{Z^*_{i,d_i-j+1}-\epsilon} \psi_{i,d_i-j+1,p}\end{aligned}$$

Si $\epsilon = 0$, alors k_{i,d_i-j+1} est entier et $\psi_{i,d_i-j+1,Z^*_{i,d_i-j+1}} = 0$. Ainsi, on a

$$\sum_{p=Z^*_{i,j}}^{k-1} \psi_{i,j,p} = \sum_{p=D-k+1}^{Z^*_{i,d_i-j+1}-1} \psi_{i,d_i-j+1,p}$$

Donc d'après la définition (5.6) p 98 des coûts, on a

$$C_{i,j,k} = C_{i,d_i-j+1,D-k+1}$$

Ainsi, quelles que soient les valeurs relatives de k et $Z^*_{i,j}$, on a

$$C_{i,j,k} = C_{i,d_i-j+1,D-k+1}$$

□

Les Propositions 26 et 27 impliquent que l'ensemble des arêtes et des coûts sont symétriques, mais pas nécessairement le couplage optimal. Cependant, dans de nombreux cas, ce couplage est symétrique et nous adoptons une représentation où seule la première moitié du graphe est dessinée. Le symbole ... indique alors que le graphe n'est pas représenté intégralement (voir Figure A.3).

Exemple 2. Nous considérons l'exemple précédent avec $d = (1, 1, 4, 4)$ où $d_1 = d_2 = 1$ et $d_3 = d_4 = 4$. Le graphe de la Figure A.2 est symétrique. Ainsi, on peut le simplifier pour obtenir le graphe de la Figure A.3.

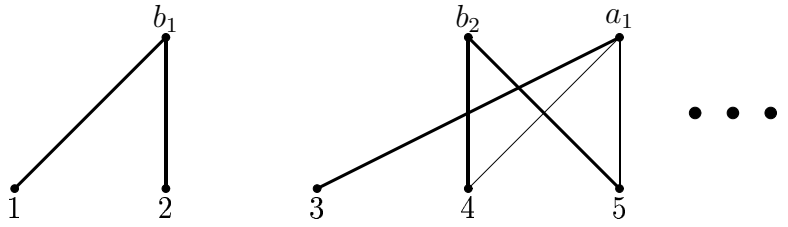


FIG. A.3 – Graphe biparti pour $d_1 = d_2 = 1$, $d_3 = d_4 = 4$ et $B = \frac{7}{10}$

A.1.1.3 Symétrie des solutions

On peut remarquer que la symétrie du graphe implique que pour certaines instances, il existe toujours des solutions optimales symétriques. On peut envisager par exemple le cas où les demandes sont toutes paires et voir facilement qu'il existe un couplage symétrique optimal pour tout problème min-sum ou pour max-abs.

Plus précisément, si un couplage existe pour une borne B donnée, on peut alors montrer qu'il en existe un qui répartit les pièces de chaque type équitablement entre les deux moitiés du graphe.

Les lemmes suivants permettent de démontrer ce résultat.

Lemme 27 *Soit i un type de pièces et B une borne inférieure strictement à $3/2$. Si d_i est impaire et D est pair, alors pour tout $i = 1, 2, \ldots, n$, les pièces (i, j), $j = 1, 2, \ldots, (d_i - 1)/2$ sont produites avant l'instant $D/2$.*

Démonstration. Soit i un type de pièces tel que d_i est impaire. Soit $j \in \{1, 2, \ldots, (d_i - 1)/2\}$.

$$\begin{aligned} L(i,j) &= \left\lfloor \tfrac{j-1+B}{r_i} + 1 \right\rfloor \\ &\leq \left\lfloor \tfrac{(d_i-1)/2+B-1}{r_i} + 1 \right\rfloor \\ &\leq \tfrac{D}{2} + 1 + \left\lfloor \tfrac{B-3/2}{r_i} \right\rfloor \end{aligned}$$

Comme $B < 3/2$, on a

$$L(i,j) \leq \frac{D}{2}$$

□

Lemme 28 *Soit i un type de pièces et $B < 1$ une borne. Si d_i est paire et D est paire, alors les pièces (i, j), $j = 1, 2, \ldots, d_i/2$ sont produites avant ou à l'instant $D/2$.*

Démonstration. Soit i un type de pièce tel que d_i est paire. Soit $j \in \{1, 2, \ldots, d_i/2\}$.

$$\begin{aligned} L(i,j) &= \left\lfloor \frac{j-1+B}{r_i} + 1 \right\rfloor \\ &\leq \left\lfloor \frac{d_i/2+B-1}{r_i} + 1 \right\rfloor \\ &\leq \frac{D}{2} + 1 + \left\lfloor \frac{B-1}{r_i} \right\rfloor \end{aligned}$$

Comme $B < 1$, on a

$$L(i,j) \leq \frac{D}{2}$$

□

Les Lemmes 27 et 28 et la symétrie du graphe impliquent donc que si la demande totale est paire, alors les pièces (i,j), $j = 1, 2, \ldots, \lfloor d_i/2 \rfloor$ sont produites avant l'instant $D/2$ et les pièces (i,j), $j = \lceil d_i/2 \rceil + 1, \lceil d_i/2 \rceil + 2, \ldots, d_i$ sont produites après $D/2$ dans toute solution dont le maximum des déviations est inférieur strictement à 1.

Proposition 28 *Pour tout problème max-abs ou min-sum B-borné avec $B < 1$, si la demande totale est paire, il existe toujours une solution optimale telle que*

$$y_{i,j,k} = 1 \Rightarrow \left(y_{i,d_i-j+1,D-k+1} = 1 \text{ et } \begin{cases} k \leq D/2 & \text{si } j \leq \lfloor d_i/2 \rfloor \\ k \geq D/2 + 1 & \text{si } \lceil d_i/2 \rceil + 1 \leq j \end{cases} \right) \quad \text{(A.4)}$$

Démonstration. Directement de la Proposition 26 et des Lemmes 27 et 28. □

Lemme 29 *Soient i un type de pièces et $B \leq 1$ une borne. Si d_i est impaire et D est impaire, alors pour tout $i = 1, 2, \ldots, n$, les pièces (i,j), $j = 1, 2, \ldots, (d_i - 1)/2$ sont produites avant ou à l'instant $(D-1)/2$.*

Démonstration. Soit i un type de pièces tel que d_i est impaire. Soit $j \in \{1, 2, \ldots, (d_i - 1)/2\}$.

$$\begin{aligned} L(i,j) &= \left\lfloor \frac{j-1+B}{r_i} + 1 \right\rfloor \\ &\leq \left\lfloor \frac{(d_i-1)/2+B-1}{r_i} + 1 \right\rfloor \\ &\leq \frac{D-1}{2} + 1 + \left\lfloor \frac{1}{2} - \frac{B-3/2}{r_i} \right\rfloor \end{aligned}$$

Comme $B \leq 1$ et $D > d_i$, on a $\frac{B-3/2}{r_i} < -\frac{1}{2}$ et donc

$$L(i,j) \leq \frac{D-1}{2}$$

□

Lemme 30 *Soit i un type de pièces et $B \leq 1$ une borne. Si d_i est paire et D est impaire, alors s'il existe un couplage parfait, il en existe un tel que les pièces (i,j), $j = 1, 2, \ldots, d_i/2$ sont produites avant l'instant $(D+1)/2$.*

Démonstration. Soit i un type de pièces tel que d_i est paire. Soit $j \in \{1, 2, \ldots, d_i/2\}$.

$$\begin{aligned} L(i,j) &= \left\lfloor \frac{j-1+B}{r_i} + 1 \right\rfloor \\ &\leq \left\lfloor \frac{d_i/2+B-1}{r_i} + 1 \right\rfloor \\ &\leq 1 + \left\lfloor \frac{D}{2} \right\rfloor \\ &\leq \frac{D+1}{2} \end{aligned}$$

Soit i un type de pièces tel que d_i est impaire.

$$\begin{aligned} L(i,(d_i+1)/2) &= \left\lfloor \frac{\frac{d_i+1}{2}-1+B}{r_i} + 1 \right\rfloor \\ &= 1 + \left\lfloor \frac{D}{2} + \frac{D}{2\,d_i} \right\rfloor \\ &\geq \frac{D+1}{2} \end{aligned}$$

Lorsque l'on cherche un couplage parfait dans le graphe, on peut utiliser la règle EDD (*Earliest Due Date*) avec comme date d'échéance pour la pièce (i,j) son instant au plus tard de production $L(i,j)$ [82]. Si i_1 est un type de pièce de demande paire et i_2 un type de pièces de demande impaire, on aura $L(i_1,j) \leq L(i_2,(d_{i_2}+1)/2)$ pour tout $j \in \{1, 2, \ldots, d_{i_1}/2\}$. La règle EDD place donc si c'est possible les pièces (i_1,j), $j = 1, 2, \ldots, d_{i_1}/2$ avant la pièce $(i_2,(d_{i_2}+1)/2)$. Supposons qu'il existe un couplage parfait pour la borne B. Utilisons la règle EDD pour placer les pièces (i,j) pour $i = 1, 2, \ldots, n$ et $j = 1, 2, \ldots, \lfloor d_i/2 \rfloor$. Si une de ces pièces (i_1, d_{i_1}) est placée en $\frac{D+1}{2}$, alors par symétrie du graphe, il est impossible de placer toutes les pièces (i,j), $i = 1, 2, \ldots, n$ $j = \lfloor d_{i_1}/2 \rfloor + 1, \lfloor d_{i_1}/2 \rfloor + 2, \ldots, d_{i_1}$ après $\frac{D+1}{2}$. Ainsi, aucune des pièces $(i_1, d_{i_1}/2)$ n'est placée en $\frac{D+1}{2}$. □

Remarque. La pièce centrale est dans ce cas la pièce $(i,(d_i+1)/2)$ d'un type de pièce i de demande impaire.

On obtient donc immédiatement le résultat suivant :

Lemme 31 *Si toutes les demandes sont paires sauf une, alors le problème max-abs a une solution optimale symétrique.*

Pour répartir les pièces $(i,(d_i+1)/2)$ entre les deux moitiés du graphe, il est possible de les placer alternativement de chaque côté de $\lfloor \frac{D+1}{2} \rfloor$ dans l'ordre des d_i décroissants. En effet, on a $E(i,(d_i+1)/2) = D - L(i,(d_i+1)/2) + 1$ pour tout type i de demande impaire d'après la Proposition 26 donc on peut placer les pièces $(i,(d_i+1)/2)$ des types i de demande impaire indifféremment d'un côté ou de l'autre. S'il existe un couplage pour une borne $B \leq 1$ donnée et qu'on utilise la règle EDD pour placer toutes les pièces (i,j), avec $i = 1, 2, \ldots, n$ et $j = 1, 2, \ldots, \lfloor d_i/2 \rfloor$, les instants de productions restants avant $(D+1)/2$ et après $(D+1)/2$ seront occupés par des pièces $(i,(d_i+1)/2)$ de types i de demande impaire. Comme $d_i > d_{i'}$ implique que $E(i,(d_i+1)/2) < E(i',(d_{i'}+1)/2)$ on peut placer les pièces $(i,(d_i+1)/2)$ des types i de demande impaire en attribuant les instants restants les plus éloignés du centre aux types de demandes les plus faibles. Ces instants étant symétriques par rapport à $\frac{D+1}{2}$ on obtient le résultat suivant :

Proposition 29 *Pour toute donnée $d_1, d_2, \ldots, d_n$ telle que les demandes $d_1, d_2, \ldots, d_p$, $p \leq n$ sont impaires et telle que pour tout $i = 1, 2, \ldots, \lfloor n/2 \rfloor$, on a $d_{2i-1} = d_{2i}$, s'il existe un couplage parfait pour la borne B du maximum des déviations, alors il existe un couplage parfait symétrique par rapport à $\lfloor \frac{D+1}{2} \rfloor$.*

L'existence de solutions symétriques ou presque symétriques a inspiré les heuristiques présentées dans la Section A.1.2.

A.1.2 Heuristiques polynomiales

La représentation d'une solution des problèmes d'ordonnancement juste-à-temps par la description complète de l'ensemble des dates d'usinage des pièces n'est pas polynomiale de la taille de l'instance. Nous proposons dans cette section de nous intéresser à des solutions suffisamment régulières pour être représentées de façon compacte et polynomiale.

A.1.2.1 Idée du codage des solutions

Comme remarqué dans la Section A.1.1.3, certaines instances admettent des solutions optimales symétriques pour les problèmes max-abs ou min-sum. Dans ces solutions, pour chaque type $i = 1, 2, \ldots, n$, les $\lfloor d_i/2 \rfloor$ premières pièces sont placées avant ou à l'instant $\lfloor D/2 \rfloor$ alors que les $\lfloor d_i/2 \rfloor$ dernières pièces seront disposées après ou à l'instant $\lfloor (D+1)/2 \rfloor + 1$. L'idée est donc de créer un ensemble M_1 contenant les pièces $(i, (d_i+1)/2)$ dont les types i sont de demandes impaires et de placer les pièces de cet ensemble autour de l'instant $\lfloor (D+1)/2 \rfloor$. Les pièces restantes sont disposées de part et d'autre de $\lfloor (D+1)/2 \rfloor + 1$. On obtient ainsi deux ensembles de pièces. On ne cherchera à placer les pièces que du premier ensemble et on déduira les positions des autres par symétrie par rapport à $\lfloor (D+1)/2 \rfloor + 1$. En répétant cette démarche, on obtiendra l'emplacement de toutes les pièces dans l'horizon de production (voir Figure A.4).

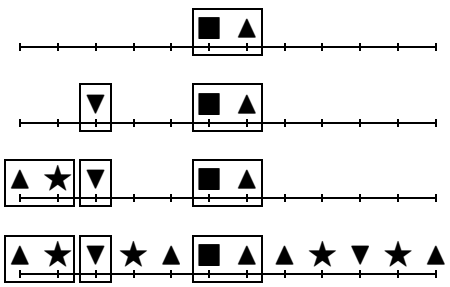


FIG. A.4 – Construction d'une solution symétrique pour l'instance $d_{\blacksquare} = 1$, $d_{\blacktriangledown} = 2$, $d_{\bigstar} = 4$, $d_{\blacktriangle} = 5$, $D = 12$

À chaque pas, on ne calcule explicitement que la disposition d'un ensemble d'au maximum n pièces, et on effectuera exactement $P = \lfloor \log_2 \max d_i \rfloor + 1$ pas. La taille de l'ensemble de ces instants d'usinage est donc un $O(n \log(\max d_i))$, ce qui est bien polynomial de la taille de l'instance.

Les autres instants d'usinage des solutions proposées sont implicites et peuvent être fixés par symétrie (comme sur la Figure A.4) ou comme proposé dans la Section A.1.1.3.

A.1.2.2 Versions possibles

Pour ordonner les pièces des ensembles M_1, $M_2, \ldots,$ M_P, on peut envisager plusieurs stratégies.

L'ensemble M_1, si l'on optimise le maximum des déviations, sera rempli de l'extérieur (instants de production les plus éloignés de son centre) vers l'intérieur par les pièces mises dans l'ordre de leurs demandes croissantes. En effet, les pièces de demandes les plus faibles ont, pour une borne B du maximum des déviations fixée, les intervalles les plus larges de part et d'autre de leur instant idéal de production.

Pour l'ensemble M_P, on placera par contre les pièces dans l'ordre de leurs demandes décroissantes de l'instant 1 à l'instant $|M_P|$. Cette méthode permet de placer optimalement les pièces de M_P quel que soit le critère min-sum ou max-abs.

Pour les autres ensembles M_2, $M_3, \ldots,$ M_{P-1} l'une comme l'autre des deux stratégies précédentes peut s'avérer meilleure, mais il est préférable en moyenne d'appliquer la première, comme le montre les tests de la Table A.1.

De plus, si une symétrie par rapport à l'instant $\lfloor (D+1)/2 \rfloor + 1$ est toujours effectuée, on peut aussi choisir comme convention, pour les sous-problèmes qui suivront, d'effectuer une simple copie et pas une symétrie, comme sur la Figure A.5.

FIG. A.5 – Construction d'une solution symétrique pour l'instance $d_{\blacksquare} = 1$, $d_{\blacktriangledown} = 2$, $d_{\bigstar} = 4$, $d_{\blacktriangle} = 5$, $D = 12$ par l'une des méthodes *copie* M_1 ou *copie* M_P

Heuristique	Meilleur maximum des déviations parmi les 4 heuristiques
symetrie, M_1	3 964 750
symetrie, M_P	2 023 923
copie, M_1	3 081 589
copie, M_P	1 347 885

TAB. A.1 – Comparaison des heuristiques sur 5 000 000 d'instances

Dans la Table A.1, M_1 signifie que les ensembles M_2, $M_3, \ldots,$ M_{P-1} sont remplis comme M_1, et M_P signifie que les ensembles M_2, $M_3, \ldots,$ M_{P-1} sont remplis comme M_P, *symetrie* que les instants non affectés le sont par symétrie, *copie* qu'ils le sont par copie. Les instances sur lesquelles ces heuristiques ont été testées ont une demande totale inférieure ou égale à 110 et des demandes différentes pour chaque type. Pour ces instances, les performances sont assez mauvaises puisque le meilleur des maximums des déviations ainsi obtenu dépasse parfois 3.

Nous n'avons pas pu, par manque de temps, évaluer les performances de ces heuristiques par rapport à l'optimum et en particulier tester des améliorations pour le cas où certains types de tâches ont la même demande. En effet, ces heuristiques se comportent particulièrement mal dans le cas où beaucoup de types de pièces sont de demandes 1. Nous avons envisagé d'agréger dans ce cas les types de même demande mais n'avons pu tester cette idée.

A.2 Programmes linéaires pour la résolution par CPLEX

Pour trouver les exemples de la Section 5.2.2, nous avons choisi de tester de façon exhaustive toutes les instances pour des valeurs données de demande totale D ou toutes les instances pour D et n donnés. Nous avons utilisé ILOG-CPLEX et choisi des formulations efficaces pour les problèmes que nous voulions résoudre.

A.2.1 Résolution des problèmes min-sum

Pour les problèmes min-sum, il n'est pas nécessaire d'adopter une formulation en nombres entiers, ce qui rend la résolution très rapide. En fait, le polyèdre (A.5) des contraintes du problème d'affectation est entier.

$$\begin{array}{lll} P = & \{y; & \\ & \sum_{i=1}^{n} \sum_{j=1}^{d_i} y_{i,j,k} = 1, & k = 1, 2, \ldots, D \\ & \sum_{k=1}^{D} y_{i,j,k} = 1, & i = 1, 2, \ldots, n;\ j = 1, 2, \ldots, d_i\} \end{array} \tag{A.5}$$

Ainsi, on peut obtenir rapidement les valeurs optimales des fonctions objectif sum-abs ou sum-sqr, mais pas forcément une séquence optimale entière.

A.2.2 Résolution des problèmes max-abs

On peut formuler le problème max-abs de la façon suivante :

$$\begin{array}{lll} & \textit{minimiser} \quad \max_{i,k} |x_{i,k} - kr_i| & \\ \text{sc} & \sum_{i=1}^{n} x_{i,k} = k, & k = 1, 2, \ldots, D \\ & x_{i,D} = d_i, & i = 1, 2, \ldots, n \\ & 0 \leq x_{i,k} - x_{i,k-1}, & i = 1, 2, \ldots, n;\ k = 2, 3, \ldots, D \\ & x_{i,k} \in \mathbb{N}, & i = 1, 2, \ldots, n;\ k = 1, 2, \ldots, D \end{array} \tag{A.6}$$

Ce problème n'est pas linéaire, mais est facilement transformé pour obtenir le programme

linéaire en nombre entier suivant :

$$\begin{array}{lll} & \textit{minimiser} \quad B & \\ \text{sc} & \sum_{i=1}^{n} x_{i,k} = k, \quad k = 1, 2, \ldots, D & (A.7.a) \\ & x_{i,D} = d_i, \quad i = 1, 2, \ldots, n & (A.7.b) \\ & 0 \leq x_{i,k} - x_{i,k-1}, \quad i = 1, 2, \ldots, n;\ k = 2, 3, \ldots, D & (A.7.c) \\ & x_{i,k} - kr_i \leq B, \quad i = 1, 2, \ldots, n;\ k = 1, 2, \ldots, D & (A.7.d) \\ & -x_{i,k} + kr_i \leq B, \quad i = 1, 2, \ldots, n;\ k = 1, 2, \ldots, D & (A.7.e) \\ & x_{i,k} \in \mathbb{N}, \quad i = 1, 2, \ldots, n;\ k = 1, 2, \ldots, D & (A.7.f) \end{array} \tag{A.7}$$

Les inégalités (A.7.d) et (A.7.e) indiquent que B est plus grande que toutes les déviations et l'objectif étant de minimiser B, les deux problèmes sont équivalents.

Cette formulation n'est pas nécessairement satisfaisante puisque le polyèdre défini par les contraintes (A.7.a) à (A.7.c) n'est pas entier. CPLEX peut tout de même le résoudre efficacement la plupart du temps. Cependant, si l'instance présente trop de symétries (trop de types de demandes identiques), la résolution peut prendre plusieurs heures. Ainsi, on n'essaie pas de résoudre directement le problème max-abs, mais on préférera la recherche dichotomique proposée dans [82].

On peut en fait décrire un problème max-abs en tant que problème d'affectation. On calcule les intervalles $[E(i,j) \,..\, L(i,j)]$ pour tout $i = 1, 2, \ldots, n$ et tout $j = 1, 2, \ldots, d_i$ pour la borne B choisie, et on utilise les coûts suivants :

$$C_{i,j,k} = \begin{cases} 0 \text{ si } k \in [E(i,j) \,..\, L(i,j)] \\ 1 \text{ sinon} \end{cases} \tag{A.8}$$

Le problème max-abs a une solution si et seulement si le problème min-sum avec ces coûts a une solution optimale de coût 0.

En utilisant la dichotomie, on trouve la valeur minimale de B telle que le problème a une solution en peu de pas et en très peu de temps puisqu'une résolution prend alors moins de 0.2 secondes.

A.2.3 Comparaison des problèmes min-sum et min-sum B-bornés

La méthode décrite dans la Section A.2.1 ne peut pas être utilisée pour les problèmes B-bornés puisque le polytope (A.9) n'est pas entier.

$$\begin{array}{ll} P_B = & \{y; \\ & \sum_{i=1}^{n} \sum_{j=1}^{d_i} y_{i,j,k} = 1, \quad k = 1, 2, \ldots, D \\ & \sum_{k=1}^{D} y_{i,j,k} = 1, \quad i = 1, 2, \ldots, n;\ j = 1, 2, \ldots, d_i \\ & \sum_{p=1}^{k} \sum_{j=1}^{d_i} y_{i,j,p} - kr_i \leq B, \quad i = 1, 2, \ldots, n \\ & kr_i - \sum_{p=1}^{k} \sum_{j=1}^{d_i} y_{i,j,p} \leq B, \quad i = 1, 2, \ldots, n\} \end{array} \tag{A.9}$$

Dans ce cas, il faut imposer y entier. La résolution est tout de même rapide. La valeur optimale de la fonction objectif du problème B-borné peut donc être obtenue par cette méthode, mais elle n'est pas forcément nécessaire : si on veut uniquement savoir si elle sera

égale ou pas à celle du problème non borné, on peut encore formuler le problème avec y réel. Pour cela, on calcule d'abord l'optimal pour le problème non borné, comme dans la Section A.2.1. On note S cette valeur. Au polytope P de (A.5) on ajoute la contrainte $\sum_{(i,j,k)} C_{i,j,k} y_{i,j,k} = S$ où C est la matrice des coûts pour le problème min-sum. On note P_S le polytope obtenu. Comme fonction objectif, on minimise la somme des coûts définis par l'équation (A.8). Comme précédemment, si l'optimum de cette fonction objectif est 0, alors il existe une solution de P_S dont le maximum des déviations est inférieur à B. Dans ce cas, les deux problèmes ont la même solution optimale. Sinon, aucun élément de P_S n'est optimal pour le problème non borné.

A.3 Optimisations simultanées pour trois types de pièces

Pour un énoncé T donné, nous noterons *plus D-petit exemple* un exemple de T tel qu'il n'existe aucune instance avec une demande totale $D' < D$ qui soit un exemple de l'énoncé T.

Pour deux types de pièces, il était possible pour toute instance d'optimiser simultanément toutes les fonctions objectif. Ce n'est plus le cas pour trois types de pièces.

Proposition 30 *Pour $n = 3$, il existe des instances du problème JAT tels qu'aucune séquence n'optimise sum-abs et max-abs simultanément.*

Démonstration. Pour $n = 3$, le plus $D-$petit exemple est l'instance $d = (2, 7, 17)$. Pour cette instance, le minimum du maximum des déviations est $B^* = \frac{16}{26}$ et le graphe associé est celui de la Figure A.6 où seule la partie gauche du graphe a été représentée. Les arêtes en gras sont dans tout couplage parfait du graphe.

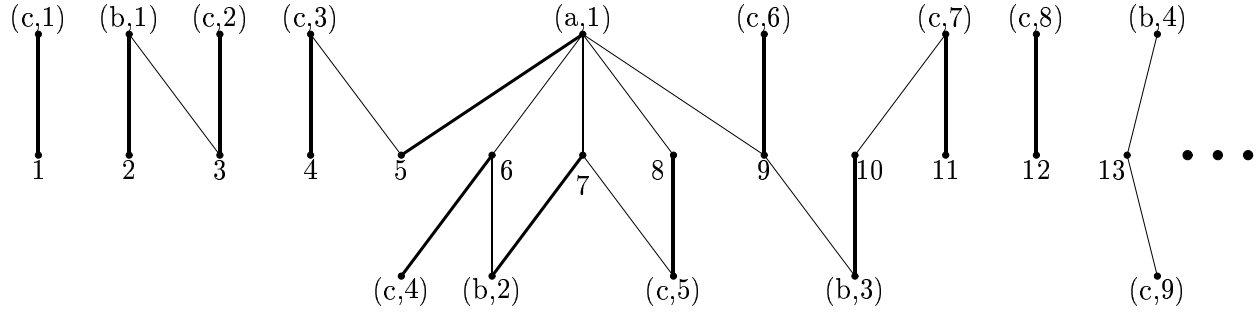


FIG. A.6 – Graphe biparti pour $d_a = 2$, $d_b = 7$, $d_c = 17$ et $B = \frac{16}{26}$

Ainsi, le graphe a exactement deux couplages parfaits correspondant à la même somme des déviations : $\frac{572}{26}$ (qui est atteinte pour la séquence $S = (c, b, c, c, a, c, b, c, c, b, c, c, b, c, c, c, b, c, c, b, c, a, c, c,$). La séquence $S = (c, b, c, c, b, c, c, a, c, b, c, c, b, c, c, c, b, c, a, c, c, b, c, c, b, c)$ a une somme des déviations égale à $\frac{556}{26}$. Aucune séquence n'est alors minimale pour les objectifs max-abs et sum-abs simultanément. □

A.4 Couplages et solutions duales pour des exemples

Dans cette section, on reprend plus en détail certains exemples de la Table 5.2 afin d'illustrer les techniques proposées à la Section 5.2.2.3 p 105.

A.4.1 Instance pour laquelle $SM1$ est vraie et $AM1$ est faux

Pour l'instance $d = (1^9, 6^4)$ le problème sum-sqr a une solution optimale dont le maximum des déviations est inférieur à 1. Cette solution est représentée sur le graphe de la Figure A.7. Les arêtes en gras sont les arêtes du couplage optimal, les nombres en italique sont les coûts des arêtes définis dans la Section 5.1.2.3 p 98, les nombres en gras sont les valeurs d'une solution duale vérifiant la condition (5.11) p 100. Les coûts et la solution duale ont été multipliés par D afin d'obtenir des entiers. Toutes les arêtes ne sont pas représentées pour des questions d'encombrement.

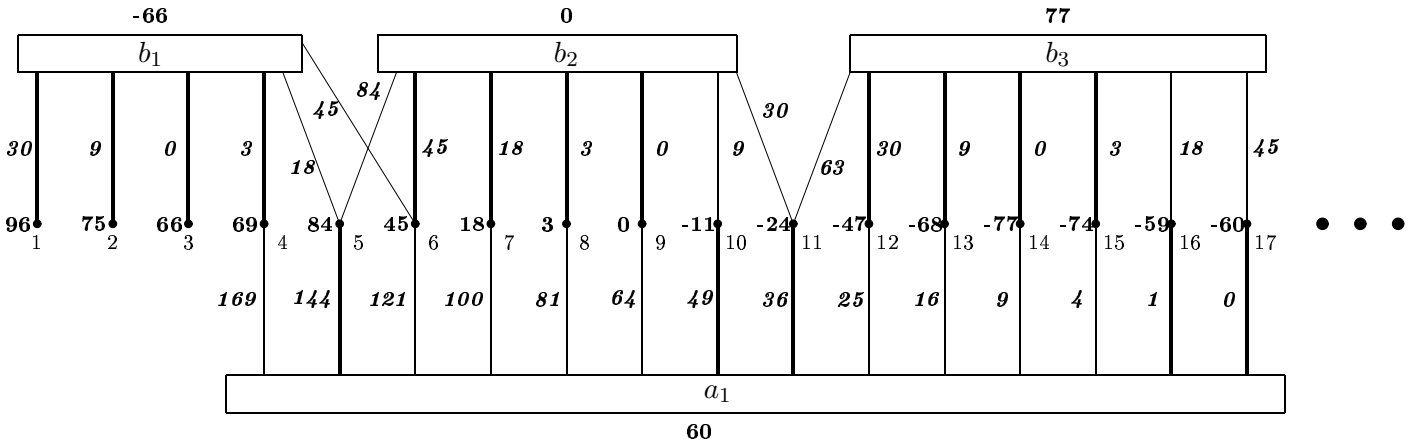


FIG. A.7 – Graphe biparti pour $d = (1^9, 6^4)$ et poids du problème sum-sqr

La Proposition 17 implique que cette solution, notée Y^* est optimale. Son maximum des déviations est $\frac{30}{33}$ et est bien inférieur à 1.

On considère la séquence $S = (b_1^4, b_2^4, a_1^3, b_3^4, a_1^3, b_4^4, a_1^3, b_5^4, b_6^4)$ où α^s représente α répété s fois, a_1 est la $1^{ère}$ pièce d'un des types de demande 1 et b_j est la $j^{ième}$ pièce d'un des types de demande 6. On note y la variable d'affectation correspondante. On a

$$\sum_{i=1}^{n}\sum_{j=1}^{d_i}\sum_{k=1}^{D} C^s_{i,j,k}\, Y^*_{i,j,k} > \sum_{i=1}^{n}\sum_{j=1}^{d_i}\sum_{k=1}^{D} C^a_{i,j}\, y_{i,j,k}$$

Ainsi, pour toute solution optimale y^* du problème sum-abs,

$$\sum_{i=1}^{n}\sum_{j=1}^{d_i}\sum_{k=1}^{D} C^s_{i,j,k}\, Y^*_{i,j,k} > \sum_{i=1}^{n}\sum_{j=1}^{d_i}\sum_{k=1}^{D} C^a_{i,j}\, y^*_{i,j,k}$$

Donc grâce au Corollaire 7, il n'existe pas de solution optimale au problème sum-abs dont le maximum des déviations est inférieur à 1.

Pour montrer que l'ensemble des solutions optimales des problèmes sum-abs et sum-sqr sont disjoints pour l'instance $d = (1^9, 6^4)$, nous obtenons dans un premier temps les valeurs optimales des fonctions objectif de chaque problème, notées par exemples S_a et S_s puis nous optimisons la fonction objectif sum-sqr en ajoutant une contrainte qui fixe la valeur de la fonction objectif sum-abs à sa valeur optimale. Si la valeur optimale obtenue est S_s, alors on peut optimiser les fonctions objectif sum-abs et sum-sqr simultanément. C'est le cas pour l'instance $d = (1^9, 6^4)$.

A.4.2 Instance pour laquelle AS, $AM1$ et $SM1$ sont fausses

Pour une borne $B = 1$ du maximum des déviations, le graphe correspondant à l'instance $d = (1^7, 6^4)$ est représenté sur la Figure A.8. Comme les coûts sont identiques pour $B \leq 1$ pour les problèmes B-bornés sum-abs et sum-sqr, la solution représentée est optimale dans les deux cas. Cette solution n'est optimale ni pour sum-abs ni pour sum-sqr. Une solution optimale et une solution duale optimale sont représentées sur le graphe de la Figure A.9 et A.10.

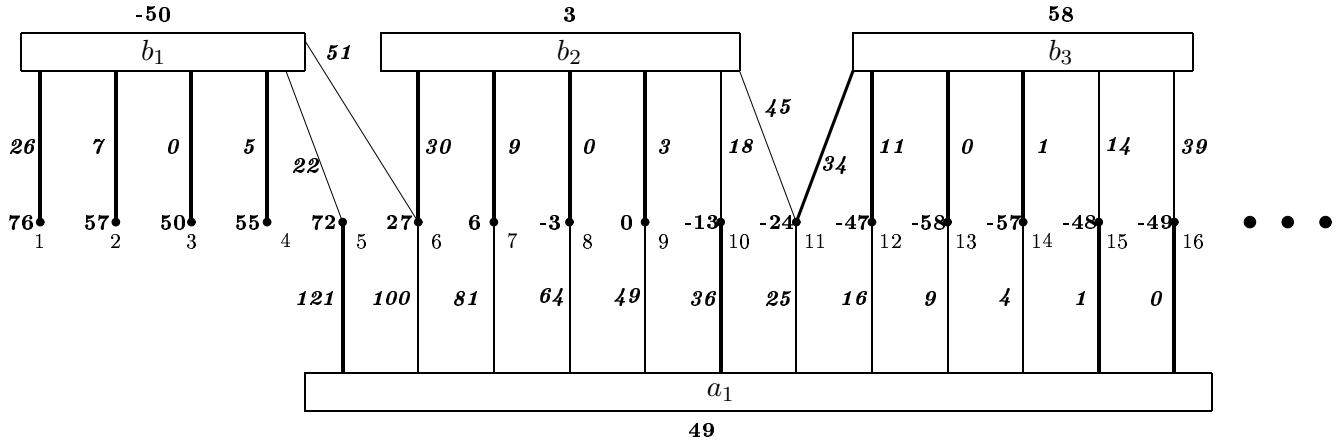


FIG. A.8 – Graphe biparti pour $d = (1^7, 6^4)$ et des poids correspondant aux problèmes min-sum 1-bornés

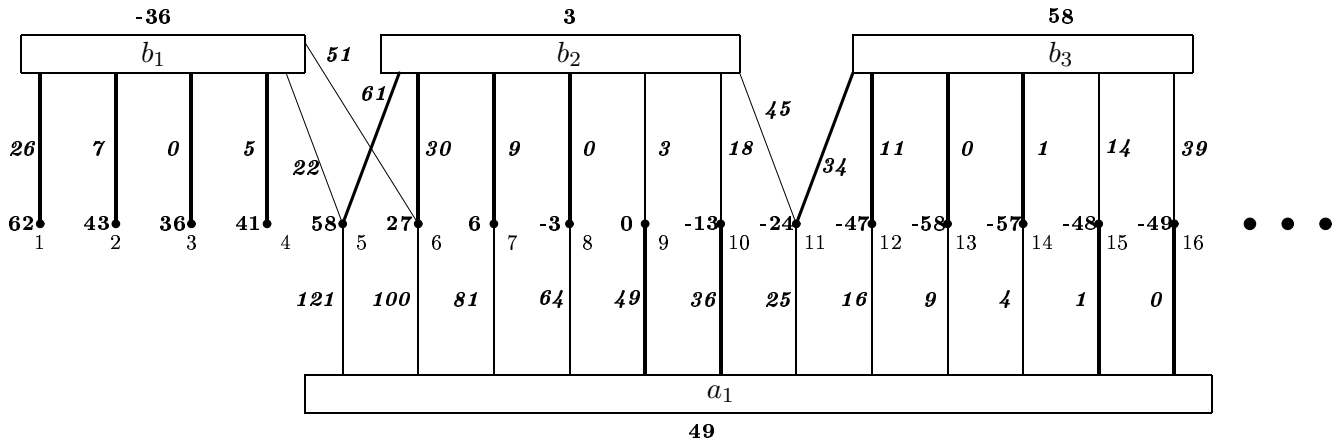


FIG. A.9 – Graphe biparti pour $d = (1^7, 6^4)$ et poids pour le problème sum-abs

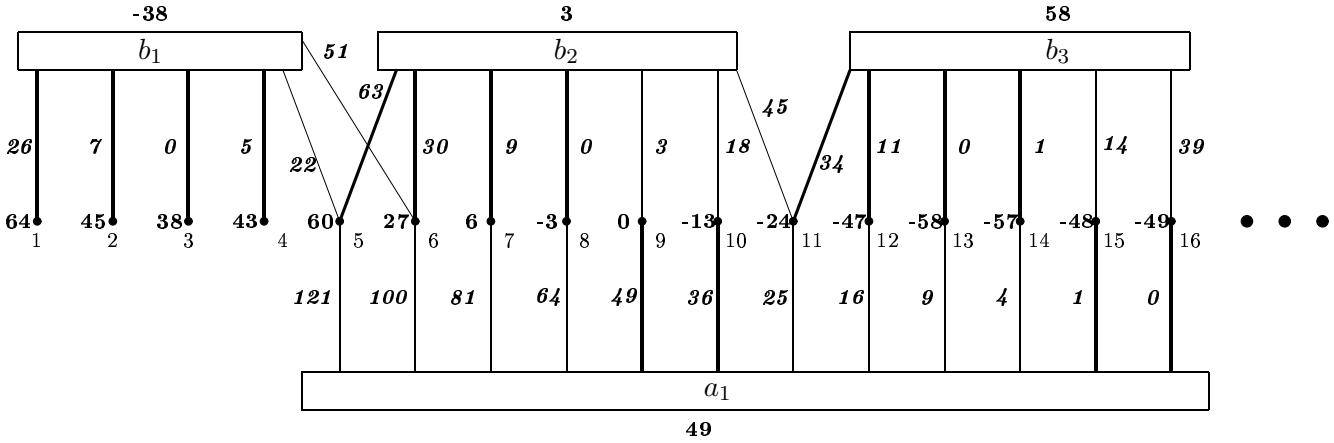


FIG. A.10 – Graphe biparti pour $d = (1^7, 6^4)$ et poids pour le problème sum-sqr

Index