Optimisation du chargement des laveurs dans un service de stérilisation hospitalière : ordonnancement, simulation, couplage

Onur Ozturk

UNIVERSITÉ DE GRENOBLE

THÈSE

Pour obtenir le grade de

DOCTEUR DE L'UNIVERSITÉ DE GRENOBLE

Spécialité : **Génie Industriel**

Arrêté ministériel : 7 août 2006

Présentée par

« Onur OZTURK »

Thèse dirigée par **« Maria DI MASCOLO»** et
codirigée par **« Marie-Laure ESPINOUSE »** et **« Alexia GOUIN»**

préparée au sein du **Laboratoire G-SCOP**
dans **l'École Doctorale I-MEP²**

Optimisation du chargement des laveurs dans un service de stérilisation hospitalière : ordonnancement, simulation, couplage

Thèse soutenue publiquement le 16 juillet 2012, devant le jury composé de :

M. András SEBÖ
Directeur de recherche au Laboratoire G-SCOP, Président
M. Jean-Charles BILLAUT
Professeur à l'Université François Rabelais de Tours, Rapporteur
M. Lionel DUPONT
Professeur à l'Ecole des Mines d'Albi Carmaux, Rapporteur
Mme. Evren SAHIN
Maitre de Conférences à l'Ecole Centrale de Paris, Examinateur
M. Michel GOURGAND
Professeur à l'Université Blaise Pascal, Examinateur
Mme. Maria DI MASCOLO
Chargée de recherche au Laboratoire G-SCOP, Directrice
Mme. Marie-Laure ESPINOUSE
Professeur à l'Université Joseph Fourrier, Co-directrice
Mme. Alexia GOUIN
Maitre de Conférences à l'Université Joseph Fourrier, Co-directrice

Remerciements

Avant tout, je tiens à remercier mes directrices de thèse, Maria Di Mascolo, Marie-Laure Espinouse et Alexia Gouin, pour m'avoir aidé, encouragé, orienté et conseillé durant mes années de doctorat. Grâce à leurs conseils, leurs commentaires et leurs connaissances, j'ai pu mener à bien cette thèse, progresser et élargir mes connaissances dans le monde de la recherche. Je remercie également Andras Sebo pour m’avoir introduit aux problèmes d’optimisation combinatoires, et pour avoir présidé mon jury de thèse.

Je remercie aussi chaleureusement les membres de mon jury : Jean-Charles Billaut et Lionel Dupont qui m'ont fait l'honneur d'accepter de rapporter cette thèse. Je remercie également Michel Gourgand et Evren Sahin d’avoir fait partie de mon jury en tant qu’examinateurs. Mes remerciements vont aussi à Bernadette Valence, responsable du service de stérilisation du CHU de Grenoble, pour m’avoir ouvert les portes de son service.

Je remercie également toutes les personnes appartenant à l'environnement de travail que j'ai côtoyé durant ces dernières années, à savoir les enseignants chercheurs, les doctorants et le personnel administratif de G-SCOP.

Pour terminer, je remercie spécialement toute ma famille qui m'a soutenu durant ces années que j’ai passées loin de mon pays.

TABLE DES MATIERES

LISTE DES TABLEAUX

LISTE DES FIGURES

~ INTRODUCTION GENERALE ~

Nous nous intéressons, dans cette thèse, au problème du chargement des laveurs dont l'origine se trouve dans le processus de stérilisation hospitalière. Le service de stérilisation d'un hôpital a pour but de stériliser tout dispositif médical utilisé dans une intervention chirurgicale et qui peut être réutilisé après une opération de stérilisation. Les dispositifs considérés sont donc des « dispositifs médicaux réutilisables (ou DMR) ».

Les étapes du processus de stérilisation sont les suivantes : rinçage, lavage, vérification, conditionnement, et enfin, stérilisation. Après toutes ces étapes, les DMR sont prêts pour être réutilisés. Parmi ces étapes, il y a une partie qui se fait manuellement, et d'autres qui se font automatiquement dans des machines. Nous pouvons compter le rinçage (manuel), la vérification et le conditionnement parmi les étapes manuelles. Le rinçage automatique, le lavage et la stérilisation sont, par contre, des opérations qui se font dans des machines. Notons que l'étape de lavage est souvent un goulot d'étranglement pour les services de stérilisation.

L'étape de lavage est composée de laveurs qui sont capables de traiter plusieurs ensembles de DMR en même temps. Un ensemble de DMR désigne tous les DMR utilisés pour une intervention chirurgicale. Il n'est pas désiré de couper un ensemble de DMR et de le laver dans des laveurs différents, ce qui rendrait difficile la traçabilité des DMR appartenant à un même ensemble. Dans le cas où un ensemble de DMR est coupé et traité par plusieurs laveurs, les DMR utilisés dans une même intervention arriveront à l'étape suivante, i.e. l'étape de conditionnement, à des instants différents. Pour le bon déroulement du conditionnement, il est important que tous les DMR d'un même ensemble soient présents en même temps. Les ensembles de DMR ont tous la même durée de lavage et arrivent dans le service de stérilisation à différents instants tout au long d'une journée. L'étape de lavage peut alors être modélisée comme un problème d'ordonnancement par batch où les ensembles de DMR peuvent être traités comme des tâches ayant des tailles et des dates de disponibilité différentes, et les laveurs comme des machines qui exécutent les tâches. Notre but est d'améliorer l'efficacité de l'étape de lavage, tout en gardant la qualité désirée pour la stérilisation. Pour ce faire, nous allons proposer des méthodes d'optimisation, après une modélisation de l'étape de lavage comme un problème d'ordonnancement par batch.

Pour augmenter l'efficacité de l'étape de lavage, nous nous sommes inspirés des différentes fonctions objectifs étudiées dans la littérature d'ordonnancement. Les premières fonctions objectifs que nous avons considérées sont la minimisation du C_{max} et la minimisation de la $\sum C_j$ qui correspondent à la minimisation de la durée totale de lavage et la minimisation des encours dans le stock de lavage, respectivement. Ensuite, une troisième fonction objectif que nous étudierons consiste à optimiser la durée de pré désinfection des ensembles de DMR. La pré désinfection des DMR commence au moment où les DMR sont placés dans le liquide pré désinfectant, et elle continue jusqu'à ce que les DMR soient rincés. Dans certains services de stérilisation, la durée de pré désinfection des DMR est considérablement longue. Pour éviter l'effet de corrosion provoqué par le liquide pré désinfectant, les DMR sont rincés à la main, puis mis dans un stock d'en-cours pour attendre le lavage. S'il n'y a pas de poste de rinçage manuel dans un service de stérilisation, les DMR peuvent avoir de longues durées de pré désinfection. Notons que dans tous les deux cas, *i.e.* l'existence d'un poste de rinçage manuel ou non, un cycle de lavage commence avec un rinçage automatique. Alors, l'opération de rinçage manuel devient inutile dans le cas où les DMR sont pré désinfectés avec une durée assez courte. Un seul rinçage qui peut être appliqué par des laveurs avant chaque lavage, suffit pour l'élimination du liquide pré désinfectant sur les DMR. De plus, l'optimisation de la durée de pré désinfection peut donner lieu à l'opportunité de la suppression du rinçage manuel pour les services de stérilisation possédant un tel poste. Enfin, un quatrième objectif sera la minimisation du nombre de cycles de lavage lancés notamment pour le cas où tous les ensembles de DMR sont disponibles simultanément.

Nous présenterons nos études en quatre parties. La première partie du manuscrit donne des informations sur le fonctionnement des services de stérilisation ainsi que des informations détaillées sur chaque étape du processus de stérilisation. La partie 1 explique aussi notre approche de modélisation pour l'étape de lavage. La deuxième partie consiste à développer des méthodes d'optimisation offline pour la minimisation du C_{max}, la minimisation de la $\sum C_j$ et l'optimisation de la durée de pré désinfection des DMR. Ces études supposent que toutes les informations concernant l'arrivée des DMR dans le service de stérilisation sont connues à l'avance. Le but de cette partie est de montrer combien une modélisation offline peut aider à améliorer la performance de l'étape de lavage. La partie 3 est consacrée à des études onlines (et semi-onlines). Nous allons d'abord proposer un algorithme online et un algorithme semi-online, puis les insérer dans un modèle de simulation afin de tester leur impact sur tout le processus de stérilisation. Nous terminerons notre étude avec une étude de cas, présentée dans la partie 4, qui vise à minimiser le nombre de cycle de lavage quand tous les DMR sont disponibles simultanément devant les laveurs. Ce cas correspond au fonctionnement d'un service de stérilisation externe.

~ PARTIE 1 ~

Stérilisation Hospitalière

Dans cette partie, nous allons d'abord expliquer le processus de stérilisation en présentant les différentes étapes qu'il contient. Ensuite, nous allons diriger notre attention sur l'étape de lavage d'où provient la problématique traitée dans cette thèse. Puisque la modélisation de l'étape de lavage a une relation forte avec le problème d'ordonnancement par batch, nous allons expliquer comment nous modélisons l'étape de lavage, et quelles pistes de travail nous avons obtenues.

Le chapitre 1 est consacré aux aspects économiques, liés au secteur de la santé, et à la description du processus de stérilisation, en mentionnant son importance dans les établissements de santé. Le chapitre 2 est ensuite consacré au problème qui nous intéresse plus particulièrement dans cette thèse : l'optimisation du chargement des ressources de lavage. Nous commençons par présenter l'étape de lavage et expliquer pourquoi cette étape nécessite des méthodes d'optimisation. Après avoir fait le rapprochement avec plusieurs problèmes mathématiques (comme le bin packing et l'ordonnancement) liés à cette optimisation, une classification sera faite pour montrer les problèmes déjà étudiés dans la littérature. Cette classification nous permettra de voir les différentes pistes de travail pour lesquelles nous pourrions développer des méthodes d'optimisation. Nous terminons cette partie en précisant l'objectif de notre étude.

Chapitre 1 : Stérilisation dans les établissements hospitaliers

1.1. Aspects économiques

En France, comme dans la plupart des pays occidentaux, on observe un accroissement des dépenses de santé [4]. Ainsi, entre 1970 et 2000, le pourcentage des dépenses de santé dans le produit intérieur brut est passé de 7 à 13% aux Etats-Unis, de 5 à 8% en Italie et de 4,5 à 7% au Royaume-Uni. D'après les études statistiques menées en 2009 pour déterminer les dépenses de santé en France, le montant des dépenses courantes s'élève à 223,1 milliards d'euros, en progression de 4 % par rapport à 2008 [40]. La France consacre ainsi 11,7% de son produit intérieur brut à la santé, alors que ce pourcentage atteint 16% aux Etats-Unis en 2009.

Les raisons des augmentations de dépenses en santé peuvent s'expliquer par 5 facteurs principaux [102] : « *les progrès rapides de médecine et l'augmentation de la complexité des soins de santé au cours des 50 dernières années ; une population de patients qui a essentiellement besoin de soins chroniques, plutôt que de soins aigus ; l'absence d'incitation à améliorer la productivité des services de soins ; un sous-investissement persistant du secteur hospitalier sur la technologie d'information et de communication ; une application limitée des outils du génie industriel à la gestion des services de soins.* » Ainsi, améliorer la gestion dans les hôpitaux peut aider à diminuer les coûts organisationnels, parmi lesquels nous pouvons compter l'achat et la stérilisation des dispositifs médicaux.

L'aspect managérial pour fournir des services de santé aux patients dans les établissements de santé est de plus en plus important depuis une dizaine d'année. Les hôpitaux veulent en effet réduire les coûts liés aux services proposés et améliorer les capitaux financiers tout en assurant une très bonne qualité dans leurs services [14]. De plus, le contexte économique présenté ci-dessus nous incite à chercher des causes et des solutions aux dysfonctionnements des organisations hospitalières pour améliorer la qualité du système de soins. Cette amélioration n'implique pas seulement l'amélioration de la qualité médicale et de l'efficience des soins, mais elle implique aussi une meilleure gestion des ressources hospitalières afin de réduire les dépenses de santé. Les services de

stérilisation des hôpitaux sont évidemment des départements où nous pouvons apporter des améliorations organisationnelles.

1.2. Importance de la stérilisation

La stérilisation hospitalière a pour but de réduire la densité des micro-organismes présents sur ou dans les objets utilisés pour les interventions chirurgicales et empêcher l'infection nosocomiale [151]. La stérilisation hospitalière a une longue histoire. Avant les années 1800, les médecins avaient la conscience des infections nosocomiales, mais ils ne pouvaient apporter aucune démonstration scientifique pour montrer leur existence. Pourtant, ils savaient, par expérience, que passer à la flamme les instruments chirurgicaux diminuait le risque d'inflammation avant leur utilisation. Pasteur découvre l'existence des micro-organismes dans les années 1870-1880. Ces découvertes ont donné lieu aux débats sur le rôle des agents vivants dans la fermentation. À la séance du 5 janvier 1874 de l'Académie des Sciences, Pasteur avait déjà dit : « *Si j'avais l'honneur d'être chirurgien, jamais je n'introduirais dans le corps de l'homme un instrument quelconque sans l'avoir fait passer dans l'eau bouillante et mieux encore dans la flamme tout aussitôt avant l'opération, et refroidir rapidement.* » [153].

La naissance du mot nosocomiale remonte aux années 1960. A cette époque là, les médecins découvrent que les microorganismes peuvent se retrouver partout dans un hôpital, et pas seulement dans les blocs opératoires ou sur les instruments chirurgicaux. Alors, l'expression « infection nosocomiale » désigne les infections qui viennent de l'hôpital même. Parmi les agents infectieux, nous pouvons compter les bactéries, les levures et champignons, les virus et les prions. De plus, la croissance de ces agents est très rapide. Alors, la stérilité des dispositifs médicaux joue un rôle très important pour empêcher l'infection nosocomiale. Ces instruments sont toujours en contact physique avec les patients. De plus, la plupart de ces instruments sont utilisés plusieurs fois et une mauvaise stérilisation peut générer un grand risque d'infection nosocomiale. Ainsi, ces raisons ont donné lieu à la naissance de la stérilisation des dispositifs médicaux.

1.3. Processus de stérilisation

La stérilisation des dispositifs médicaux (ou DM) est primordiale pour la lutte contre l'infection nosocomiale. La définition des dispositifs médicaux est très large. Le Code de la Santé publique définit les dispositifs médicaux comme suit [152]:
« On entend par Dispositif Médical tout instrument, appareil, équipement, matière, produit, à l'exception des produits d'origine humaine, ou autre article

utilisé seul ou en association, y compris les accessoires et logiciels intervenant dans son fonctionnement, destiné par le fabricant à être utilisé chez l'homme à des fins médicales et dont l'action principale voulue n'est pas obtenue par des moyens pharmacologiques ou immunologiques ni par métabolisme, mais dont la fonction peut être assistée par de tels moyens. »

La définition ci-dessus couvre tout équipement utilisé à des fins médicales. Même s'il est possible de donner une définition générale pour les DM on a une caractérisation différente pour certains produits en fonction de leur utilité, comme par exemple les dispositifs fabriqués sur mesure, implantables, etc. On peut les catégoriser comme suit [117] :

- *Les DM implantables actifs : Ce sont les DM qui sont montés dans le corps humain. Les implants dentaires et les stimulateurs cardiaques sont dans ce groupe.*
- *Les DM fabriqués sur mesure : Ce sont les DM qui ne sont pas fabriqués en série. Ils sont demandés spécialement par les médecins pour des patients si nécessaire. Par exemple, les prothèses mammaires externes sont dans cette catégorie.*
- *Les accessoires des DM : Les DM de cette catégorie sont traités comme des dispositifs à part entière. Un exemple serait les produits d'entretien de lentilles de contact sans revendication d'action thérapeutique.*
- *DM de diagnostic in vitro : Ce sont les DM qui sont utilisés pour déterminer la sécurité d'un prélèvement d'éléments du corps humain ou sa compatibilité avec un receveur potentiel, contrôler des mesures thérapeutiques, fournir une information concernant un état physiologique ou pathologique*

Nous nous intéressons ici à tout type de dispositif médical qui nécessite un processus de stérilisation afin de pouvoir être réutilisés après utilisation. On les nommera donc « dispositifs médicaux réutilisables » (ou DMR). Notons que d'une part il existe un très grand nombre de références de DMR dans un hôpital et d'autre part les DMR utilisés dans les interventions chirurgicales peuvent être nombreux et de différents types. Les figures ci-dessous peuvent nous aider à mieux comprendre la quantité et la diversité de DMR utilisés dans des interventions chirurgicales.

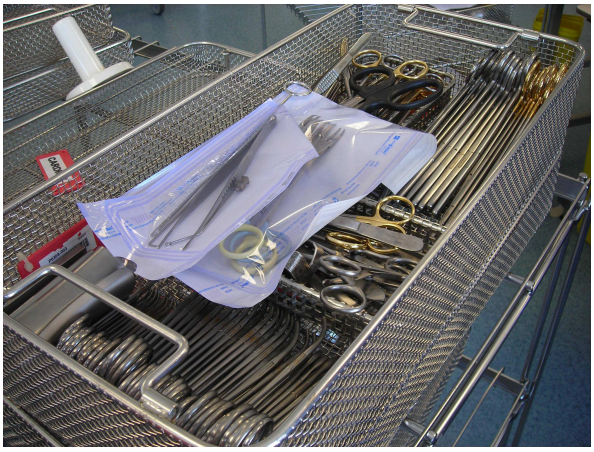

Figure 1.1. Ciseaux et pinces utilisés pour une seule intervention chirurgicale

Figure 1.2. Différents types de dispositifs utilisés pour une intervention chirurgicale

Il est indiqué par les « Bonnes Pratiques de Pharmacie Hospitalière » que les DMR sont considérés comme stériles si la probabilité théorique qu'un micro-organisme viable soit présent est inférieure ou égale à 1 pour 10^6 [98]. En effet, la stérilisation des DMR est réglementée par des normes AFNOR (Association Française de Normalisation) [126]. AFNOR oblige les hôpitaux à avoir un système de stérilisation centrale ou à externaliser leur activité de stérilisation. Dans le cas d'un service de stérilisation centrale dans un hôpital, tous les DMR utilisés dans les interventions chirurgicales sont envoyés au service central. Si un hôpital externalise son activité de stérilisation, c'est un sous-traitant qui exerce l'activité de stérilisation et il n'y a pas de service de stérilisation au sein de l'hôpital [118].

Le processus de stérilisation est composé des étapes suivantes : pré désinfection, rinçage, lavage, vérification, conditionnement, stérilisation. Entre

chaque stérilisation, les DMR sont stockés et réutilisés. Le processus de stérilisation est donc un processus cyclique (*cf.* figure 1.3).

Il peut y avoir des différences en pratique sur les modalités d'application des étapes de stérilisation [102]. Quand un service de stérilisation fait le trempage et le rinçage des DMR au niveau des blocs, un autre service peut faire commencer la pré désinfection au niveau de blocs et la termine par un rinçage manuel après l'arrivée des DMR au service de stérilisation. Ou encore, un service de stérilisation peut éviter le rinçage manuel et faire seulement le rinçage automatique dans les laveurs. Expliquons maintenant en détail ces différentes étapes.

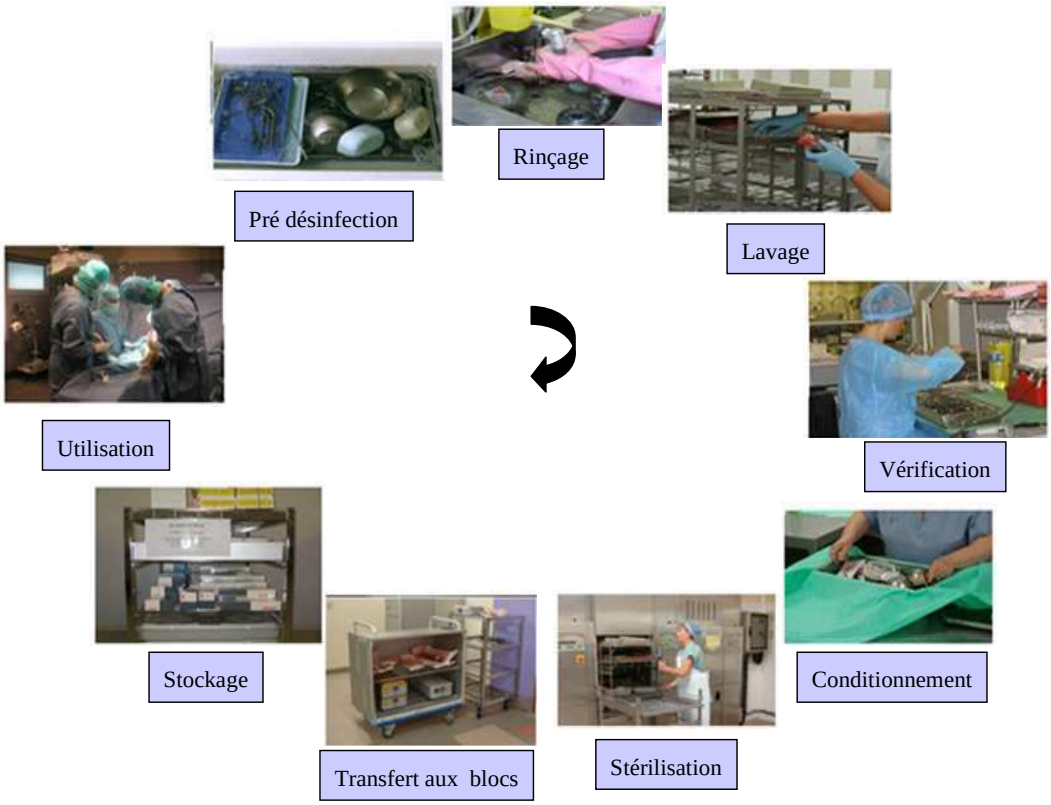

Figure 1.3. Processus de stérilisation et flux de DMR entre les blocs et le service de stérilisation

Pré désinfection (ou trempage) : Le but de cette étape est d'éviter le dessèchement des souillures et de faciliter le décrochage ultérieur des souillures lors du nettoyage. De cette façon, la pré désinfection aide à diminuer la population de micro-organismes avant les étapes suivantes. La pré désinfection a également pour but de protéger le personnel lors de la manipulation des instruments. Elle permet aussi d'éviter la contamination de l'environnement.

La pré désinfection commence dans les blocs opératoires. Les DMR sont immergés dans un liquide pré désinfectant et transférés au service de stérilisation. D'après nos investigations, la durée minimale de pré désinfection est de 15 minutes alors que 50 minutes est généralement considéré être la limite supérieure, et 20 minutes la durée idéale. Les DMR risquent d'être abimés par le liquide à partir de 50 minutes.

Rinçage : Les DMR doivent être rincés avant d'être lavés. Le but de cette opération est d'éviter le risque d'interférence entre le produit pré désinfectant et le produit de lavage. Idéalement, ce rinçage devrait être effectué avant que la durée de pré désinfection ne dépasse la limite supérieure. Le rinçage peut être réalisé, soit manuellement, soit automatiquement dans les laveurs, ou encore les deux l'un après l'autre [36]. Dans le cas du rinçage manuel, les DMR sont rincés dans les blocs opératoires, ou au service de stérilisation après leur arrivée. Ensuite, ils attendent le lavage dans un stock. Par contre, s'il n'y a que le rinçage automatique dans un service de stérilisation, la durée d'attente des DMR dans le stock de lavage devient un sujet important à traiter pour que la durée de pré désinfection ne dépasse pas 50 minutes.

Lavage : Le lavage est une étape critique qui influence directement le résultat du processus de stérilisation. Le lavage est une action physico-chimique combinée à une action mécanique. Le but du lavage est d'enlever la salissure sur les DMR. A la fin du lavage, les DMR doivent être propres, fonctionnels et aussi séchés pour le conditionnement. Les laveurs sont remplis par des paniers qui contiennent les DMR (*cf.* figure 1.4). La capacité des laveurs peut varier (de 4 à 12 paniers DIN[1]) [36]. D'après nos investigations, les DMR ont généralement la même durée de lavage. Il peut y avoir différents programmes de lavage dans les laveurs, mais la différence en durée de lavage entre ces programmes est quasiment nulle.

Vérification : Cette étape est en fait le premier pas de l'étape de conditionnement. Le but est de voir si tous les DMR sont en bon état, c'est-à-dire s'ils sont bien lavés, s'il n'y a pas de DMR cassés, abimés, etc. On vérifie également la constitution des boîtes d'un ensemble de DMR.

[1] Panier DIN : Unité de volume pour les laveurs automatiques. Un panier DIN correspond à peu près à 3 dm^3.

Figure 1.4. Chargement d'un laveur automatique

Conditionnement : Les DMR sont conditionnés en sachet ou conteneur (ou boîte). Au poste de conditionnement, les boîtes et sachets sont recomposés. Les ensembles de DMR sont en fait un ensemble de boîtes et de sachets de DMR. Le conditionnement assure aussi la stérilité des DMR jusqu'à l'utilisation. Les boîtes et sachets de conditionnement constituent des barrières imperméables pour éviter l'entrée de tout micro-organisme. Par contre, ces barrières permettent bien sûr la pénétration de l'agent de stérilisation dans l'étape suivante. Les figures 1.5 et 1.6 montrent quelques exemples de boîtes, et la figure 1.7 montre un exemple de sachet.

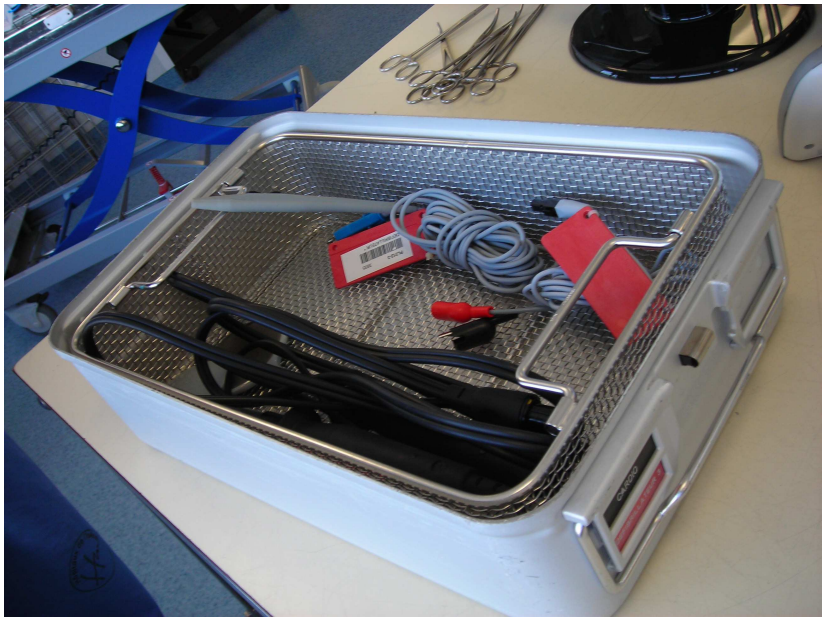

Figure 1.5. Conditionnement de DMR dans une boite

Figure 1.6. Quelques boites de taille différente

Une fois que les tous les DMR d'une boite sont mis dedans, la boite est fermée et envoyée aux autoclaves. La figure 1.8 montre deux boites (une grande et une petite) fermées et envoyées vers le poste de stérilisation.

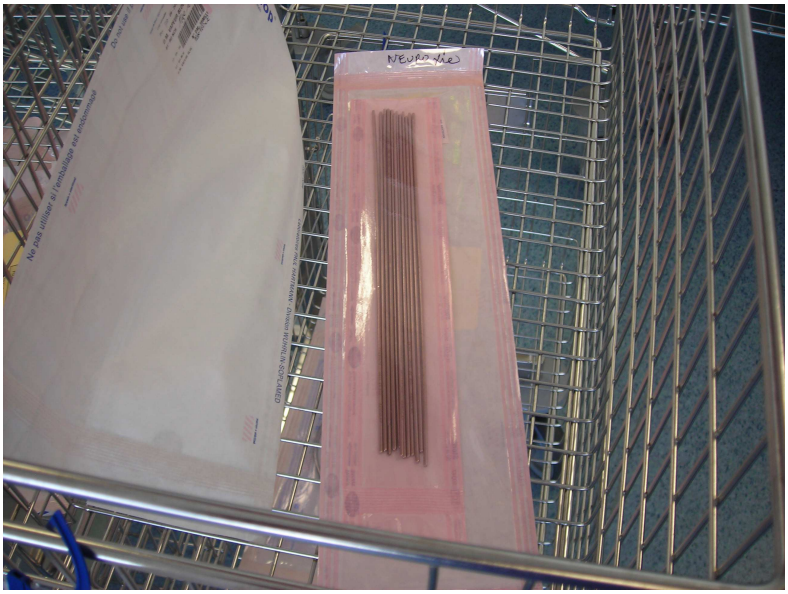

Figure 1.7. Exemple de sachet

Stérilisation : La stérilisation se fait via des machines qui s'appellent des « autoclaves ». Les autoclaves appliquent de la vapeur d'eau aux DMR conditionnés. Le but est d'éliminer tous les germes ou les contaminants en mettant l'objet à une température de 134 °C plus une surpression de 2,2 bars pendant au moins 18 min, et cela sur la totalité de la surface à stériliser. Le réchauffement et la surpression sont appliqués plusieurs fois dans un cycle de stérilisation. La durée d'un cycle dans un autoclave peut varier d'un autoclave à l'autre (de 1 à 2 heures).

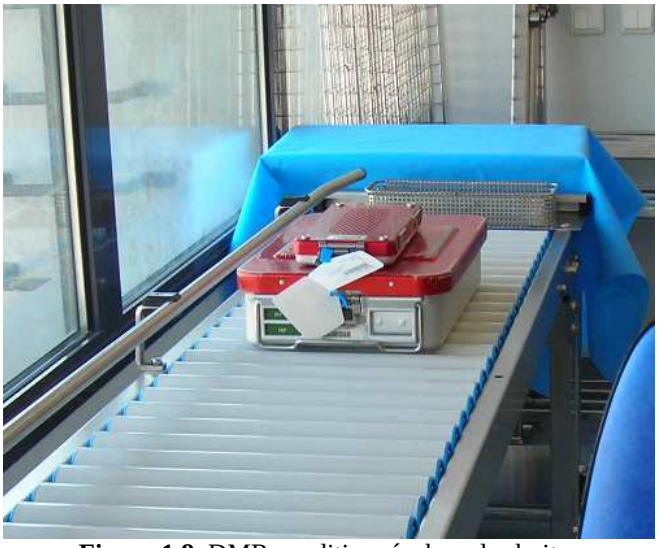

Figure 1.8. DMR conditionnés dans des boites

Transfert aux blocs : Une fois que les DMR sont stérilisés, il s'agit maintenant de les transférer aux blocs opératoires. Le transfert se fait via des armoires ou des chariots fermés. Notons que les armoires et les chariots sont également nettoyés avant chaque utilisation. Dès leur arrivée au service de stérilisation, ils sont lavés et ensuite mis en attente de l'arrivée des DMR stériles dans un milieu propre.

Stockage : Les DMR doivent être stockés à l'abri de l'humidité et de la lumière solaire directe.

Après ces étapes, les DMR sont prêts à être réutilisés.

1.4. Etat de l'art sur la stérilisation et quelques exemples d'application des méthodes de la recherche opérationnelle dans le domaine de la santé

La plupart des travaux sur la stérilisation hospitalière se concentrent plutôt sur les techniques de stérilisation à utiliser et les règles à respecter ([35], [117], [121], [122]). Nous avons trouvé très peu de travaux qui étudient l'amélioration du service de stérilisation. Ce sont ceux qui vont nous intéresser plus particulièrement, et que nous présentons ci-dessous.

Bernard et Lacroix [9] établissent une restructuration pour le service de stérilisation du CHU de Bordeaux. Cette restructuration consiste à centraliser la stérilisation conformément aux bonnes pratiques de pharmacie hospitalière [98]. La stérilisation de DMR dans des endroits différents nécessitait cette restructuration. Alors que l'hôpital possédait une unité centrale de stérilisation où se trouvaient seulement les autoclaves, les opérations de péristérilisation, à savoir le lavage et le conditionnement, étaient éclatées à différents niveaux de l'hôpital.

Certains hôpitaux ne possèdent pas un service de stérilisation. Ils ont recours à un centre externe pour la stérilisation de leur DMR. Ce cas est appelé « externalisation de la stérilisation ». Dans son travail, Bardet [8] explique les avantages et les inconvénients de l'externalisation pour les hôpitaux.

L'enquête EESS [36] nous a permis d'avoir une meilleure connaissance des hôpitaux de la région Rhône-Alpes. En effet, EESS est une enquête de recherche mené en 2007 par une équipe de chercheurs provenant de trois laboratoires rhônalpins : le LASPI (Laboratoire d'Analyse des Signaux et des Processus Industriels), G-SCOP (Grenoble – Sciences pour la Conception, l'Optimisation et la Production), GIPSA-Lab (Grenoble Image Parole Signal Automatique) et par des membres du CERCLH (CEntre de Recherche et de Compétences en Logistique Hospitalière). L'enquête comporte une centaine de

questions portant sur l'activité et l'organisation actuelles des services. Sur les 75 hôpitaux qui ont été interrogés, 39 ont répondu à cette enquête. Le projet EESS a permis d'identifier les différences concernant le fonctionnement de la stérilisation auprès de ces hôpitaux (comme par exemple, l'application du rinçage manuel, la stratégie de lavage, la durée désirée pour la pré désinfection, etc.). Reymondon et al. [118] dressent un état des lieux de l'existant grâce à cette enquête.

Après nous être intéressés aux analyses de l'existant, nous allons maintenant étudier l'application de méthodes de recherche opérationnelle dans le domaine de la stérilisation hospitalière.

Reymondon et al [119] se concentrent sur la minimisation du coût de stockage des DMR après le processus de stérilisation. Puisqu'un DMR peut servir à plusieurs types d'intervention, il est possible de profiter de cette utilité multiple. Les auteurs proposent une stratégie de stockage permettant de ne pas stocker simultanément un trop grand nombre d'ensembles de DMR de type identique.

Albert et al. [1] ont fait une analyse sur le compromis entre la minimisation du temps d'attente des DMR à l'étape de lavage et du nombre de cycles de lavage lancés. Pour ce problème, ils proposent une heuristique. Les tests numériques montrent que la minimisation de la durée d'attente de DMR est en relation inverse avec la maximisation de la charge des laveurs.

Une autre étude qui concerne l'organisation dans les services de stérilisation est la thèse de doctorat de Ngo Cong [102]. Dans sa thèse, il propose un modèle générique de simulation qui montre les flux physiques dans un service de stérilisation (Dans notre étude, nous allons nous inspirer de ce modèle générique afin de construire nos modèles de simulation dans les chapitres suivants.). Il modélise aussi le modèle générique d'un service de stérilisation par réseau de files d'attente.

Notons que l'arrivée des DMR au service de stérilisation dépend principalement de l'emploi du temps des interventions chirurgicales. Di Mascolo et al. [31] simulent un service de stérilisation afin de voir l'intérêt du lissage de l'arrivée de DMR au service. Dans ce but, ils proposent que les DMR soient ramassés régulièrement auprès des blocs opératoires. Les résultats montrent que le lissage de l'arrivée des DMR permet de minimiser les durées de pré désinfection ainsi que l'attente de DMR dans le stock de lavage.

L'application des méthodes de recherche opérationnelle en milieu hospitalier n'est pas bien sûr limitée aux services de stérilisation. Parmi quelques sujets de recherche en Génie Industriel et Recherche opérationnelle en milieu hospitalier, nous pouvons citer l'ordonnancement des interventions dans les blocs

opératoires, l'allocation de lits dans les unités de soin, l'optimisation du routage des véhicules et des patients, etc.

Parmi quelques exemples représentatifs pour le problème d'ordonnancement des blocs opératoires, nous pouvons citer le travail de Guinet et Chaabane [52] où les auteurs présentent une heuristique constructive pour affecter les patients aux jours d'interventions et aux blocs opératoires. Fei et al. [38] présentent un algorithme de « branch and price » pour le planning des interventions, qui peuvent avoir des dates limites, afin de minimiser le coût des interventions. La littérature sur la planification des blocs opératoires est très vaste. Pour plus d'information sur ce sujet, le lecteur peut se référer à [14]. Dans cet article, les auteurs proposent une classification des différents travaux pour montrer les différentes techniques appliquées ainsi que les mesures de performance.

Un des objectifs de l'application des méthodes d'optimisation dans les chaines logistiques hospitalières est d'améliorer le flux de matériel tout en diminuant les coûts de transport [70]. Kergosien et al. [66] considèrent le problème dynamique d'optimisation de tournées d'ambulances du complexe hospitalier de Tour (France). Dans ce complexe hospitalier, une station centrale des ambulances doit planifier le transport entre les différentes unités de soin. Outre les flux physiques, les flux d'informations entre les différents acteurs d'une chaine logistique ont également été étudiés ([39], [112]).

Comme autres applications de la recherche opérationnelle à la santé, nous pouvons citer l'hospitalisation à domicile et l'allocation de lits dans les établissements de soin. Un exemple pour ce premier serait la livraison de produits pour la chimio-thérapie à domicile [15]. Pour la problématique d'allocation de lits, Marcon et al. [92] utilisent un modèle de simulation afin de déterminer le nombre de lits nécessaires dans une unité de salle de réveil.

Les articles cités ci-dessus ne représentent pas une analyse exhaustive pour l'application de la recherche opérationnelle et du génie industriel à la santé, mais des exemples assez récents pour montrer le spectre des applications. Pour plus d'informations sur les méthodes de recherche opérationnelle et du génie industriel appliquées au service de santé, le lecteur peut se référer aux états de l'art [11], [59], [82].

1.5. Conclusion

Depuis une dizaine d'années, l'utilisation des méthodes d'ingénierie et de recherche opérationnelle dans différents services des hôpitaux se développent.

Parmi les différents endroits où des méthodes d'optimisation sont appliquées, nous pouvons compter la préparation de plannings pour les interventions chirurgicales, l'allocation de lits aux patients, l'hospitalisation à domicile, etc. Le domaine qui nous intéresse dans cette étude est la stérilisation des dispositifs médicaux réutilisables (DMR).

Dans cette thèse, nous considérons le service de stérilisation dans son aspect « système de production de DMR stériles ». Les DMR, comme leur nom l'indique, sont utilisés plusieurs fois. Ils sont stérilisés entre chaque utilisation. Le processus pour avoir des DMR stériles est assez compliqué et donc il faut beaucoup d'attention pour assurer une bonne stérilité. Un seul exemple peut nous aider à mieux comprendre son importance : suite à une mauvaise stérilisation des dispositifs médicaux en 1997 dans une clinique parisienne, soixante et un patients de la clinique ont subit des infections nosocomiales [71]. De plus, le processus de stérilisation est composé de plusieurs opérations, et comme tous les systèmes de production, la stérilisation hospitalière a besoin d'améliorations pour certaines étapes.

Dans ce chapitre, nous avons fait une introduction au sujet de la stérilisation. Après avoir expliqué les différentes étapes du processus de stérilisation, nous avons parlé des différents domaines de santé où les méthodes de recherche d'opérationnelle sont appliquées. Dans le chapitre suivant, nous allons introduire l'étape de lavage des services de stérilisation d'où provient notre problématique.

Chapitre 2 : Présentation et modélisation de l'étape de lavage

2.1. Introduction

Ce chapitre est consacré à la présentation de l'étape de lavage. Nous parlerons de cette étape en précisant les différentes opérations qui sont effectuées jusqu'à la fin du lavage des DMR. D'après les réponses des hôpitaux qui ont participé au questionnaire EESS [36], l'étape de lavage est généralement un goulet pour le processus de stérilisation. Puisque notre objectif est d'améliorer la performance du processus de stérilisation en proposant différents stratégies de chargement des laveurs, nous allons ensuite expliquer quelles approches de modélisation qui peuvent s'appliquer à l'étape de lavage. Le faire nous permettra de déterminer différentes pistes de travail parmi lesquelles nous choisissons les problématiques traités ici.

2.2. Présentation de l'étape de lavage

Rappelons que la stérilisation des DMR commence dans les blocs opératoires, juste après les interventions chirurgicales. La première étape de la stérilisation est la pré désinfection (ou trempage). Une fois qu'une intervention est terminée, tous les DMR ayant servi pour cette intervention sont mis dans un liquide pré désinfectant. Ensuite, en fonction de l'organisation de l'hôpital, les DMR peuvent être rincés dans les blocs opératoires puis envoyés au service de stérilisation, ou envoyés au service de stérilisation pendant la pré désinfection. Il peut y avoir deux types d'acheminement pour les DMR : soit un personnel du bloc ramène au service de stérilisation à la fin d'une intervention chirurgicale, soit un personnel du service de stérilisation visite les blocs opératoires régulièrement pour chercher des DMR utilisés. Les DMR non rincés au niveau des blocs peuvent soit être rincés manuellement, puis mis en attente pour le lavage, soit mis dans le stock de lavage pour attendre le rinçage dans les laveurs avant le lavage. Enfin, au fur et à mesure de l'arrivée des DMR les laveurs sont remplis avec ces DMR. Nous montrons l'acheminement de DMR sur la figure 2.1.

Comme montré sur la figure 2.1, un hôpital (ou une clinique) peut posséder plusieurs blocs opératoires. L'équipe d'un bloc opératoire est composée

de deux groupes [129] : un premier groupe anesthésique composé de médecins anesthésistes et d'infirmiers anesthésistes, un deuxième groupe chirurgical composé de chirurgiens et d'infirmiers du bloc opératoire. De plus, d'autres intervenants peuvent travailler au bloc opératoire : radiologues, gastro-entérologues, pneumologues, aide-soignant, brancardier, etc. Bien évidemment, plusieurs types d'interventions chirurgicales peuvent être réalisés dans les blocs opératoires. Parmi les différents types de chirurgies, nous pouvons compter la chirurgie orthopédique, la chirurgie vasculaire, l'urologie, l'obstétrique, la pneumochirurgie, etc.

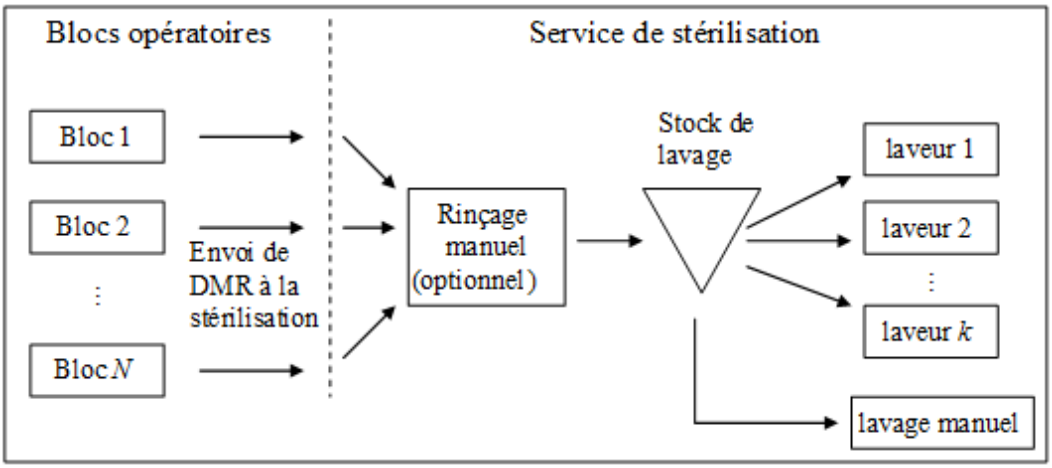

Figure 2.1. Acheminement de DMR des blocs opératoires vers le service de stérilisation

Tous les DMR utilisés pour une intervention chirurgicale constituent l'ensemble de DMR de cette intervention. Evidemment, chaque intervention nécessite différents types de DMR en nombres différents. Voire, pour une même intervention chirurgicale, le choix de DMR de deux médecins différents peut varier. Alors, la taille des ensembles de DMR peut beaucoup varier d'une intervention à l'autre. Il est possible de laver plusieurs ensembles de DMR simultanément dans un seul laveur. Mais, il n'est généralement pas permis de couper les ensembles de DMR et de les laver dans deux laveurs différents. La raison de ce choix relève de la multiplicité des références de DMR. Les DMR appartenant aux différents ensembles peuvent se mélanger, ce qui complique la traçabilité des DMR. Si un ensemble de DMR est coupé et traité dans deux ou trois laveurs différents, les étapes suivantes (vérification et conditionnement) risquent d'être compliquées et cette stratégie risque d'être une source importante d'erreur.

Bien entendu, les interventions chirurgicales ne commencent pas et ne se terminent pas forcément en même temps. Ainsi les ensembles de DMR n'arrivent pas au service de stérilisation en même temps. Si un ensemble de DMR est

présent dans le stock de lavage et s'il est suffisamment trempé dans le liquide pré désinfectant, il est disponible pour être chargé dans un laveur. La décision à prendre pour le chargement des laveurs concerne l'instant de lancement du cycle de lavage, et le contenu du laveur lors de ce cycle. Il est alors possible de modéliser le problème de chargement de laveurs comme un problème d'ordonnancement par batch.

La modélisation de l'étape de lavage comme un problème d'ordonnancement est comme suit : nous traitons les laveurs de l'étape de lavage comme des machines identiques, les ensembles de DMR comme des tâches à exécuter et le stock de lavage comme le stock d'en-cours. Les machines et les tâches ont les caractéristiques suivantes :

- Les tâches peuvent avoir des tailles différentes,
- Les tâches arrivent au service de stérilisation à différents moments dans une journée,
- Les tâches ont toutes la même durée d'exécution. Les machines sont identiques,
- Il est possible de créer un batch avec plusieurs tâches. Toutes les tâches d'un batch sont exécutées en même temps dans la machine à laquelle le batch est affecté.

Maintenant, nous allons expliquer la stratégie généralement appliquée à l'étape de lavage des services de stérilisation, tout en expliquant les inconvénients de cette stratégie.

2.3. Stratégie naturelle pour le chargement des laveurs

D'après l'enquête EESS, les services de stérilisation essaient soit de maximiser la charge des laveurs, soit de minimiser l'attente des ensembles de DMR avant le lavage. Pour cela, une stratégie simple est souvent appliquée. Un seuil de remplissage est déterminé. Ensuite, en fonction de ce seuil, un batch est créé avec des ensembles de DMR au fur et à mesure de leur arrivée.

Quand un ensemble de DMR entre dans le stock de lavage, s'il y a un laveur disponible, les paniers de cet ensemble sont placés dans les étagères d'un laveur. A chaque arrivée d'un nouvel ensemble de DMR, les paniers sont placés dans les étagères. Dès que le seuil de remplissage est atteint, un cycle de lavage est lancé. Bien entendu, s'il y a encore des DMR disponibles, le seuil de remplissage peut être dépassé en respectant la capacité du laveur. S'il n'y a pas de laveur disponible lorsque les DMR arrivent, ils sont mis dans un stock d'attente.

Nous pouvons montrer comme exemple le service de stérilisation du Centre Hospitalier Privé Saint Martin de Caen (CHPSM) où ce genre de stratégie est appliqué pour le remplissage des laveurs. Dans ce service, ils essaient de maximiser la charge des laveurs. Donc, la condition de fermer un batch est qu'il y ait un ensemble de DMR qui n'entre pas complètement dans un batch actuel. C'est-à-dire qu'avec une nouvelle arrivée d'un ensemble, on vérifie s'il y a de la place disponible dans le batch qui est en train d'être rempli. Si ce batch ne contient pas assez d'espace libre pour placer tous les paniers de l'ensemble qui vient d'arriver, le batch est fermé. Un cycle de lavage est lancé et un nouveau batch est créé avec le nouvel ensemble. Puisque les batch sont créés avec des ensembles de DMR successifs, la formation des batch correspond à une stratégie de type *FIFO*[2]. Une raison pour appliquer cette stratégie est le manque d'interaction entre les blocs opératoires et le service de stérilisation. Les arrivées des ensembles de DMR au service ne sont pas connues à l'avance. La stratégie appliquée au service de stérilisation de CHPSM correspond à du *FIFO online*.

Notons que l'application de *FIFO online* peut provoquer une longue durée d'attente pour les DMR dans le stock de lavage. Alors, nous pouvons nous poser la question suivante : est-ce qu'il pourrait être possible de remplacer *FIFO online* par d'autres stratégies afin d'améliorer l'organisation de l'étape de lavage ? Pour ce but, une première réflexion serait de chercher dans la littérature pour voir s'il existe des méthodes qui peuvent être appliquées au problème du chargement des laveurs. Mais d'abord, nous avons besoin de bien définir notre problème du chargement des laveurs en identifiant les hypothèses différentes. La section suivante a pour but de faire une classification des problèmes qui peuvent être identifiés en considérant des hypothèses diverses. De cette façon, nous aurons aussi déterminé les différentes pistes sur lesquelles nous pouvons travailler.

2.4. Classifications de problèmes concernant l'étape de lavage

Il est possible de définir plusieurs problèmes mathématiques afin de proposer des méthodes d'optimisation pour l'étape de lavage en prenant en compte différentes hypothèses. Nous avons vu que chaque service de stérilisation peut avoir sa propre organisation (rinçage manuel ou non, arrivée des DMR en cours de pré désinfection ou déjà pré désinfectés, etc.). Maintenant, nous allons présenter les éléments de l'étape de lavage où des différentes hypothèses peuvent s'appliquer. Combiner les différentes hypothèses nous amènera à des problèmes différents. De cette façon, nous pourrons scruter la littérature pour déterminer les problèmes étudiés et ceux restant à traiter.

---

[2] FIFO : first in first out

2.4.1. Cas possibles

Nous avons vu, en particulier dans l'enquête EESS, que plusieurs choix sont possibles pour l'organisation d'un service de stérilisation. Nous regroupons ici les différentes hypothèses possibles en deux groupes principaux : hypothèses sur le service de stérilisation, hypothèses concernant les DMR.
Les hypothèses sur le service de stérilisation comprennent les points suivants:

- existence d'un poste de rinçage manuel,
- capacité du stock de lavage (limitée ou pouvant être considérée comme infinie),
- nombre de laveurs,
- laveurs identiques ou non.

Expliquons les points ci-dessus un par un.

Existence d'un poste de rinçage manuel

Les services de stérilisation peuvent posséder, ou non, un poste de rinçage manuel. Donc, nous pouvons considérer les deux cas pour construire nos hypothèses.

Capacité du stock du lavage

En général, la capacité du stock du lavage est assez grande et donc il y a assez d'espace pour les DMR attendant le lavage. Mais lors d'une visite des blocs ambulatoires d'un hôpital à Leuven/Belgique (visite organisée par les organisateurs de la conférence ORAHS'09), nous avons remarqué que le service de stérilisation responsable de la stérilisation des DMR des blocs ambulatoires avait très peu d'espace pour l'attente des DMR. Alors, nous pouvons considérer deux cas pour la capacité du stock du lavage : stock de capacité limitée et illimitée.

Nombre de laveurs

Chaque service de stérilisation possède au moins un laveur. Donc, on distingue deux cas : le nombre de laveurs est un ou plus qu'un.

Laveurs identiques ou non

Lors de nos visites, nous avons observé que les laveurs d'un service étaient souvent du même type. La seule exception que nous avons rencontrée est le service de stérilisation du CHU Grenoble qui possède un laveur dont la capacité est 4 fois plus grande que les autres laveurs. Cependant, sa durée de cycle de lavage est la même que les autres machines. Donc, nous pouvons considérer encore deux cas pour les laveurs : laveurs identiques ou non-identiques au niveau de la capacité.

Les hypothèses sur les ensembles de DMR comprennent les points suivants:

- tailles différentes ou identiques
- dates d'arrivée au service stérilisation différentes ou date d'arrivée unique pour tous les ensembles de DMR
- pré désinfection réalisée complètement dans les blocs opératoires ou non

Nous détaillons ces hypothèses ci-dessous.

Tailles différentes ou identiques

En effet, la taille des ensembles de DMR peut beaucoup varier. Dans les hôpitaux, il est difficile d'avoir des ensembles de DMR dont les tailles sont toujours égales. Mais, la considération des tailles identiques nous amènera à travailler sur des aspects théoriques dans les sections suivantes.

Dates d'arrivée au service de stérilisation différentes ou date d'arrivée unique pour tous les ensembles de DMR

Ces deux cas sont tout à fait possibles. Considérons un service de stérilisation d'un hôpital qui reçoit des DMR des blocs opératoires pendant toute la journée. Bien entendu, puisque les interventions chirurgicales ne commencent pas toutes en même temps et peuvent avoir des durées différentes, les ensembles de DMR arriveront dans le service à différents instants dans une journée. Considérons maintenant un hôpital dont les DMR sont stérilisés par un service de stérilisation externe. Dans ce cas, tous les DMR de l'hôpital sont envoyés en même temps à ce service et donc ils ont une date unique pour leur arrivée au service. On peut aussi imaginer que les DMR arrivent tout au long de la journée, mais qu'on ne réalise la stérilisation qu'à des instants fixés, comme par exemple à midi et le soir.

Pré désinfection réalisée complètement dans les blocs opératoires ou non

Il y a certains hôpitaux dont l'organisation pour la pré désinfection est la suivante : les DMR sont plongés dans le liquide pré désinfectant. Avant qu'ils soient envoyés au service de stérilisation, les DMR sont aussi rincés à la main au niveau des blocs opératoires. Ainsi, les DMR arrivent au service dans un état rincés. Mais il y a aussi d'autres hôpitaux où la pré désinfection des DMR commence au niveau des blocs opératoires et se termine au service de stérilisation. Dans ce cas, les DMR arrivent au service de stérilisation au cours de la pré désinfection. Ensuite, la pré désinfection se termine via un rinçage manuel ou automatique (ou les deux).

La figure 2.2 montre la combinaison de ces hypothèses.

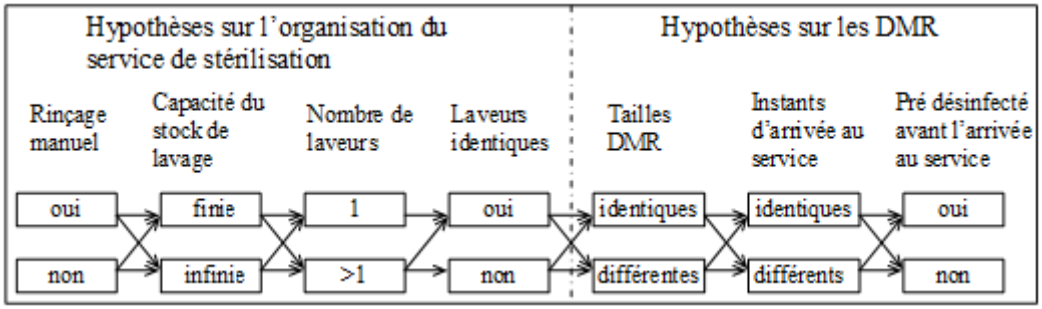

Figure 2.2. Combinaison d'hypothèses pour l'étape de lavage

Les combinaisons représentées dans la figure 2.2 donnent lieu à $3*2^5$ configurations possibles, soit 96 configurations. La considération de certaines hypothèses présente un intérêt purement théorique comme par exemple le cas où tous les ensembles de DMR ont la même taille, ou encore, le cas où le nombre de laveur est égal à 1. En réalité, les ensembles de DMR ont des tailles variées et d'après nos investigations, il y a au moins deux laveurs.

Une fois qu'un problème est construit en fonction des différentes hypothèses montrées sur la figure 2.2, différents objectifs en fonction du besoin de l'étape de lavage peuvent être considérés. Citons notamment :

- minimisation du nombre de cycles de lavage lancés,
- minimisation de la durée totale de lavage,
- minimisation du nombre d'ensembles de DMR en attente dans le stock de lavage,
- optimisation des durées de pré désinfection

Minimisation du nombre de cycles de lavage lancés

Ce critère est parmi les critères les plus appliqués dans les services de stérilisation. Le seuil de remplissage est déterminé à 100% pour les laveurs. Par contre, pour le cas où les ensembles de DMR arrivent au service de stérilisation aux différents moments d'une journée, ce critère provoque des attentes excessives pour les DMR dans le stock de lavage. Cependant, si tous les DMR sont disponibles en même temps (cas d'un service de stérilisation externe), ce critère ne minimise pas seulement le nombre de cycles de lavage lancés, mais aussi la durée totale de lavages quand tous les laveurs sont identiques.

Minimisation de la durée totale de lavage

Parmi les indicateurs de performances des services de stérilisation, nous pouvons considérer « le nombre de DMR stérilisés par jour ». Une bonne utilisation des ressources de lavage pourrait aider à augmenter le nombre de DMR

lavés et envoyés aux postes suivants. Ainsi, le nombre de DMR stérilisés par jour pourrait augmenter.

Minimisation du nombre d'ensembles de DMR en attente dans le stock de lavage

Pour certains services, l'étape de lavage est un goulet d'étranglement. Donc, l'attente des DMR dans le stock de lavage risque d'être assez longue. Il serait désirable de trouver des moyens pour diminuer le nombre de DMR attendant le lavage, ce qui diminuerait la durée d'attente dans le stock de lavage.

Optimiser les durées de pré désinfection

Comme déjà dit, la durée idéale de pré désinfection est autour de 20 minutes. Pour les services de stérilisation qui n'appliquent pas un rinçage manuel, une bonne utilisation des ressources de lavage est cruciale pour que les DMR aient des bonnes durées de pré désinfection. L'optimisation de la durée de pré désinfection est alors un bon critère sur lequel nous pouvons travailler.

Ces différentes configurations et différents objectifs peuvent donner lieu à différents problèmes d'ordonnancement, que nous listons par la suite.

2.4.2. Problèmes d'ordonnancement correspondants

Les différentes hypothèses que nous avons expliquées ci-dessus servent à déterminer quels types d'approche peuvent s'appliquer pour la modélisation et l'optimisation de l'étape de lavage. Nous avons maintenant assez d'informations sur le problème du chargement des laveurs. Nous avons des laveurs qui sont responsables du lavage des ensembles de DMR. Alors, nous avons tout simplement un problème d'ordonnancement où les laveurs sont des machines et les ensembles de DMR sont les tâches.

En ce qui concerne la conformité de ces hypothèses à la réalité, le questionnaire EESS [36] nous renseigne sur l'organisation des hôpitaux pour la pré désinfection des DMR. A l'étude de cette enquête, il apparait que tous les hôpitaux de la région Rhône-Alpes font commencer la pré désinfection au niveau des blocs opératoires et les DMR arrivent au service de stérilisation en train d'être pré désinfectés. La moitié des hôpitaux font un rinçage manuel alors que l'autre moitié ne le fait pas. Nous considérons pour notre étude qu'il n'y a pas de rinçage manuel (De plus, nous verrons dans le dernier chapitre de la partie 2 qu'il pourra être possible de supprimer le poste de rinçage manuel pour certains services qui l'appliquent tout en garantissant une bonne durée de pré désinfection pour les DMR).

Dans notre étude, on considère qu'il n'y a pas de rinçage manuel, que la capacité du stock de lavage est infinie (ou suffisamment grande pour qu'il n'y ait aucun problème de stockage) et que les DMR arrivent en cours de pré désinfection au service de stérilisation. De cette façon, nous réduisons le nombre de configurations différentes et il devient plus facile de faire une correspondance entre le problème du chargement des laveurs et un problème d'ordonnancement.

A partir de maintenant, nous traitons le problème du chargement des laveurs comme un problème d'ordonnancement. Comme montré sur la figure 2.3, nous pouvons représenter notre problème d'ordonnancement en trois parties : « machine », « tâches », « fonction objectif ». En nous inspirant de la notation de Graham et al. [51], nous allons donc présenter notre problème avec une notation à trois champs qui est de la forme suivante :
hypothèses sur les machines | hypothèses sur les tâches | fonction objectif.

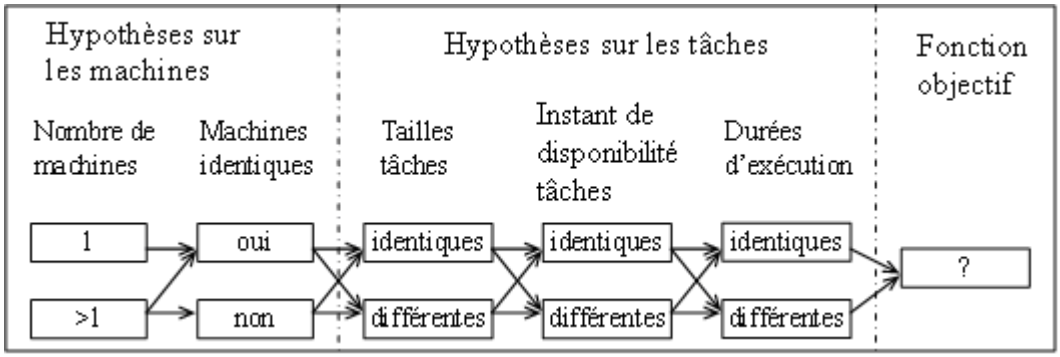

Figure 2.3. Combinaison d'hypothèses pour la version en ordonnancement du problème de chargement des laveurs

Une revue de la littérature plus conséquente sera faite dans les parties 2 et 3 de ce mémoire. Ici, nous donnons quelques notions préliminaires en ordonnancement. En ce qui concerne les hypothèses sur les machines, nous pouvons avoir une ou plusieurs machines dans un problème étudié. S'il y a plus d'une machine, ces machines peuvent être identiques, uniformes ou indépendantes. Les machines identiques sont des machines pour lesquelles les durées d'exécution des tâches ne changent pas en fonction des machines sur lesquelles elles sont traitées. Pour les machines uniformes, la durée de traitement des tâches varie uniformément en fonction de la performance de la machine. Pour les machines indépendantes, la durée de traitement des tâches est totalement variable entre les différentes machines. Dans la notation de Graham, *P* est utilisé pour désigner les cas avec des machines identiques, *Q* pour les machines uniformes, enfin, *R* pour les machines indépendantes. Si un nombre est rajouté à coté de ces lettres, ce nombre montre le nombre de machines dans le problème.

Le deuxième champ de notre notation est composé par les hypothèses sur les tâches. Nous avons trois grandes hypothèses pour les tâches : la taille, la date de disponibilité, et la durée. La taille d'une tâche est simplement son volume sachant que les machines ont une capacité fixée et que la taille d'une tâche ne peut pas être plus grande que la capacité des machines. Les ensembles de DMR peuvent avoir des tailles différentes et donc les tâches de notre problème d'ordonnancement auront des tailles différentes, aussi. Mais, la littérature d'ordonnancement nous montre que les tâches ayant la même taille sont largement étudiées. L'expression « v_j » que nous allons insérer dans la notation de Graham signifie l'existence de tâches ayant des tailles différentes. Concernant les dates de disponibilité des tâches, nous n'avons pas pris en compte le rinçage manuel. La date de disponibilité d'une tâche représente le moment où elle est prête à être exécutée. Comme il est fait dans la littérature, nous désignons par « r_j » l'existence des dates de disponibilité dans un problème. Une tâche est disponible pour le lavage dès que le temps de pré désinfection est suffisant et si elle est arrivée dans le service de stérilisation. Enfin, les tâches peuvent avoir différentes durées d'exécution alors que les ensembles de DMR ont tous la même durée de lavage dans la réalité Nous désignons par « p_j » le fait que les tâches ont différentes durées d'exécution. Pour désigner que les durées d'exécution des tâches dans notre problème sont égales, nous mettrons « $p_j = p$ » dans le deuxième champ de la notation de Graham.

Enfin, le troisième champ dans la notation de Graham montre la fonction objectif. Nous avons cité dans la section précédente que les critères d'optimisation que nous considérons pour le chargement des laveurs sont : minimisation du nombre de cycles de lavage lancés, minimisation de la durée totale de lavage, minimisation du nombre d'ensembles de DMR en attente dans le stock de lavage, optimisation des durées de pré désinfection. Les trois premières fonctions objectifs ont des équivalences dans la littérature d'ordonnancement et de bin packing. Le tableau 2.1 montre ces équivalences.

Tableau 2.1. Fonctions objectifs relevant du problème de chargement des laveurs et leur équivalents en ordonnancement et bin packing

Chargement des laveurs	**Fonc. obj. de la littérature**
Minimisation du nombre de cycles de lavage lancés	Minimisation du nombre de bin utilisé
Minimisation de la durée totale de lavage	Minimisation du C_{max}
Minimisation du nombre d'ensembles de DMR en attente dans le stock de lavage	Minimisation du $\sum C_j$

Pour le cas où la durée d'exécution des tâches est unique et les machines sont identiques, la minimisation du nombre de cycles de lavage lancés est équivalent à la minimisation du nombre de bin utilisés dans un problème de bin packing. La minimisation de la durée totale de lavage est simplement la

minimisation du C_{max} qui signifie le temps total d'exécution en ordonnancement. Quant à la minimisation du nombre d'ensembles de DMR en attente dans le stock de lavage, son équivalent est la minimisation du stock d'en-cours montré par $\sum C_j$ dans la littérature d'ordonnancement. L'optimisation de la durée de pré désinfection est une extension de ce dernier et nécessite quelques modifications concernant la disponibilité des tâches. Nous expliquerons ces fonctions objectifs en détail dans les sections suivantes. Nous allons aussi citer quelques références importantes de la littérature pour chacun de ces problèmes.

A. Approches de type bin packing

Considérons le cas où tous les ensembles de DMR sont disponibles en même temps. Dans un tel cas, une fonction objectif naturelle est évidemment la minimisation du nombre de cycles de lavage lancés. Alors, le problème correspondant dans la littérature de recherche opérationnelle est le *bin packing*. Plus formellement, dans un problème de bin packing, il s'agit de trouver le rangement le plus économique possible pour un ensemble d'articles dans des boîtes, ou encore, il s'agit de minimiser le nombre de boites dans lesquelles les objets de tailles différentes sont rangés. Ainsi, les ensembles de DMR sont maintenant des objets et les machines sont des boites.

La minimisation du nombre de cycles de lavage peut être traitée dans deux cas : le cas où tous les ensembles de DMR sont simultanément disponibles devant les laveurs, et le cas où les ensembles de DMR arrivent au fur et à mesure de leur arrivée dans une journée. Le premier cas est simplement le cas des services de stérilisation externes. Les ensembles de DMR arrivent en grands lots, en général en fin de journée. Le deuxième cas est l'organisation générale de la stérilisation dans les hôpitaux possédant un service de stérilisation. Les ensembles de DMR sont traités au fur et à mesure depuis la première arrivée d'un ensemble de DMR jusqu'à la fermeture du service de stérilisation. Encore pour ce deuxième cas, nous pouvons imaginer une configuration de lavage même si elle n'est pas appliquée dans les services de stérilisation. Au lieu de laver les ensembles de DMR à différents moments d'une journée, il pourrait être possible de fixer, par exemple, deux moment différents, comme le midi et le soir. Alors, un grand nombre d'ensemble de DMR s'accumulent dans le stock de lavage et devient prêt pour être lavé. Donc, la modélisation de type bin packing peut s'appliquer à ce type de configuration.

Nous pouvons considérer deux cas différents d'un problème de bin packing : bin packing offline et bin packing online.

Le bin packing offline est le problème classique de bin packing. Tous les objets sont simultanément disponibles. Parmi les références les plus importantes,

nous pouvons citer le travail de Coffman et al. [25] où des algorithmes d'approximation sont proposés. Ce sont en fait les algorithmes classiques de bin packing parmi lesquels nous trouvons « next fit (NF), first fit (FF), best fit (BF), worst fit (WF), first fit decreasing (FFD) ». Ces algorithmes ont une garantie de performance asymptotique de 2, 1.7, 1.7, 1.7 et 11/9, respectivement.

Une extension du bin packing offline est le cas où les boites peuvent avoir différentes capacités [42]. En effet, les services de stérilisation possèdent en général des laveurs identiques. Mais il peut arriver d'avoir des machines de capacité différente. Le bin packing en présence de boites de taille variable peut être un problème dont les méthodes de résolution de la littérature peuvent être utilisées. Puisque dans les services de stérilisation nous avons des laveurs identiques, nous ne considérons pas ici le cas des machines de capacité différente.

Les deux types de problèmes cités ci-dessus peuvent être modifiés en considérant l'arrivée au fur et à mesure des objets. Dans ce cas, nous n'avons aucune connaissance sur l'instant d'arrivée des objets. Donc, ce nouveau problème s'appelle « bin packing online ». Un simple exemple d'algorithme pour ce problème est l'algorithme NF. Un meilleur algorithme est proposé par Lee et Lee [73]. Ils développent un algorithme, appelé *Harmonic*, dont la garantie de performance de 1.636.

Pour le cas online avec des boites ayant des capacités différentes, plusieurs stratégies peuvent être considérées comme par exemple des batch temporaires. Un batch temporaire signifie pour notre problème de chargement des laveurs, la possibilité de détruire un batch et de reconstruire un nouveau batch si nécessaire. Dans ce cas, les ensembles de DMR sont placés dans le stock de lavage qui représente le batch actuel. Ce batch est rempli au fur et à mesure des arrivées avec des nouveaux ensembles de DMR. Ensuite, l'arrivée d'un nouvel ensemble de DMR peut provoquer la destruction du batch actuellement ouvert. Nous référons le lecteur à Coffman et al. [25] pour plus d'information.

En fonction des explications données ci-dessus, nous pouvons résumer certaines hypothèses, qui peuvent être importantes pour notre problème, dans une figure (*cf.* figure 2.4).

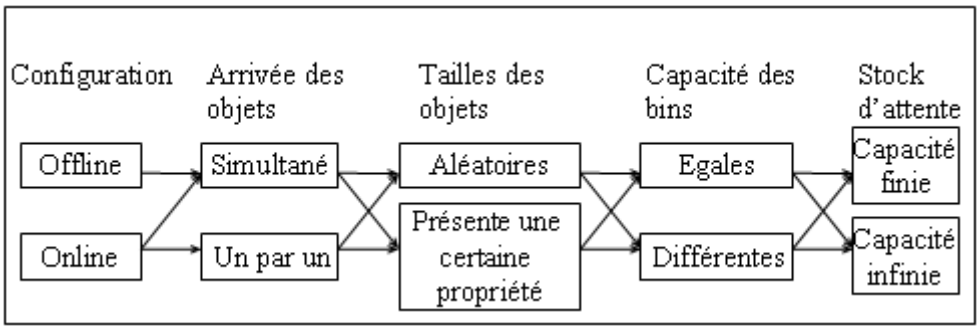

Figure 2.4. Différentes hypothèses pour une modélisation de type bin packing

Nous allons proposer une modélisation de type bin packing dans le chapitre 9 pour le fonctionnement d'un service de stérilisation externe. Dans ce cas, les hypothèses considérées sont :

- configuration offline
- tous les ensembles de DMR sont disponibles en même temps, donc arrivée simultanée,
- tailles aléatoires
- laveurs identiques, donc des batch de capacité égale,
- capacité du stock d'attente assez large, donc infinie.

B. Approches de type ordonnancement

Une deuxième identification possible consiste à modéliser l'étape de lavage comme un problème d'ordonnancement. En effet, nous avons la possibilité de laver plusieurs ensembles de DMR en même temps. Donc, notre problème d'ordonnancement est de type « ordonnancement par batch » où plusieurs tâches peuvent être exécutées en même temps sur une seule machine. Il s'agit donc de créer des batch qui peuvent contenir plusieurs tâches. De plus, les batch sont de type *p-batch*[3] puisque plusieurs tâches sont exécutées en même temps. Si les tâches d'un batch sont exécutées une par une sur la même machine, alors, nous avons un problème d'ordonnancement de type *s-batch*[4]. Une information plus détaillée est donnée dans la partie 2. Quand nous parlons de l'ordonnancement par batch dans ce mémoire, nous nous focalisons sur le *p-batch*.

Il y a beaucoup de critères qui ont été étudiés dans la littérature d'ordonnancement par batch. Nous représentons ici ceux qui sont les plus proches de notre problème. Commençons par les études offlines. Puisque, dans notre problème de stérilisation, les ensembles de DMR ont des tailles différentes, nous

[3] p-batch : *parallel batching* en anglais
[4] s-batch : *serial batching* en anglais

prenons en compte les travaux où les tâches ont des tailles différentes (Notons que la plupart des travaux sur l'ordonnancement par batch considère des tâches de taille identiques. Nous allons présenter ces travaux dans la partie 2.). Le tableau 2.2 montre les problèmes qui nous intéressent et indique s'ils ont été traités dans la littérature. Les quatre premières colonnes représentent les hypothèses concernant les machines et les tâches. La dernière colonne nous dit si un problème a déjà été étudié dans la littérature et avec quelle fonction objectif. Comme déjà dit dans la partie 2.4.2, nous nous sommes inspirés de la notation de Graham [51] pour décrire un problème. Par exemple, considérons un problème où nous avons des machines identiques en vitesse et en capacité avec des tâches de tailles et de dates de disponibilité différentes mais des durées d'exécution égales. Soit la minimisation des encours la fonction objectif. Ce problème peut être représenté par la notation : $P \mid p\text{-}batch,\ r_j,\ p_j=p,\ v_j,\ B \mid \sum C_j$. Dans le deuxième champ, *p-batch* veut dire que l'on a problème d'ordonnancement par batch de type *p-batch*, puis *B* veut dire que les machines ont une capacité fixe. Bien entendu, il peut y avoir d'autres hypothèses pour les machines ou pour les tâches, ou encore, d'autres fonctions objectifs. Par contre, nous n'allons pas détailler les travaux de la littérature dans cette section, mais dans la partie 2 (*cf.* chapitre 3).

Tableau 2.2. Etude offline pour certains problèmes d'ordonnancement par batch dans la littérature

Machines	Tâches			Etudié
	(tailles) v_j	*(durée)* p_j	*(date de disponibilité)* r_j	
1	≠	=	=	pour C_{max}, $\sum C_j$ (*cf.* section 3.1.2)
1	≠	=	≠	Non
1	≠	≠	=	pour C_{max} (*cf.* section 3.1.2)
1	≠	≠	≠	pour C_{max} (*cf.* section 3.1.2)
P	≠	=	=	pour C_{max} (*cf.* section 3.1.2)
P	≠	=	≠	Non
P	≠	≠	=	pour C_{max} (*cf.* section 3.1.2)
P	≠	≠	≠	pour C_{max} (*cf.* section 3.1.2)

Le tableau 2.2 a deux buts principaux. Il nous montre à la fois les problèmes étudiés et les pistes de travail pour lesquels nous pouvons apporter notre contribution. Nous avons simplement présenté ici des hypothèses qui relèvent de notre problème de stérilisation. Le tableau 2.2 met en évidence, malgré le grand nombre de travaux résolus en ordonnancement par batch, qu'il n'existe pas beaucoup de travail en ordonnancement par batch avec des tailles, durées d'exécution et dates de disponibilité différentes des tâches, or des dates d'arrivées différentes fait partie des hypothèses fortes de notre problème. Comme

nous l'avons déjà dit, nous pouvons encore considérer d'autres hypothèses comme par exemple, des dates de fins d'exécution souhaitée ou impérative pour les tâches, des machines de capacité différente, etc. Apparemment, la minimisation du C_{max} est la fonction objectif la plus étudiée. Ce critère est important aussi pour notre problème de stérilisation. Puisque l'étape de lavage est un goulet d'étranglement sur le processus de stérilisation, minimiser la durée totale de lavage est intéressant d'un point de vue opérationnel. De plus, la minimisation des encours peut être aussi un critère important pour l'étape de lavage. Par contre, cette fonction objectif n'est pas encore étudiée dans la littérature sauf un cas où nous avons une machine et des tâches de tailles différentes mais de durées d'exécution et de dates de disponibilité identiques.

D'après nos investigations, la durée de lavage est la même pour les DMR en présence des laveurs identiques. De plus, ils arrivent au service de stérilisation à différents moments d'une journée et ils ont des tailles différentes. Nous pouvons alors présenter notre problème d'ordonnancement par la notation suivante : $P \mid p\text{-}batch,\ r_j,\ p_j=p,\ v_j,\ B \mid \ldots$. La première fonction objectif que nous étudierons consiste à minimiser le C_{max}. Nous voyons, grâce au tableau 2.2, qu'un cas plus général est déjà étudié dans la littérature (le cas $P \mid p\text{-}batch,\ r_j,\ p_j,\ v_j,\ B \mid C_{max}$). Mais, l'étude du cas particulier représente bien l'étape de lavage. De plus, notre but est de proposer des méthodes plus puissantes pour ce cas que les approches proposées dans la littérature. Nous parlerons plus en détail de l'étude de ce cas dans le chapitre 4.

Une autre fonction objectif que nous allons traiter est la minimisation de l'attente de DMR dans le stock de lavage. Ce type d'optimisation peut aider à améliorer la performance du système de stérilisation tout en enlevant le goulet de l'étape de lavage. La fonction objectif équivalente de la minimisation d'attente dans le stock de lavage est la minimisation des en-cours, *i.e.* $\sum C_j$. Cette fonction objectif est moins étudiée dans la littérature que la minimisation du C_{max}. De plus, le problème $P \mid p\text{-}batch,\ r_j,\ p_j=p,\ v_j,\ B \mid \sum C_j$ n'est pas pour l'instant traité dans la littérature. Ce problème sera étudié dans le chapitre 5.

Une extension du problème de la minimisation de l'attente de DMR dans le stock de lavage peut être l'optimisation des durées de pré désinfection. Dans les problèmes ci-dessous, nous n'avons pas pris en compte les durées de trempage des DMR. Par contre, d'un point de vue opérationnel, l'optimisation des durées de trempage est un critère pertinent. Rappelons qu'il y a des cibles pour les durées de pré désinfection de DMR : 15 minutes, la durée minimale, 50 minutes, la durée maximale, et 20 minutes, la durée idéale. Pour ce but, nous allons légèrement modifier la notion de date de disponibilité des tâches dans le problème $P \mid p\text{-}batch,\ r_j,\ p_j=p,\ v_j,\ B \mid \sum C_j$. Dans le nouveau cas, une tâche sera dite disponible quand elle entre dans le stock de lavage et quand elle est pré

désinfectée au moins 15 minutes. Nous allons noter $\sum f_j$ la fonction objectif qui consiste à optimiser les durées de pré désinfection pour que les tâches aient la durée idéale de pré désinfection. Alors, la nouvelle notation sera la suivante : $P \mid p\text{-}batch, r_j, p_j=p, v_j, B \mid \sum f_j$. Cette fonction objectif est assez importante pour l'organisation de l'étape avale au lavage. Si nous pouvons assurer de bonnes durées de pré désinfection pour le DMR, le rinçage manuel peut s'avérer inutile et il peut être supprimé du processus de stérilisation pour les services qui l'appliquent. De plus, l'opérateur du poste de rinçage peut être affecté à d'autres postes. Des études offline seront faites pour ce critère dans le dernier chapitre de la partie 2.

En ce qui concerne la version online de ces problèmes, précisons qu'il y a un certain nombre de travaux prenant en compte des tâches de taille identique. Nous pouvons noter ces problèmes par la notation suivante : $P \mid p\text{-}batch, p_j=p, r_j, B, online \mid$ *une fonction objectif*. D'après notre connaissance, il n'y a pas encore un travail traitant les tâches de tailles différentes en présence de machines identiques. Pour la version online de notre problème de lavage, nous allons considérer le critère d'optimisation des durées de pré désinfection. Des études semi-online et online seront faites dans la partie 3.

Notons que si toutes les informations concernant les données d'un problème sont connues à l'avance, nous avons un problème offline. Les données de notre problème d'ordonnancement sont le nombre de machines et leur capacité, le nombre de tâches ainsi que leur taille, date de disponibilité et durée d'exécution. Si dans notre problème nous connaissons toutes les données sur les tâches à l'avance sauf une, par exemple la date de disponibilité des tâches, alors nous avons une configuration semi-online. Si aucune donnée concernant les tâches n'est connue à l'avance, alors la décision pour la création et l'exécution des batch dépend du moment où une tâche/les tâches devient/deviennent disponible(s). Dès qu'une tâche devient disponible pour l'exécution, toute autre information de cette tâche devient connue. Dans ce cas, nous avons un problème online. Nous allons donner plus d'information sur les configuration online et semi-online dans la partie 3.

2.5. Conclusion

Plusieurs caractéristiques du remplissage des laveurs nous permettent de modéliser ce problème comme un problème d'ordonnancement par batch. Les laveurs sont représentés par des machines et les ensembles de DMR par des tâches. Parallèlement aux caractéristiques des laveurs et des ensembles de DMR, nous considérons un problème d'ordonnancement en présence de machines identiques et de tâches qui peuvent avoir des tailles et des dates de disponibilité

différentes, mais des durée d'exécution égales. Nous divisons nos travaux en deux groupes principaux : optimisation offline, et optimisation online (aussi semi-online). Les premières méthodes d'optimisation offline que nous présenterons consistent à minimiser la durée totale de lavage et l'attente de DMR dans le stock de lavage. L'étude de ces premières fonctions objectifs étant des approches préliminaires, nous allons ensuite continuer avec une autre fonction objectif pour optimiser la durée de pré désinfection des DMR. Pour cette dernière fonction objectif, nous proposons d'abord des approches offline dans la partie 2, puis des approches semi-online et online dans la partie 3. Puisqu'il y a très peu (ou pas) d'interaction entre les blocs opératoires et le service de stérilisation, une configuration online est évidemment plus réaliste d'un point de vue application. Notons que le choix de ces fonctions objectifs correspond aux spécificités de l'étape de lavage.

Outre les méthodes d'optimisation pour l'étape de lavage, nous avons un deuxième but qui est de tester l'impact de nos méthodes d'optimisation semi-online et online sur l'ensemble du processus de stérilisation. Pour ce faire, nous implémenterons nos méthodes d'optimisation online et semi-online comme des stratégies de remplissage des laveurs dans un modèle de simulation. Ce modèle de simulation est réalisé en ARENA (version 11.0).

Conclusion de la partie 1

Comme tous les secteurs, les hôpitaux doivent faire face aux contraintes, comme par exemple un nombre limité de lits dans les services de soin, les problèmes budgétaires, un manque de bonnes méthodes de planification pour les interventions chirurgicales, etc. Nous nous intéressons ici plus particulièrement à l'amélioration de l'organisation du service de stérilisation.

Comparativement à d'autres services, la stérilisation n'apparaît pas toujours comme un service avec un fort enjeu. Cependant, il ne faut pas oublier que toutes les activités d'un hôpital sont liées entre elles. Au cas où l'une de ces activités ne fonctionne pas bien, tout le reste est inévitablement influencé par cette chute de performance. Dans notre cas, si les dispositifs médicaux ne sont pas bien stérilisés, il devient impossible de les utiliser pour une intervention chirurgicale.

Nous allons développer dans cette thèse des méthodes d'optimisation pour le chargement des laveurs dans l'étape de lavage des services de stérilisation. L'étape de lavage est généralement un goulet d'étranglement pour les services de stérilisation. Alors, il semble intéressant d'appliquer différentes stratégies de chargement des laveurs en nous inspirant des études déjà faites. En effet, l'étape de lavage d'un service de stérilisation est un endroit idéal où différentes techniques d'optimisation peuvent être appliquées. Ainsi, nous avons modélisé le problème du chargement des laveurs comme un problème d'ordonnancement par batch.

Pour notre problème de lavage, nous étudierons d'abord deux fonctions objectifs, la minimisation du C_{max} et la minimisation de $\sum C_j$, en nous inspirant des fonctions objectifs classiques des problèmes d'ordonnancement. Ensuite, en considérant la durée de trempage des DMR, nous proposons l'étude d'une troisième fonction objectif qui a pout but d'optimiser cette durée de trempage. La partie 2 étudie ces fonctions objectifs d'un point de vue offline. Dans la partie 3, nous étudierons l'optimisation de la durée de trempage pour laquelle nous proposerons des algorithmes online et semi-online. Dans la partie 4, nous étudierons la minimisation du nombre de cycles de lavage lancés quand tous les ensembles de DMR sont simultanément disponibles devant les laveurs. Ce type d'organisation est classique pour les services de stérilisation externes. La partie 3 et 4 présentent aussi des modèles de simulation afin de tester l'impact de nos méthodes d'optimisation sur tout le processus de stérilisation.

~ PARTIE 2 ~

Méthodes de résolution offline pour le problème de chargement des laveurs

Comme nous l'avons vu dans la partie 1, le problème du chargement des laveurs dans un service de stérilisation est un très bon exemple de problème d'ordonnancement par batch. Les caractéristiques de ce problème sont les suivantes : à l'étape de lavage, nous avons des laveurs identiques qui sont capables de traiter plus d'un ensemble de DMR à la fois. Le nombre d'ensembles de DMR mis dans un laveur dépend de la taille de ces ensembles et aussi de la capacité du laveur. Une autre particularité concernant les ensembles de DMR est qu'ils arrivent au service de stérilisation à différents instants. Nous modélisons donc le problème du chargement des laveurs comme un problème d'ordonnancement par batch où les ensembles de DMR sont considérés comme des tâches ayant des tailles et des dates de disponibilité différentes, les laveurs comme des machines.

Nous avons vu également dans la partie 1 que l'étape de lavage est généralement un goulet [36] dans les services de stérilisation et que la durée de pré désinfection des DMR est importante pour la stérilité et le cycle de vie des DMR. Le goulet du lavage peut provoquer des problèmes tels que « lavage d'un nombre insuffisant de DMR dans une journée », « attente d'un grand nombre de DMR dans le stock de lavage », « attentes longues des DMR avant le lavage », « amoncellement des DMR dans l'étape de conditionnement », etc. Ces problèmes sont similaires à ceux rencontrés dans la plupart des systèmes de production.

Nous consacrons cette partie aux méthodes d'optimisation offline, *i.e.* supposant que toutes les informations sur les tâches (dates d'arrivée, taille, durée d'exécution) sont connues à l'avance. Nous développons ici des méthodes de résolution pour différents problèmes d'ordonnancement et les comparons, le cas échéant, aux méthodes existantes dans la littérature. Nous commençons cette

partie par un premier chapitre sur la revue de littérature concernant les travaux sur l'ordonnancement par batch. Ensuite, nous considérons notre première fonction objectif dans le chapitre 4. Cette fonction objectif consiste à minimiser la durée totale de lavage. Dans le 5ème chapitre, nous cherchons à minimiser la durée d'attente des DMR à l'étape de lavage. Nous terminerons cette partie par un dernier chapitre où l'objectif est l'optimisation de la durée de pré désinfection des DMR.

Chapitre 3 : Motivations et Etat de l'art en ordonnancement par batch offline

3.1. Etat de l'art en ordonnancement par batch offline

Deux types de problèmes existent quand on parle de l'ordonnancement par batch : l'ordonnancement *s-batch* et l'ordonnancement *p-batch*.

Ordonnancement *s-batch*

Dans ce type de problème, les tâches sont exécutées sur les machines une par une. C'est-à-dire que toutes les tâches d'un même lot sont exécutées séquentiellement et généralement un réglage de la machine est nécessaire avant de commencer l'exécution d'un lot. Les tâches appartenant à un même lot peuvent avoir des durées d'exécution et des dates de disponibilité différentes. Le traitement du lot se termine quand l'exécution de toutes les tâches du lot est complètement terminée.

Ordonnancement *p-batch*

Contrairement à *s-batch*, les machines de type *p-batch* peuvent exécuter plusieurs tâches en même temps. Il peut y avoir, ou non, une durée de réglage de machine. La durée d'exécution d'un lot dépend de la plus grande durée d'exécution des tâches mises dans le lot.

Dans notre étude, les opérations de lavage peuvent être modélisées sous la forme d'un problème d'ordonnancement par batch de type *p-batch*. Donc, dans la suite de ce travail, seuls des problèmes de type *p-batch* seront considérés. De plus, nous allons utiliser dans la suite de notre travail le mot « *batch* » pour désigner un ensemble de tâches constituant un lot.

Les problèmes d'ordonnancement par batch (*p-batch*) peuvent encore être divisés en deux catégories, en fonction de la taille (ou volume) des tâches : les tâches peuvent être de tailles égales ou des tâches de tailles différentes. Dans les problèmes étudiés dans cette thèse, les tâches peuvent avoir des tailles différentes. En ce qui concerne la revue de la littérature d'ordonnancement par batch, les premières études parues posent l'hypothèse que les tâches ont des tailles unitaires.

Nous commençons donc d'abord par la revue de littérature des problèmes traitant les tâches de tailles unitaire. Ensuite, nous présenterons les travaux pour lesquels les tailles des tâches peuvent être différentes.

3.1.1. Ordonnancement *p-batch* de tâches de taille unitaire

L'ordonnancement par batch des tâches de taille unitaire a donné lieu à une littérature très vaste. De plus, tant du point de vue complexité que du point de vue méthode de résolution, ce type de problème est relativement éloigné du problème qui nous intéresse où les tâches ont des tailles différentes**.** Aussi, la revue présentée dans cette partie se veut-elle synthétique et pas forcément exhaustive.

Dans les problèmes prenant en compte les tâches de taille unitaire, une tâche exige une unité de la capacité de la machine. La machine a une capacité fixe, notée *B*, et elle peut traiter au plus *B* tâches en même temps. Par contre, contrairement aux problèmes traitant des tâches de tailles différentes, les problèmes concernant les tâches de tailles unitaires peuvent avoir une machine de capacité illimitée. C'est-à-dire que même si les tâches nécessitent toujours une unité de la capacité de la machine, sa capacité est tellement grande que toutes les tâches du problème peuvent être mises dans un même batch. Donc, s'il y a n tâches dans le problème, on a $n \leq B$. Par la suite, les durées d'exécution des tâches sont supposées différentes, sauf mention spécifique.

Dans le tableau 3.1, nous présentons une synthèse des travaux étudiant les problèmes d'ordonnancement par batch avec des tâches de tailles unitaires. Nous avons regroupé les références en fonction de la capacité de la machine, *i.e.* machine de capacité limitée ou illimitée. La première colonne indique cette particularité, la deuxième les références qui étudient le problème, la troisième colonne montre la méthode de résolution ainsi que la complexité de la méthode (si elle est précisée dans l'article), la colonne 4 donne la fonction objectif étudiée et enfin, la dernière colonne indique la complexité du problème.

Le tableau 3.1 résume les travaux réalisés sur une seule machine pour le cas où toutes les tâches sont disponibles en même temps. Outre les résultats présentés sur ce tableau, [10] montre que la minimisation de la somme pondérée des tâches en retard ($\sum w_j U_j$), et de la somme pondérée des retards ($\sum w_j T_j$) sont *NP-dur* au sens fort pour le cas où la capacité est limitée. Pour le cas de la capacité illimitée, $\sum T_j$ est démontré être *NP-dur* au sens faible dans [84].

Nous allons maintenant considérer le cas où les tâches ont des dates de disponibilité différentes, toujours sur une machine. Le tableau 3.2 montre des résultats importants sur les cas avec une machine de capacité limitée ou illimitée.

Tableau 3.1. Ordonnancement *p-batch* des tâches de taille unitaire, sans date de disponibilité, en présence d'une machine

Capacité machine	*Ref*	*Méthode & sa complexité*	*Fonc. obj.*	*Complexité*
capacité illimitée	[10]	*Algo. exact, 0(nlogn)*	$\sum C_j$	*Polynomial*
	[10]	*Algo. exact., 0(nlogn)*	$\sum w_j C_j$	*Polynomial*
	[10]	*Algo. exact, 0(n^2)*	L_{max}	*Polynomial*
	[10]	*Algo. exact, 0(n^3)*	$\sum U_j$	*Polynomial*
capacité limitée	[10]	*PD, 0($n^{b(b-1)}$)* *PD, 0($b^2k^2\ 2^k$)* (1)	$\sum C_j$	*Ouvert*
	[17]	*PD, 0(k^3b^{k+})* (1)		
	[16]	*B&B, heuristiques*		
	[33]	*B&B, heuristiques*		
	[55]	(1)*Algo. exacte, 0($k^2 3^k$)* *Algo. approx.*		
	[12]	*PTAS*		
	[135]	*B&B, heuristiques*	$\sum w_j C_j$	*Ouvert*

(1) Cas particulier où le nombre de durées d'exécutions différentes est égal à *k*

Tableau 3.2. Ordonnancement *p-batch* des tâches de taille unitaire avec dates de disponibilité différentes en présence d'une machine

Capacité machine	*Ref*	*Méthode & sa complexité*	*Fonc. obj.*	*Complexité*
capacité limitée	[127]	*Heuristiques, B&B*	C_{max}	*NP-dur au sens fort*
	[75]	*Heuristiques*		
	[83]	(1)*Algo. pseudo-polynomial, Algo. approx.*		
	[134]	(2)*PD*		
	[28]	*PTAS*	$\sum C_j$	*NP-dur au sens fort*
	[85]	*PTAS*	$\sum w_j C_j$	*NP-dur au sens fort*
	[53]	*Heuristiques*	$\sum U_j$	*NP-dur au sens fort*
	[134]	(2)*Heuristiques*	L_{max}	*NP-dur au sens fort*
	[137]	*Algorithme génétique*	L_{max}	
	[22]	*Algorithme génétique*	L_{max}	
	[53]	*Algorithme génétique*	T_{max}	*NP-dur au sens fort*
capacité illimitée	[114]	*Algorithme exact, 0(nlogn)*	C_{max}	*Polynomial*
	[29]	*PTAS*	$\sum C_j$	*NP-dur*
	[30]	*PTAS*	$\sum w_j C_j$	*NP-dur*
	[29]	*PTAS*		
	[83]	*heuristiques*	L_{max}	*NP-dur*
	[5]	*PTAS*		

(1) Cas particulier où le nombre de dates de disponibilité différentes est fixé
(2) Familles de tâches considérées

Les machines de capacité limitée étant le sujet de notre travail, la suite de la synthèse se restreint aux problèmes considérant différentes hypothèses, mais avec des machines de capacité limitée. Notons que les différentes hypothèses peuvent concerner les familles de tâches, des dates souhaitées ou impératives ou encore des contraintes de précédence entre les tâches. La première colonne du tableau 3.3 regroupe les travaux en fonction des dates de disponibilité. La deuxième colonne précise les hypothèses supplémentaires, *i.e.* familles de tâches, relation de concordance, etc. Pour un critère quelconque, si les tâches sont *en bonne concordance* (*agreeable* en anglais), les trier en fonction d'un critère les trie aussi de la même façon par rapport à un autre critère. Par exemple, en ce qui concerne la relation de concordance entre la date de disponibilité et la date souhaitée des tâches d'un problème, trier les tâches en ordre croissant des dates de disponibilité les trie aussi dans le même ordre des dates souhaitées. . Quand il y a une relation de concordance entre 2 critères (ou plus), cela est noté par le symbole « $\nearrow$ ».

Tableau 3.3. Ordonnancement *p-batch* des tâches de taille unitaire sur une machine avec des hypothèses supplémentaires

Dates de disponibilité des tâches	*Hypothèses supplémentaires*	*Ref.*	*Méthode & sa complexité*	*Fonc. obj.*	*Complexité*
Sans dates de disponibilité	*F, p_j comp.*	[134]	*Algo. exact, 0(n)*	*Cmax*	*Polynomial*
	$\nexists d_j$	[10]	*Algo. exact, min{0(nlogn), $0(n^2/b)$*		*Polynomial*
	$p_j \nearrow d_j$	[86]	*PD*	*$\sum T_j$*	*NP-dur au sens faible*
	F, p_j comp.	[94]	*Heuristique*		
	F, p_j comp.	[69]	*PD*	*$\sum w_j T_j$*	*NP-dur au sens fort*
	$p_j \nearrow d_j$	[72]			
	F, p_j comp.	[134]	*Algo. exact, 0(nlogn)*	*$\sum w_j C_j$*	*Polynomial*
	F, p_j comp.	[134]	*Algo. exact, 0(nlogn)*	*Lmax*	*Polynomial*
	F, p_j comp.	[58]	*PD*	*$\sum wjUj$*	*NP-dur*
	F, p_j comp.	[80]	*PD*		*NP-dur*
Dates de disponibilité différentes	*$p_j=p$*	[56]	*Algo. exact, 0(n)*	*Cmax*	*Polynomial*
	$p_j=p$, $\nexists d_j, r_j \nearrow d_j$	[56]	*Algo. exact, $0(n^2)$*		*Polynomial*
	F	[104]	*PTAS*		*NP-dur au sens fort*
	F, p_j comp.	[134]	*Algo. exact, 0(nlogn)*		*Polynomial*
	$p_j=p$	[7]	*PD, $0(n^8)$*	*$\sum C_j$*	*Polynomial*
	F, p_j comp.	[69]	*Heuristiques*	*$\sum w_j T_j$*	*NP-dur*
	F, p_j comp.	[79]	*Meta-heuristiques*		
	F, p_j comp.	[134]	*Heuristiques*	*Lmax*	*NP-dur*
	$r_j \nearrow d_j$	[139]	*Algo. exact, $0(n^3)$*		*Polynomial*
	$r_j \nearrow p_j \nearrow d_j$	[76]	*Algo. exact, 0(nlog$\sum$pj)*	*Tmax*	*Polynomial*
	$r_j \nearrow d_j$, $p_j=p$	[74]	*Algo. exact, $0(n^2 log((n-1)p)$*		*Polynomial*
	$r_j \nearrow d_j$, $p_j=p$	[74]	*Algo. exact, $0(n^3)$*	*$\sum U_j$*	*Polynomial*

Nous voyons dans le tableau 3.3 que plusieurs problèmes sont polynomiaux. A partir du tableau 3.3, les tableaux suivant contiennent un grand nombre de notation dans la colonne des hypothèses. Pour faciliter la lecture, le lecteur peut se référer à l'annexe *C* (et également à l'annexe *B*).

Après avoir synthétisé les cas d'une seule machine, nous allons maintenant continuer avec le cas de plusieurs machines. Pour ce cas plus général encore, nous présentons différents problèmes sous la forme de tableaux avec leurs complexités et les références correspondantes. Les problèmes présentés sur le tableau 3.4 sont tous soit *NP-dur* ou soit de complexité ouverte. Nous avons vu dans le tableau 3.3 que le problème de minimisation du C_{max} en présence des dates de disponibilité différentes est polynomial si toutes les durées d'exécution sont égales. Ce cas particulier n'est pas, à notre connaissance, étudié pour plusieurs machines. Dans le chapitre suivant, nous allons nous inspirer de ce cas pour réduire notre problème au cas des tailles égales sur des machines parallèles afin de donner un algorithme optimal.

Tableau 3.4. Ordonnancement *p-batch* sur des machines parallèles pour des tâches de taille unitaire

Dates de disponibilité des tâches	*Hypothèses supplémentaires*	*Ref.*	*Méthode & sa complexité*	*Fonc. obj.*	*Complexité*
Plusieurs machines, sans date de disponibilité		[16]	*Heuristiques*	$\sum C_j$	*ouvert*
		[125]	*Meta-heuristiques*	*Cmax*	*NP-dur*
	(3)mach. Ind.	[148]	*Heuristiques*	*Cmax*	*NP-dur au sens fort*
	mach. ind., $p_{mj} \geq p_{m'j}$	[149]	*PTAS*	*Cmax*	*NP-dur au sens fort*
	F, p_j *comp.*	[134]	*Heuristiques*	*Cmax*	*NP-dur*
	F, p_j *comp.*	[134]	*Heuristiques*	*Lmax*	*NP-dur*
	F, p_j *comp.*	[134]	*Heuristiques*	$\sum w_j C_j$	*NP-dur*
	F, p_j *comp.*	[6]	*Meta-heuristiques*	$\sum w_j T_j$	*NP-dur*
Plusieurs machines, dates de disponibilité différentes		[74]	*heuristiques*	*Cmax*	*NP-dur au sens fort*
		[77]	*PTAS*	*Cmax*	*NP-dur au sens fort*
		[74]	*heuristiques*	*Lmax*	*NP-dur au sens fort*
		[87]	*PTAS*	*Lmax*	*NP-dur au sens fort*
	F	[81]	*PTAS*	*Cmax*	*NP-dur*
	F, p_j *comp.*	[134]	*Heuristiques*	*Cmax*	*NP-dur*
	F, p_j *comp.*	[91]	*Heuristiques*	*Lmax*	*NP-dur au sens fort*
	F, p_j *comp.*	[100]	*Meta-heuristiques*	$\sum w_j T_j$	*NP-dur*

(3) machines indépendantes : le cas où la durée d'exécution d'une tâche dépend des machines sans qu'il y ait un ratio d'uniformité entre les machines

Nous allons maintenant continuer avec le cas des tâches ayant des tailles différentes. Les problèmes que nous allons présenter ci-après sont plus importants pour nous. Nous allons donc faire une présentation plus détaillée de ces travaux.

3.1.2. Ordonnancement *p-batch* de tâches de tailles différentes

Dans les problèmes d'ordonnancement avec des tâches qui peuvent avoir des tailles différentes, les machines ont toujours des capacités limitées. Puisque les machines ont des capacités fixées, la taille totale des tâches qui sont exécutées ensemble dans un même batch ne doit pas dépasser la capacité de la machine. Chaque tâche est affectée à un seul batch (sauf cas particuliers) et la durée d'exécution d'un batch est égale à la plus grande durée d'exécution des tâches contenues dans ce batch [113]. Si les familles de tâches sont considérées, les tâches sont regroupées dans différents ensembles en fonction de leur durée d'exécution, de la durée de réglage de la machine, etc. Il y a deux différents types de famille : famille compatible et famille incompatible. Dans le cas où les familles compatibles sont considérées, les tâches appartenant aux différentes familles peuvent être mises dans un batch [3]. Si les familles incompatibles sont considérées, seules les tâches de la même famille peuvent être regroupées dans un batch [101]. Notons que les tâches d'une même famille peuvent avoir des tailles, des durées d'exécution et des dates de disponibilité différentes.

Nous faisons le même classement que dans la section précédente et regroupons les travaux en fonction du nombre de machines et des dates de disponibilité. Nous présentons ces articles dans le tableau 3.5 et expliquons ensuite plus en détail leur contenu.

Uzsoy [133] a été, à notre connaissance, le premier à étudier l'ordonnancement par batch des tâches ayant des tailles différentes. Dans son problème, les tâches sont toutes disponibles en même temps, mais elles peuvent avoir des durées d'exécution différentes. Il étudie la minimisation du C_{max} et de la $\sum C_j$ sur une machine et montre que ces problèmes sont *NP-dur*. Pour la minimisation du C_{max}, il propose des heuristiques et pour la minimisation de la $\sum C_j$, il développe un algorithme de *B&B*. Pour le même problème sur le C_{max}, Dupont et Ghazvini [34] proposent des heuristiques plus performantes que celles d'Uzsoy [133]. Dans les années qui suivent, nous voyons apparaitre des algorithmes d'approximation et des méta-heuristiques. Zhang *et al.* [144] proposent un algorithme d'approximation ayant une garantie de 7/4 pour la minimisation du C_{max} et le comparent aux heuristiques d'Uzsoy [133]. Toujours pour le C_{max}, Dupont et Dhaenens-Flipo [32] développent une méthode de *B&B*. Kashan *et al.* [61] proposent un algorithme génétique et affirment que leur méthode est plus performante que le recuit simulé proposé par Melouk *et al.* [95]. Kasan *et al.* [63] travaillent sur un cas particulier où les durées d'exécution des tâches ayant une taille plus grande que $1/k$ ne sont pas plus petites que celles des tâches dont la taille est plus petite que $1/k$ (avec k entier plus grand que 1). Ils proposent un algorithme avec une garantie de performance asymptotique de $(k+1)/k$. Parsa *et al.* [111] proposent un algorithme de *B&P* et le comparent à la

seule méthode exacte qui est le *B&B* de Dupont et Dhaenens-Flipo [32]. Pour la plupart des instances, *B&P* trouve la solution optimale plus vite que le *B&B* de Dupont et Dhaenens-Flipo [32]. Kempf *et al.* [65] prennent en compte des familles de tâches avec une seconde hypothèse : le nombre de tâches à mettre dans un batch est aussi limité. Ils proposent un modèle PLNE et des heuristiques pour la minimisation des critères C_{max} et la $\sum C_j$. En ce qui concerne la minimisation de $\sum C_j$, Ghazvini et Dupont [49] proposent des heuristiques, alors que Zhang *et al.* [150] étudient le cas particulier où les durées d'exécution des tâches sont égales à 1. Ces derniers proposent 3 algorithmes d'approximation ayant une garantie asymptotique 4, 2 et 3/2, respectivement. Azizoglu et Webster [2] considèrent des tâches pondérées. Enfin, Azizoglu et Webster [3] considèrent des familles de tâches pour lesquelles la durée d'exécution est la même pour toutes les tâches d'une même famille.

Tableau 3.5. Ordonnancement *p-batch* des tâches de tailles différentes et sans date de disponibilité

Nombre de mach.	*Hypothèses supplémentaires*	*Ref.*	*Méthode*	*Fonc. obj.*	*Complexité*
Une machine		[133]	Heuristiques	*Cmax*	*NP-dur*
		[34]	Heuristiques		
	F, p_j comp. nombre limité de tâches dans un batch	[65]	MILP, heuristiques		
		[144]	Algo. d'approx.		
		[32]	*B&B*		
		[95]	Recuit simulé		
		[61]	Algo. génétique		
	$p_j \nearrow v_j$	[63]	Algo. d'approx.		
		[111]	Branch and price		
		[64]	Algo. génétique	*Cmax+Lmax*	*NP-dur*
		[133]	*B&B*	$\sum C_j$	*NP-dur*
		[49]	Heuristiques		
	F, p_j comp. nombre limité de tâches dans un batch	[65]	MILP, Heuristiques		
	$p_j = 1$	[150]	Algo. d'approximation		
		[2]	*B&B*	$\sum w_j C_j$	*NP-dur*
	F, p_j comp.	[3]	*B&B*		
Plusieurs machines		[18]	Recuit simulé	*Cmax*	*NP-dur*
	F, p_j comp.	[67]	Algo. génétique		
		[62]	Algo. génétique		
	Flowshop	[68]	MILP, recuit simulé		
	F, p_j comp.	[67]	Algo. génétique	$\sum C_j$	*NP-dur*
	F, p_j comp.	[67]	Algo. génétique	$\sum w_j C_j$	*NP-dur*

Pour le cas des machines parallèles, Kashan *et al.* [62] développent un algorithme génétique qui est, d'après les expérimentations numériques, plus performant que le recuit simulé proposé par Chang *et al.* [18] pour la minimisation du C_{max}. Koh *et al.* [67] étudient la minimisation du C_{max}, de la $\sum C_j$ et de la $\sum w_j C_j$ pour lesquels ils développent 2 algorithmes génétiques. Kumar *et al.* [68] étudient le cas d'un flowshop composé de 2 machines *p-batch* pour lequel un algorithme de recuit simulé est proposé.

Puisque le reste des travaux considère des dates de disponibilité différentes, ils sont plus importants pour notre étude. Le premier travail sur la minimisation du C_{max} sur une seule machine avec des dates de disponibilité différentes est celui de Li *et al.* [77]. Les auteurs présentent un algorithme d'approximation de garantie de performance $2+\varepsilon$. Nong *et al.* [104] étudient d'abord le cas de tâches ayant des tailles unitaires pour lequel ils proposent un PTAS. Ensuite, pour le cas général, ils proposent un algorithme d'approximation de garantie 5/2. Lu *et al.* [90] travaillent sur la minimisation simultanée du C_{max} et de la pénalité de rejeter des tâches (le rejet d'une tâche signifie qu'elle n'est pas exécutée). Ils proposent d'abord un PTAS pour le cas où les tâches peuvent être coupées. Ensuite, ils l'utilisent pour proposer un algorithme d'approximation de garantie $2+\varepsilon$.

Tableau 3.6. Ordonnancement *p-batch* de tâches de tailles différentes et avec date de disponibilité

Nombre de machines	*Hypothèse supplémentaire*	*Ref.*	*Méthode*	*Fonc. obj.*	*Complexité*
une machine, dates de disponibilité différentes		[77]	Algo. d'approximation	*Cmax*	*NP-dur*
		[78]			
	F	[104]	Algo. d'approximation		
	Pénalité de la tâche rejetée	[90]	Algo. d'approximation	*Cmax+rejet de tâche*	*NP-dur*
plusieurs machines, dates de disponibilité différentes		[23]	*PLNE*, heuristiques	*Cmax*	*NP-dur*
		[26]	Méta-heuristique		
		[27]	heuristiques		
		[21]	Algo. génétique, colonie de fourmis		
	Machines de capacité ≠	[138]	Algo. génétique, recuit simulé		

Chung *et al.* [23] sont, à notre connaissance, les premiers à avoir étudié l'ordonnancement par batch des tâches ayant des tailles et des dates de disponibilité différentes sur les machines parallèles. Ils ont d'abord proposé un modèle PLNE et ensuite deux heuristiques. Leurs heuristiques sont composées de deux phases : la première phase construit des batch, la deuxième phase affecte les batch aux machines. La première phase est commune aux deux heuristiques et est inspirée de l'heuristique « DELAY » présentée par Lee et Uzsoy [75] qui considèrent des tâches de taille unitaire. La phase 1 utilise deux paramètres : α,

pour déterminer un horizon de temps dans lequel les tâches sont disponibles pour créer un batch, β, pour déterminer un seuil de remplissage de batch. Ils exécutent l'algorithme avec différentes combinaisons de α et β. La rapidité de l'algorithme dépend largement du choix de ces paramètres. Comme cité dans Lee et Uzsoy [75], il faudrait tourner l'heuristique avec peu de valeurs différentes de α et β pour avoir une heuristique rapide. Damodaran *et al.* [26] développent une méta-heuristique appelée Greedy Randomised Adaptive Research Procedure (GRASP). Ils signalent que cette approche GRASP garantit la solution optimale pour les petites instances et est plus efficace que les heuristiques proposées par Chung *et al.* [23]. Damodaran et Velez Gallego [27] proposent une heuristique. Cette heuristique fonctionne d'abord en déterminant un horizon de temps, ensuite, elle résout un problème de sac à dos pour sélectionner les tâches à mettre dans un batch. Elle est comparée au modèle PLNE et aux heuristiques proposées par Chung *et al.* [23] ainsi qu'à la méta-heuristique de Damodaran *et al.* [26]. Il est indiqué que leur heuristique surpasse celle de Chung *et al.* [23] et donne des résultats similaires à la méthode GRASP. Par contre, cette heuristique est de complexité pseudo-polynomiale, puisqu'elle résout le problème de sac à dos en utilisant l'algorithme dynamique proposé par Martello et Toth [93]. Chen *et al.* [21] proposent un algorithme génétique et une optimisation par colonies de fourmis pour la création des batch. Pour l'affectation aux machines, ils proposent une heuristique qui est utilisée en commun dans les deux méta-heuristiques. Ils comparent leurs méthodes de résolution aux heuristiques proposées par Dupont et Ghazvini [34] en rajoutant à celle-ci leurs propres méthodes d'affectation de batch aux machines. Wang et Chou [138] considèrent des machines ayant des capacités différentes. Ils développent un algorithme génétique et un algorithme recuit simulé et testent leurs performances sur les instances définies par Chung *et al.* [23]. D'après les expérimentations numériques, leurs méta-heuristiques sont plus performantes que les méthodes de Chung *et al.* [23].

3.2. Conclusion

Les travaux sur l'ordonnancement par batch des tâches de taille unitaire sont nombreux, et ainsi de nombreux résultats sont connus. Par contre, l'ordonnancement par batch des tâches de tailles différentes, surtout avec des dates de disponibilité différentes, est moins étudié. Grâce au fait que l'étape de lavage des services de stérilisation est un processus d'ordonnancement par batch, nous allons essayer de contribuer à la littérature d'ordonnancement, tout en appliquant nos résultats aux services de stérilisation.

Le premier problème que nous traitons a pour but de minimiser le C_{max}. D'après la revue de la littérature (*cf.* tableau 3.6), il existe des approches méta-heuristiques et heuristiques. Ces heuristiques donnent de bons résultats, mais ont

une complexité élevée. Donc, nous allons proposer des algorithmes rapides (et de complexité polynomiale) et aussi étudier des cas particuliers pour lesquels nous pouvons proposer des algorithmes exacts.

Ensuite, nous allons continuer avec une autre fonction objectif, la minimisation de la $\sum C_j$, qui n'a pas encore été étudiée dans la littérature. Pour cette fonction objectif aussi, nous allons étudier le cas général et quelques cas particuliers. Nous allons terminer les études offlines avec un quatrième chapitre qui a pour but de fournir des bonnes durée de pré désinfection pour les DMR, tout en étudiant l'opportunité de supprimer l'étape de rinçage manuel.

Chapitre 4 : Minimisation de la durée totale de lavage

4.1. Introduction

La moitié des services de stérilisation qui ont participé au questionnaire EESS [36] remarquent que l'étape de lavage est le goulet principal du système de stérilisation. Ce fait nous a fait réfléchir aux moyens pour gérer ce goulet. Ainsi, minimiser la durée totale de lavage semble être un critère important afin de lutter contre ce goulet. Plus la durée totale de lavage est raccourcie, plus le nombre d'ensembles de DMR lavés pourra augmenter.

Le problème de la minimisation de la durée totale de lavage correspond à la minimisation du C_{max} dans la littérature des problèmes d'ordonnancement. En nous inspirant de la notation de Graham [51], nous proposons la notation suivante pour le problème étudié : $P \mid p\text{-}batch, r_j, p_j=p, v_j, B \mid C_{max}$. Dans cette notation, P correspond aux machines identiques, *p-batch* à l'ordonnancement à traitement par batch, r_j et v_j aux dates de disponibilité et tailles (ou volumes) des tâches, $p_j = p$ signifie que les durées d'exécution sont identiques et B est la capacité des machines. Finalement, C_{max} est l'objectif d'optimisation.

Les travaux effectués dans chapitre ont donné lieu aux publications [106] et [110].

4.2. Complexité du problème

Nous avons déjà cité la complexité de certains problèmes dans le chapitre précédent. Il est possible de réduire notre problème à un problème de bin packing en considérant un cas particulier.

Le cas où toutes les dates de disponibilité des tâches sont égales (*i.e.* $r_j = r$ pour tout j). Ce cas est équivalent à un problème de bin packing qui est un problème *NP-dur* au sens fort [25]. Le problème étudié dans cette section est donc *NP-dur* au sens fort.

4.3 Relation entre le bin packing et la minimisation du C_{max}

Nous considérons de nouveau le cas où les tâches peuvent avoir des dates de disponibilité différente. Intuitivement, une première approche pour la minimisation du C_{max} serait de minimiser le nombre de batch. Nous allons montrer que la minimisation du C_{max} n'est pas équivalent à minimiser le nombre de batch.

Un cas particulier de notre problème est là où toutes les dates de disponibilité sont égales. Alors, notre problème devient équivalent à un problème de bin packing. La minimisation du nombre de batch conduit alors à la minimisation du C_{max}. Par contre, quelle valeur obtiendrait-on pour le C_{max} en minimisant le nombre de batch lorsque les dates de disponibilité des tâches sont différentes ?

Lemme 1. Minimiser le nombre de batch ne minimise pas forcément la fonction objectif du problème $P \mid p\text{-}batch, r_j, p_j=p, v_j, B \mid C_{max}$

Preuve. Pour prouver le lemme 1, il suffit de donner un exemple pour lequel le C_{max} n'est pas minimisé lorsque le nombre de batch est minimisé. Considérons 2 machines et 6 tâches dont les dates de disponibilité et les tailles sont données dans le tableau 4.1. En considérant la durée d'exécution des tâches égale à p et la capacité des batch égale à B, nous allons ordonnancer ces tâches sur deux machines.

Tableau 4.1. Instance illustrative pour le lemme 1

Tâche	**1**	**2**	**3**	**4**	**5**	**6**
r_j	p	$2p$	$3p$	$4p$	$4p$	$4p$
v_j	B-ε	B-ε	B-ε	ε	ε	ε

Notons que dans le tableau 4.1, ε signifie un nombre très petit inférieur à $B/3$. Nous avons 3 tâches de grande taille et 3 taches de petite taille. Une solution minimisant le C_{max} pour cette instance est montrée sur la figure 4.1 par un diagramme de Gantt. Elle comporte 4 batch.

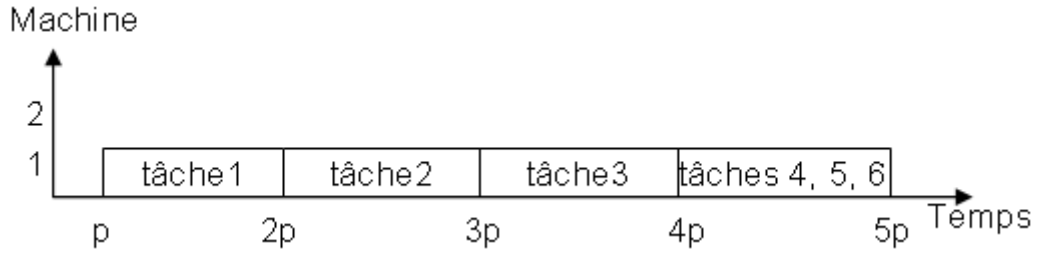

Figure 4.1. Solution optimale pour le critère C_{max} de l'instance donnée dans le tableau 4.1.

Si notre but est de minimiser le nombre de batch, nous pouvons obtenir seulement 3 batch en mettant les grandes tâches avec les petites tâches. L'ordonnancement de ces 3 batch est montré sur la figure 4.2 par un diagramme de Gantt.

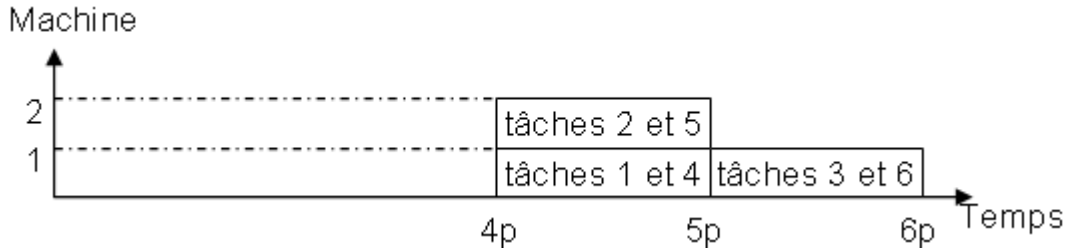

Figure 4.2. Résolution de l'instance donnée dans le tableau 4.1. en minimisant le nombre de batch

Le C_{max} trouvé en minimisant le nombre de batch est plus grand que la solution optimale. ■

Ainsi, en général, minimiser le nombre de batch ne minimise pas le C_{max}, mais, il peut y avoir des cas où minimiser le nombre de batch donne la solution optimale pour le C_{max}. Outre le cas où les tâches sont disponibles en même temps, considérons le cas particulier où toutes les dates de disponibilité des tâches sont dans l'intervalle [0, p [.

Lemme 2. Si toutes les dates de disponibilité des tâches sont dans l'intervalle [0,*p*[, alors, minimiser le nombre de batch minimise aussi la fonction objectif du problème 1 | *p-batch*, r_j, $p_j=p$, v_j, B | C_{max}

Preuve. Supposons que pour une instance donnée, le nombre minimal de batch est égal à *nb*. Comme toutes les tâches de l'instance sont disponibles dans [0, p [, au pire cas, l'exécution du premier batch commence à l'instant $p-\varepsilon$ avec ε un nombre très petit. Alors, l'exécution de *nb* batch se termine à l'instant $nb*p + p-\varepsilon$. Maintenant, augmentons le nombre de batch de 1. Alors au meilleur des cas, l'exécution du premier batch commence à l'instant 0. Ainsi, la durée totale d'exécution de $nb+1$ batch devient égale à $(nb+1)*p$, qui est supérieur au C_{max} en minimisant le nombre de batch. ■

Par contre, ce lemme n'est valide que pour le cas d'une seule machine. Minimiser le nombre de batch dans un problème à M ($M > 1$) machines n'est pas équivalent à minimiser le C_{max} si toutes les dates de disponibilités des tâches sont dans l'intervalle [0, *p* [.

Lemme 3. Si toutes les dates de disponibilité des tâches sont dans l'intervalle [0,p[, alors, minimiser le nombre de batch n'est pas équivalent à minimiser la fonction objectif du problème $P \mid p\text{-}batch, r_j, p_j=p, v_j, B \mid C_{max}$.

Preuve. Pour prouver le lemme 3, il suffit de donner un exemple pour lequel le C_{max} n'est pas minimisé lorsque le nombre de batch est minimisé. Considérons l'instance donnée dans le tableau 4.2 avec 2 machines dans le système.

Tableau 4.2. Instance illustrative pour le lemme 3

tâche	**1**	**2**	**3**	**4**	**5**	**6**
r_j	*0*	*0*	*0*	$p-\varepsilon$	$p-\varepsilon$	$p-\varepsilon$
v_j	$B-\varepsilon$	$B-\varepsilon$	$B-\varepsilon$	ε	ε	ε

ε est un nombre très petit. La solution optimale minimisant le C_{max} comporte 4 batch où les 3 premiers contiennent les tâches 1, 2 et 3, respectivement et le $4^{ème}$ batch contient les tâches 4 à 6. Montrons la solution optimale sur un diagramme de Gantt (*cf.* figure 4.3).

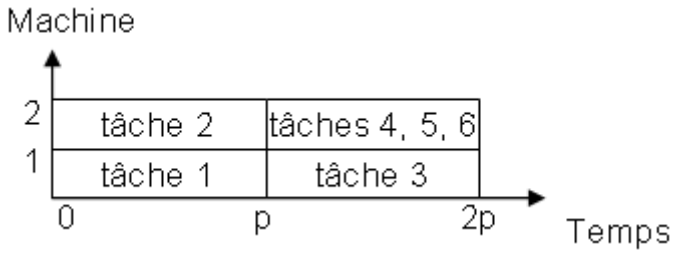

Figure 4.3. C_{max} optimal pour l'instance donnée dans le tableau 4.2

Si l'objectif est la minimisation du nombre de batch, la solution optimale contient 3 batch, chacun contenant une grande et une petite tâche. Comme les petites tâches sont disponibles à l'instant $p-\varepsilon$, le début d'exécution commence à $p-\varepsilon$ et on exécute forcément 2 batch sur l'une des deux machines. Une configuration possible sur la figure 4.4.

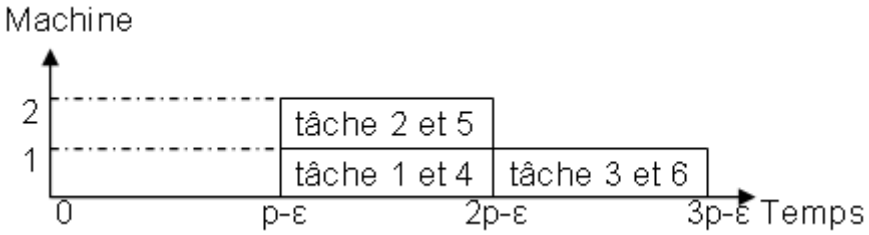

Figure 4.4. Résolution de l'instance donnée sur le tableau 4.2 en minimisant le nombre de batch

Le C_{max} trouvé en minimisant le nombre de batch est plus grand que la solution optimale. ■

Nous voyons grâce aux exemples donnés pour les lemmes 1 et 3 que la minimisation du nombre de batch n'est pas forcément une bonne approche pour minimiser le C_{max}. Mais quand même, il est évident que l'augmentation du nombre de batch a tendance à faire augmenter le C_{max}. Donc, diminuer le nombre de batch peut servir à avoir des bonnes valeurs pour le C_{max} mais pas nécessairement optimales. Par la suite, nous utiliserons cette intuition pour fournir des heuristiques performantes.

Les sections suivantes sont consacrées à la résolution du cas général et des cas particuliers du problème.

4.3.1. Heuristiques inspirées des algorithmes classiques de bin packing

Dans cette section, nous présentons des algorithmes inspirés des heuristiques classiques de bin packing pour notre problème. Nous avons vu dans la section précédente que minimiser le nombre de batch ne minimise pas forcément la valeur de C_{max}. Rappelons-nous du contre exemple que nous avons donné pour le lemme 1. Dans ce contre exemple, nous avons 6 tâches, le nombre minimal de batch est 3 (valeur du C_{max} 6p) alors que la valeur optimale pour le C_{max} est obtenue avec 4 batch (valeur du C_{max} 5p). Alors, avoir un nombre de batch considérablement petit (mais pas forcément minimal) peut aider à minimiser le C_{max}. Donc, même si en minimisant le nombre de batch on ne minimise pas le C_{max}, trouver le nombre de batch minimal, ou considérablement petit, permet d'avoir une bonne solution approchée pour le C_{max}.

Nous proposons 4 heuristiques : First Fit Modifié (*FFM*), Best Fit Modifié (*BFM*), Worst Fit Modifié (*WFM*) et Next Fit Modifié (*NFM*). Ces algorithmes, qui sont de complexité polynomiale, ont pour but de trouver des solutions rapides en créant un nombre de batch considérablement petit. La première étape de chacune des heuristiques que nous proposons consiste à trier les tâches en ordre croissant des dates de disponibilité. Une fois qu'un batch est ouvert, la première tâche de la liste précédemment créée est placée dans ce batch et elle est effacée de la liste. Ensuite, les algorithmes classiques [25] First Fit, Best Fit, Worst Fit et Next Fit sont appliqués pour les tâches qui n'ont pas encore été placées dans un batch.

Nous donnons ci-après les algorithmes *FFM*, *BFM*, *WFM*, *NFM* et nous illustrons leur principe en les appliquant tous sur un exemple.

A. First Fit Modifié (FFM)

Tout d'abord, une liste, L_1, contenant les tâches en ordre croissant des dates de disponibilité est créée. A chaque fois qu'une tâche est placée dans un nouveau batch, nous parcourons la liste L_1 contenant le reste des tâches afin de remplir ce batch. Le but principal de cette heuristique est d'éviter le gaspillage de capacité des batch.

Algorithme *FFM*
1. Trier les jobs en ordre croissant des dates de disponibilité r_j : L_1
2. Tant que L_1 n'est pas vide
    2.1. Choisir le premier élément de L_1 et le mettre dans le premier batch où il peut entrer. Si aucun batch ne convient, créer un nouveau batch pour cet élément
2.2 Mettre à jour L_1.
Fin tant que
3. Fixer pour chaque batch la date au plus tôt [1] égale à la plus grande date de disponibilité des tâches qu'il contient.
4. Ordonnancer les batch consécutivement en ordre croissant des dates au plus tôt sur les machines.

En cas d'égalité des dates de disponibilité à l'étape 1 ou des dates au plus tôt à l'étape 4, le choix de l'ordre des tâches dans l'étape 1 et des batch dans l'étape 4 est arbitraire.

Exemple illustratif : Soient 6 tâches ayant les tailles *B/4, B/2, B/4+ε, ε, 3B/4, ε,* respectivement. Affectons à ces tâches des dates de disponibilité : 20, 10, 30, 60, 100, 80, respectivement et en minutes. Soit la durée d'exécution égale à 100 minutes. Nous avons 2 machines de capacité *B*. ε est un nombre très petit. Numérotons ces tâches de 1 à 6. Notons que pour cet exemple, le nombre minimal de batch est 2, et le C_{max} optimal est 200, ce qui est trouvé avec 2 batch.

Résolution de l'exemple illustratif avec *FFM* : Après avoir trié les tâches en ordre croissant des dates de disponibilité, nous obtenons une liste, L_1, des tâches en ordre suivant : 2, 1, 3, 4, 6, 5. Nous pouvons montrer l'arrivée de ces tâches par un diagramme de date de disponibilité/taille (*cf.* figure 4.5).

[1] Date au plus tôt d'un batch : moment où le batch est prêt à être exécuté

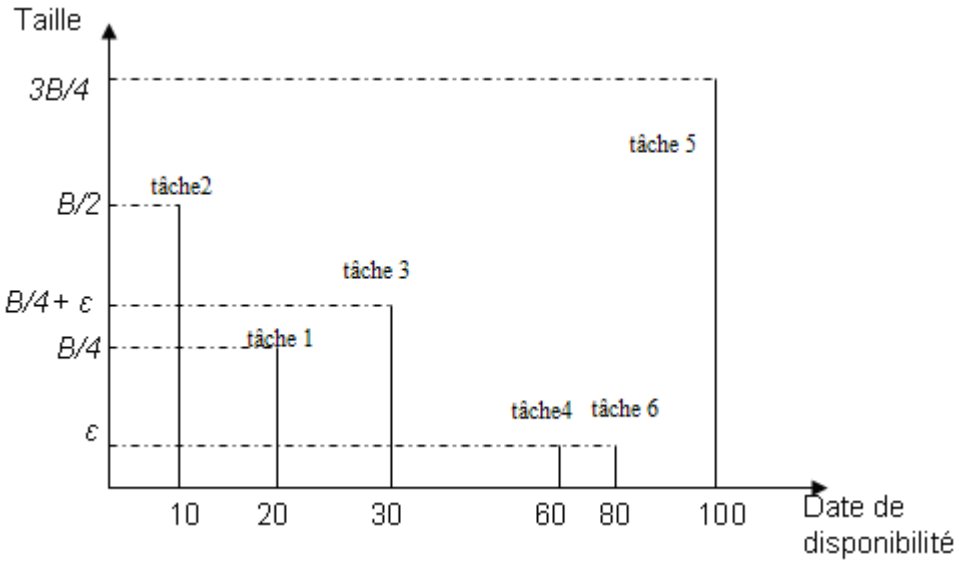

Figure 4.5. Diagramme *date de disponibilité/taille* pour les tâches de l'exemple illustratif

Pour l'instant, aucune tâche n'est placée dans un batch. Le batch à remplir est le batch numéro 1. Nous prenons le premier élément de L_1 (tâche 2) qui est de taille *B/2*, et le plaçons dans le batch 1. Le deuxième élément (tâche 1) a la taille *B/4* et peut entrer dans le batch 1. Mais, le troisième élément (tâche 3) a la taille $B/4+\varepsilon$ et ne peut pas entrer dans le batch 1. Alors, on ouvre un nouveau batch, batch 2, pour cet élément. Le quatrième élément (tâche 4) a la taille ε et nous avons assez de place pour lui dans le batch 1. Il est alors placé dans le batch 1. De même que le $5^{ème}$ élément (tâche 6). Maintenant, le batch 1 a la taille $3B/4 + 2\varepsilon$, et le batch 2 a la taille $B/4+\varepsilon$. Le dernier élément de la liste (tâche 5) a la taille *3B/4*. Il ne peut entrer ni dans le batch 1 ni dans le batch 2. Alors, il est placé dans un nouveau batch (batch numéro 3).

Il nous reste à affecter les dates au plus tôt à ces trois batch. La plus grande date de disponibilité des tâches dans le batch 1 est 80 (la date de disponibilité de la tâche 6). Le batch 2 ne contient qu'une seule tâche dont la date de disponibilité fixe la date au plus tôt du batch 2 à l'instant 30. De la même manière, le batch 3 a une date au plus tôt égale à 100.

La dernière étape de *FFM* est d'affecter ces tâches aux machines. Le batch 2 a la date au plus tôt la plus petite, donc on peut l'affecter à la première machine disponible et commencer son exécution à l'instant 30. Puisque les machines sont identiques et nous avons une durée d'exécution unique, nous pouvons l'affecter à la machine numéro 1. Le batch 1 et 3 ont les dates au plus tôt égales à 80 et 100. On affecte alors le batch 1 à la machine 2. Enfin, le batch 3 est affecté à la

machine 1. Ainsi, la valeur du C_{max} trouvé est 230. Cette solution est montrée dans la figure 4.6 par un diagramme de Gantt.

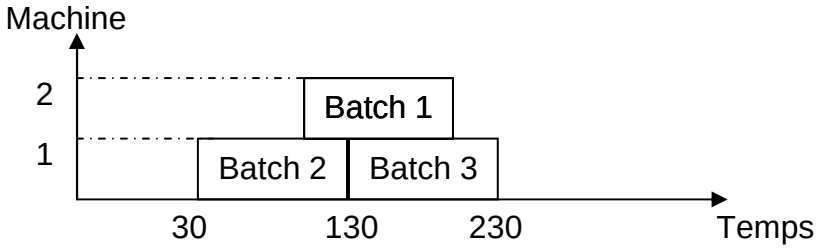

Figure 4.6.Résolution de l'exemple illustratif avec *FFM*

A part le tri à l'étape 1, *FFM* est identique à *FF*. Donc, la complexité de *FFM* est égale à la complexité de *FF* qui est de *O(NlogN)* [25]. Notons que l'algorithme *FF* est un algorithme online. C'est-à-dire que nous n'avons pas besoin de connaître à l'avance les tâches. Elles sont mises dans les batch au fur et à mesure de leur arrivée. Par contre dans notre cas, *FFM* devient un algorithme offline, car l'étape 1 consiste à trier les tâches d'après les dates de disponibilité.

B. Best Fit Modifié (BFM)

A chaque fois qu'un batch est ouvert, une liste contenant les tâches dans l'ordre croissant des dates de disponibilité fournit le premier élément du batch. Ensuite, l'algorithme cherche à placer la tâche qui va laisser le moins d'espace dans le batch.

Algorithme *BFM*

1. Trier les jobs en ordre croissant des dates de disponibilité r_j : L_1
2. Trier les jobs en ordre décroissant des tailles v_j : L_2
3. Tant que L_1 n'est pas vide
    - 3.1. Mettre le premier élément de L_1 : j_k dans un nouveau batch : B_k
    - 3.2. Effacer j_k des listes L_1 et L_2
    - 3.3. Pour tous les éléments j_l de L_2
        - 3.3.1. Si j_l peut entrer dans le batch B_k
            - Alors $B_k = B_k \cup j_l$
            - Effacer j_l des listes L_1 et L_2
        - Fin si
    - Fin pour

Fin tant que

4. Fixer pour chaque batch la date au plus tôt égale à la plus grande date de disponibilité des tâches qu'il contient.
5. Ordonnancer les batch consécutivement en ordre croissant des dates au plus tôt sur les machines.

En cas d'égalité des dates ou taille aux étapes 1, 2 ou 5, le choix de l'ordre des tâches ou batch est arbitraire.

Résolution de l'exemple illustratif avec *BFM* : Nous créons d'abord une première liste, L_1, qui contient les tâches en ordre croissant des dates de disponibilité. Alors dans L_1, nous avons les tâches 2, 1, 3, 4, 6, 5. La deuxième liste, L_2, contient les tâches en ordre décroissant des tailles. Alors, l'ordre des tâches dans L_2 est le suivant : 5, 2, 3, 1, 4, 6. Nous pouvons maintenant commencer à remplir le premier batch.

La liste L_1 fournit son premier élément comme le premier élément du batch 1. Alors, la tâche 2 est placée dans le batch 1 et elle est effacée des listes L_1 et L_2. Nous allons maintenant continuer à remplir le batch 1 avec les éléments de L_2. Le premier élément de la L_2 est la tâche 5 qui a la taille $3B/4$. Mais la tâche 2 dans le batch 1 a la taille $B/2$. Donc, la tâche 5 ne peut pas être placée dans ce batch. On continue avec le deuxième élément de L_2, la tâche 3, qui a la taille $B/4+\varepsilon$. Donc, on la met dans le batch 1 et on l'efface des listes L_1 et L_2. L'élément suivant dans L_2 est la tâche 1 et sa taille est $B/4$. Elle n'entre pas dans le batch 1. Les éléments restant dans L_2 sont les tâches 4 et 6. Elles ont la taille toute petite, ε. Alors, elles sont placées dans le batch 1. Puisque toute la liste L_2 est parcourue, nous fermons le batch 1.

Les éléments qui restent dans L_1 et L_2 sont les tâches 1 et 5. Le premier élément dans L_1 est la tâche 1, donc, cette tâche est placée dans un nouveau batch, *i.e.* batch 2. Il nous reste seulement la tâche 5 dans L_2 (aussi dans L_1). Quand on met cette tâche dans le batch 2, on obtient un batch 100% plein. Puisqu'il n'y a plus d'élément dans L_1 (ni dans L_2), on passe à l'étape suivante pour affecter les dates au plus tôt des batch.

La tâche dont la date de disponibilité est la plus grande dans le batch 1 est la tâche 6. Donc, la date au plus tôt du batch 1 est 80. De la même façon, la date au plus tôt du batch 2 est 100. Le batch 1 est alors affecté à la machine 1, et le batch 2 à la machine 2. Puisque l'exécution d'un batch dure 100 minutes, la valeur du C_{max} est trouvée comme 200.

La complexité de *BFM* dépend principalement de l'étape de 3. Pour chaque élément de L_1, L_2 est parcourue afin de trouver des tâches pour mettre dans un batch. Alors, pour chaque élément de L_1 de 1 à N, au pire des cas N itérations sont effectuées pour rechercher des tâches dans L_2. Donc, la complexité du pire des cas est bornée par $O(N^2)$.

C. Worst Fit Modifié (WFM)

La différence entre *WFM* et *BFM* est qu'une fois qu'un batch est ouvert, *WFM* essaie de placer d'abord les petites tâches dans ce batch, alors que *BFM* chercherait à placer la plus grande.

Algorithme *WFM*
1. Trier les jobs en ordre croissant des dates de disponibilité r_j : L_1
2. Trier les jobs en ordre croissant des tailles v_j : L_2
3. Tant que L_1 n'est pas vide
    3.1. Mettre le premier élément de L_1 : j_k dans un nouveau batch : B_k
    3.2. Effacer j_k des listes L_1 et L_2
    3.3. Pour tous les éléments j_l de L_2
        3.3.1. Si j_l peut entrer dans le batch B_k
            Alors $B_k = B_k \cup j_l$
            Effacer j_l des listes L_1 et L_2
            Fin si
    Fin pour
Fin tant que
4. Fixer pour chaque batch la date au plus tôt égale à la plus grande date de disponibilité des tâches qu'il contient.
5. Ordonnancer les batch consécutivement en ordre croissant des dates au plus tôt sur les machines.

Résolution de l'exemple illustratif avec *WFM* : Comme *WFM* est très proche de *BFM*, nous donnons directement le résultat : Nous obtenons avec *WFM* trois batch. Le premier batch contient les tâches 1, 2 4 et 6, le deuxième batch contient la tâche 3 et le batch trois contient la tâche 5. Les dates au plus tôt de ces batch sont 80, 30 et 100, respectivement. Alors, le batch 2 est affecté à la machine 1, le batch 1 à la machine 2 et le batch 3 à la machine 1. Le C_{max} obtenu est égal à 230.

En adoptant le même raisonnement que pour l'algorithme *BFM*, nous trouvons pour *WFM* une complexité de $O(N^2)$.

D. Next Fit Modifié (NFM)

A part le tri fait à l'étape 1, cet algorithme n'est pas différent de l'heuristique next fit. Quand on ouvre un batch, on essaie de le remplir avec des tâches qui se succèdent. Contrairement aux algorithmes précédents, dès qu'il y a une tâche qui n'entre pas dans le batch courant, le batch est fermé et un nouveau batch est ouvert pour la tâche suivante.

Algorithme *NFM*

1. Trier les jobs en ordre croissant des dates de disponibilité r_j : L_1
2. Ouvrir le premier batch
3. Tant que L_1 n'est pas vide
    3.1. Choisir le premier élément de L_1 et le mettre dans le batch courant. S'il n'entre pas, fermer ce batch, ouvrir un nouveau batch pour cet élément.
    3.2. Mettre à jour L_1

Fin tant que

4. Fixer pour chaque batch la date au plus tôt égale à la plus grande date de disponibilité des tâches qu'il contient.
5. Ordonnancer les batch consécutivement en ordre croissant des dates au plus tôt sur les machines.

Après avoir trié les tâches en ordre croissant des dates de disponibilité, les batch sont formés avec des tâches consécutives.

Résolution de l'exemple illustratif avec *NFM* : Pour cet exemple, le tri des tâches par rapport aux dates de disponibilité nous donne la liste des tâches suivante : 2, 1, 3, 4, 6, 5. Un batch est ouvert pour la tâche 2 dont la taille est *B/2*. La taille de la tâche suivante, *i.e*. tâche 1, est égale à *B/4*. Alors, on la met dans le premier batch aussi. Mais, il ne reste plus assez de place dans ce batch pour la tâche 3. Donc, on ferme le batch 1 et on ouvre un nouveau batch (batch 2) pour la tâche 3. Les tâches suivantes (tâches 4 et 6) ont des tailles très petites et elles entrent dans le batch 2. De cette façon, le batch a une taille égale à $B/4 + 3\,\varepsilon$. Mais la dernière tâche (tâche 5) a une taille égale à *3B/4*. Donc, le batch 2 est fermé et la tâche 5 est mise dans un nouveau batch (batch 3). Nous fixons les dates au plus tôt de ces batch grâce à l'étape 3. Alors, les dates au plus tôt des batch 1, 2 et 3 sont 20, 80 et 100, respectivement. Quant à l'affectation de ces batch aux machines par l'étape 4, nous affectons le batch 1 à la machine 1, le batch 2 à la machine 2 et enfin le batch 3 à la machine 1. La figure 4.7 montre cette solution via un diagramme de Gantt.

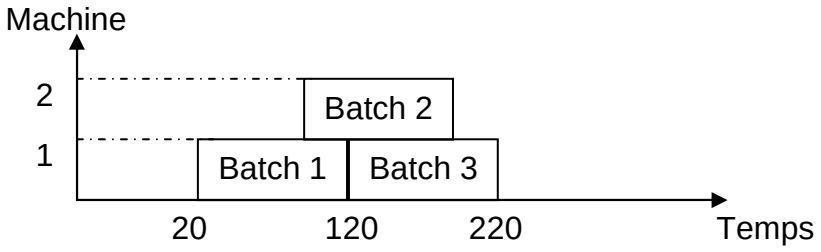

Figure 4.7. Résolution de l'exemple illustratif avec *NFM*

Notons que cet algorithme est beaucoup plus sensible aux dates de disponibilité des tâches puisque les batch sont constitués avec des tâches consécutives. La complexité de *NFM* dépend de l'étape 1 qui a une complexité de *O(NlogN)*.

4.3.2. Garantie de performance de FFM, BFM, WFM et NFM

Il est possible de calculer la garantie de performance des algorithmes présentés précédemment. A part *NFM*, les trois autres algorithmes ne sont pas très différents les uns des autres. Leur but étant la maximisation de la capacité utilisée des batch, les batch ne sont pas forcément constitués avec des tâches consécutives. Dans ce cas, le plus grand risque est de retarder l'exécution des tâches dont les dates de disponibilité sont petites. C'est-à-dire que si les tâches avec une date de disponibilité petite sont combinées avec des tâches avec une date de disponibilité assez grande, les dates au plus tôt des batch risquent d'être assez élevées. Nous pouvons expliquer ce fait sur une figure.

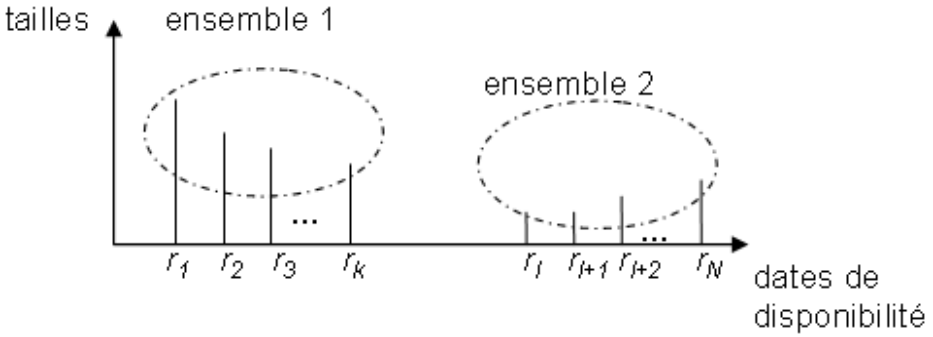

Figure 4.8. Exemple illustratif pour la création des batch avec *FFM*, *BFM* et *WFM*

Sur la figure 4.8, nous avons 2 ensembles de tâches ayant des tailles et des dates de disponibilité différentes. Pour chaque algorithme, *i.e. FFM*, *BFM* et *WFM*, l'ensemble 1 fournit le premier élément quand un batch est créé. Ensuite, en fonction de la spécificité de l'algorithme, ce batch est rempli avec des tâches qui n'ont pas encore été placées. Supposons que les tâches appartenant à l'ensemble 1 soient de grandes tailles et qu'elles ne puissent pas être placées ensembles dans un batch. Supposons également que les tâches de l'ensemble 2 soient de petites tailles. Dans ce cas, la création des batch par des algorithmes *FFM*, *BFM* et *WFM* nécessite de combiner les tâches de l'ensemble 1 avec les tâches de l'ensemble 2. Ensuite, les batch commencent à être exécutés à partir des dates de disponibilités des tâches de l'ensemble 2. Mais, si r_k est beaucoup plus petit que r_l (*cf.* figure 4.8), retarder l'exécution des tâches de l'ensemble 1 peut être très pénalisant pour le critère C_{max}. Ce cas représente en fait le pire cas des algorithmes *FFM*, *BFM* et *WFM*. Nous allons utiliser ce cas pour montrer leurs

garanties de performances. Mais d'abord commençons par une petite explication sur la garantie de performance d'un algorithme.

Nous avons 2 types de garanties de performance pour les problèmes d'optimisation [99] : garantie de performance absolue (*absolute performance guarantee* en anglais) et garantie de performance relative (*relative performance guarantee* en anglais). Pour une instance I d'un problème d'optimisation, notons la solution optimale par Opt(I) et la solution trouvée par l'algorithme A par A(I).

Définition 1. ([99]) Un algorithme d'approximation absolue est un algorithme polynomial tel que pour une constante $k > 0$, et pour toute instance I, on a :

$$|A(I) - Opt(I)| \leq k$$

Pour ce qui concerne la garantie de performance relative, elle est désignée par un coefficient appelé « ratio de performance ».

Définition 2. ([99]) Soit A un algorithme quelconque pour un problème d'optimisation. Pour un problème de minimisation, le ratio de performance de A, $R_A(I)$, de l'algorithme A pour une instance I est défini par

$$R_A(I) = A(I) / Opt(I)$$

avec $A(I)$ la valeur trouvée par l'algorithme A pour l'instance I. Si nous avons un problème de maximisation, Le ratio de performance, $R_A(I)$, est définie par

$$R_A(I) = Opt(I) / A(I)$$

Définition 3. ([54]) La garantie de performance relative, R_A, d'un algorithme d'approximation A est définie par

$$R_A = \min \{c \geq 1 \mid R_A(I) \leq c \text{ pour toute instance } I\}$$

et la garantie de performance relative asymptotique

$$R_A^{\infty} = \min \{c \geq 1 \mid \exists n \in Z^+, R_A(I) \leq c \text{ pour toute instance } I \text{ t.q. } Opt(I) \geq n\}$$

En effet, la garantie de performance relative asymptotique d'un algorithme est calculé pour le cas où la valeur de la solution optimale tend vers l'infinie. Dans l'ensemble ci-dessus, n représente un tres grand nombre.

Si le contraire n'est pas précisé, nous parlerons de garantie de performance relative dans notre étude.

Afin de calculer les garanties de performances des algorithmes *FFM, BFM et WFM,* nous avons besoin de déterminer le nombre de batch qu'ils créent dans le pire des cas. Le lemme suivant nous donne des informations sur ce sujet.

Lemme 4. La garantie de performance de *FFM, BFM* et *WFM* dans un objectif de minimisation du nombre de batch, est égale à 1,7**nb* +2 où *nb* représente le nombre optimal de batch.

Preuve. Nous savons que les algorithmes classiques de bin packing, *i.e. FF, BF* et *WF*, ont une garantie de performance de 1,7**nb* + 2 pour le nombre de batch créés quel que soit l'ordre des tâches en entrée [25]. Concernant le nombre de batch créés, les algorithmes *FFM, BFM* et *WFM* ont la même garantie de performance. En effet, considérons d'abord *FFM*. La différence entre *FFM* et *FF* est que *FFM* fait d'abord un tri des tâches, ensuite *FF* est appliqué sur les tâches, alors que pour *FF* on utilise une liste triée arbitrairement. Mais, la liste triée arbitrairement peut tout à fait être égale à une liste contenant des tâches en ordre croissant de dates de disponibilité. Dans ce cas, *FFM* devient équivalent à *FF*. La garantie de performance est alors la même pour le nombre de batch créés.

La même observation est valide pour *BFM* et *WFM*. Après avoir créé une liste avec des tâches en ordre croissant des dates de disponibilité, cette liste fournit le premier élément de chaque batch. Ensuite, lorsque l'on trie le reste des tâches en ordre croissant ou décroissant des tailles afin de remplir les batch, on obtient des cas particuliers d'une liste des tâches arbitrairement ordonnées. Donc, *BFM* et *WFM* deviennent aussi équivalents à *FF* quand on a pour objectif le nombre de batch formés. Leur garantie de performance pour le nombre de batch créés est alors la même que celle de *FF*. ■

Théorème 1. La garantie de performance des algorithmes *FFM, BFM* et *WFM* pour le problème 1 | *p-batch*, r_j, p_j=*p*, v_j, *B* | C_{max} est 3,2.

Preuve. Le nombre de batch formé par les heuristiques algorithmes *FFM, BFM* et *WFM* est borné par 1,7*nb*+2 avec *nb* le nombre minimal de batch. Notons r_N la dernière date de disponibilité des tâches et $C_{\max}^{A}$ la solution trouvé avec les heuristiques proposées. Evidemment dans le pire cas, l'exécution des batch commence à l'instant r_N et tous les batch sont exécutés à partir de cet instant. On obtient donc

$$C_{\max}^{A} \leq r_N + 1{,}7 * nb * p + 2p$$

Notons $C^*_{\max}$ la solution optimale. Il est clair que le début d'exécution du dernier batch dans la meilleure solution est égal à la date de disponibilité de la dernière tâche, *i.e.* à r_N. Il y aura au moins un batch qui ne pourra pas commencer avant r_N. Alors,

$$C^*_{\max} \geq r_N + p$$

De plus, on sait que le nombre minimal de batch est *nb* et donc,

$$C^*_{\max} \geq nb * p\,.$$

En utilisant deux dernières inégalités, nous pouvons obtenir pour $C^A_{\max}$

$$C^A_{\max} \leq C^*_{\max} + 1{,}7C^*_{\max} + p$$
$$\Leftrightarrow\ C^A_{\max} \leq 2{,}7C^*_{\max} + p$$

Si dans la solution optimale, nous n'avons qu'un seul batch, cela signifie que toutes les tâches peuvent entrer dans un batch. Et l'exécution de ce batch commence à l'instant r_N. Si toutes les tâches peuvent entrer dans un batch, *FFM*, *BFM* et *WFM* sont capables de fournir la solution optimale. Mais si dans la solution optimale, nous avons au moins deux batch, on a $C^*_{\max} \geq 2p$. Alors,

$$C^A_{\max} / C^*_{\max} \leq 2{,}7 + p/2p$$
$$\Leftrightarrow C^A_{\max} / C^*_{\max} \leq 3{,}2\ \blacksquare$$

Théorème 2. La garantie de performance des algorithmes *FFM*, *BFM* et *WFM* pour le problème $P \mid p\text{-}batch, r_j, p_j{=}p, v_j, B \mid C_{max}$ est 3.

Preuve. Dans le pire cas des algorithmes *FFM*, *BFM* et *WFM*, nous avons encore 1,7**nb* + 2 batch et commençons l'exécution de ces batch à partir de l'instant r_N. Cette fois-ci, on a *M* machines dans le problème avec *M*>1. Puisque ces batch sont ordonnancés consécutivement sur les machines, le nombre maximal de batch par machine est borné par $\lceil (1{,}7nb+2)/M \rceil$. Donc, la valeur de $C^A_{\max}$ est bornée par :

$$C^A_{\max} \leq r_N + \lceil (1{,}7nb+2)/M \rceil * p$$

$C^*_{\max}$ étant la solution optimale, il est clair que $C^*_{\max} \geq r_N + p$ et que $C^*_{\max} \geq \lceil nb/M \rceil * p$. Nous pouvons utiliser ces inégalités pour exprimer la valeur qui borne $C^A_{\max}$.

$$C^A_{\max} \leq r_N + \lceil (1{,}7nb+2)/M \rceil * p$$

$$\Rightarrow C_{\max}^{A} \leq r_N + \lceil 1{,}7nb/M \rceil * p + \lceil 2/M \rceil * p$$

Puisque $\lceil 2/M \rceil * p \leq p$,

$$C_{\max}^{A} \leq C_{\max}^{*} + \lceil (1{,}7nb/M \rceil * p$$
$$\Rightarrow C_{\max}^{A} \leq C_{\max}^{*} + \lceil 1{,}7 \rceil * \lceil nb/M \rceil * p$$

En utilisant le fait que $C_{\max}^{*} \geq \lceil nb/M \rceil * p$, on a

$$C_{\max}^{A} \leq C_{\max}^{*} + \lceil 1{,}7 \rceil * C_{\max}^{*}$$
$$\Rightarrow C_{\max}^{A} \leq C_{\max}^{*} + 2C_{\max}^{*}$$
$$\Rightarrow C_{\max}^{A} \leq 3C_{\max}^{*} \blacksquare$$

L'algorithme *NFM* est un peu différent des trois premiers. Les batch sont formés seulement avec des tâches consécutives, et donc, *NFM* est plus sensible aux dates de disponibilité des taches. Il est facile de voir que la garantie de performance de *NFM* pour la minimisation de nombre de batch est la même que la garantie de performance de *NF*, et qui est égale à $2nb$-1 avec nb le nombre minimal de batch.

Lemme 5. Soit I une instance et I_k une sous-instance de I, *i.e.* $I_k \subset I$. Si nb_k est le nombre minimal de batch que l'on peut créer avec les tâches de I_k, *NFM* crée $2nb_k$-1 batch dans son pire cas avec les tâches de I_k.

Preuve. Rappelons que *NFM* crée des batch avec des tâches consécutives. L'application de *NFM* sur I_k va encore créer des batch avec des tâches consécutives. Donc, la même garantie de performance est valide pour chaque sous-instance de I. ■

Théorème 3. La garantie de performance de l'algorithme *NFM* pour le problème 1 | *p-batch*, r_j, p_j=p, v_j, B | C_{max} est 2.

Preuve. Si l'exécution du dernier batch commence à l'instant r_N, avec r_N étant la date de disponibilité de la dernière tâche, alors le C_{max} trouvé par *NFM* est optimal. Sinon, il y a une chaîne de batch sans temps mort tel que le dernier batch exécuté dans cette chaîne détermine la valeur du C_{max}. Notons qu'il peut y avoir (ou non) d'autres batch exécutés avant cette chaîne de batch. Un exemple pour le cas où d'autres batch sont exécutés avant la dernière chaîne de batch est montré sur la figure 4.9.

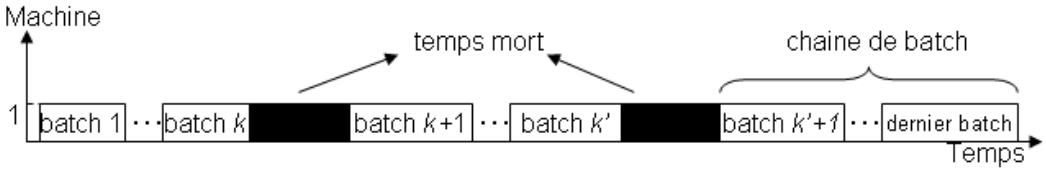

Figure 4.9. Une chaîne de batch sans temps mort sur une machine

Nous référons le batch *k'*+1 dans la figure 4.9. comme le premier batch de la chaîne de batch. La date de début du premier batch de cette chaîne est forcément égale à la date de disponibilité d'une tâche, par exemple la tâche *l*, et soit r_l la date de disponibilité de cette tâche. La dernière tâche mise dans le premier batch de la chaîne est donc la tâche *l*. Il peut y avoir (ou non) d'autres tâches mises ce batch avant la tâche *l*. La date de disponibilité de ces tâches est inférieure ou égale à r_l.

Soit *l'* la première tâche qui n'entre pas dans le premier batch de la chaîne et soit $r_{l'}$ sa date de disponibilité. Evidemment, $r_{l'} \geq r_l$. Notons par $nb_{l'}$ le nombre minimal de batch qui peut être créé par les tâches exécutées dans les batch *k'*+1, *k'*+2, *k'*+3, …, dernier batch. En excluant les tâches du batch *k'*+1, nous pouvons alors créer au moins $nb_{l'} - 1$ batch avec les tâches exécutées dans les batch *k'*+2, *k'*+3, …, dernier batch. Notons par $C^*_{\max}$, le C_{max} optimal. Alors,

$$C^*_{\max} \geq r_{l'} + (nb_l - 1) * p \geq r_l + (nb_l - 1) * p .$$

D'après le lemme 5, dans son pire cas, *NFM* va créer $2nb_{l'} - 1$ batch avec les tâches exécutées dans les batch *k'*+1, *k'*+2, *k'*+3, …, dernier batch. Puisque l'exécution du premier batch de la chîne commence avec r_l, l'exécution d'un nombre maximal de batch nous donne une borne supérieur pour le C_{max}. Notons $C^{NFM}_{\max}$ la valeur trouvée par *NFM*. Alors,

$$C^{NFM}_{\max} \leq r_l + (2nb_l - 1) * p$$

$$=> C^{NFM}_{\max} \leq C^*_{\max} + nb_l * p$$

Puisque $C^*_{\max} \geq nb_l * p$,

$$C^{NFM}_{\max} \leq 2C^*_{\max} \blacksquare$$

Pour démontrer la garantie de performance de *NFM* pour le cas de plusieurs machines, nous avons besoin d'un autre lemme.

Lemme 6. Soit un problème avec N tâches et M machines. La valeur minimale de la durée totale d'exécution après la tâche j, avec $1 \leq j \leq N$, est égale à la somme de la date de disponibilité de j et de la durée d'exécution d'un nombre minimal de batch affectés sur la même machine que j.

Preuve. Pour un problème quelconque avec N tâches et M machines, le nombre minimal de batch, noté nb, est donné par : $nb = \left\lceil \sum_{j=1}^{N} v_j / B \right\rceil$ avec v_j la taille de la tâche j et B la capacité de chaque machine. Soit les tâches triées en ordre croissant des dates de disponibilité. Considérons une tâche j telle que $1 \leq j \leq N$ et $r_j \leq r_N$. Il est possible de calculer une borne inférieure pour le nombre de batch que l'on peut créer avec les tâches dont la date de disponibilité est supérieure ou égale à r_j. Après la tâche j (j inclus), un nombre minimal de batch, noté nb_j, peut être calculé par : $nb_j = \left\lceil \sum_{k \in S} v_k / B \right\rceil$ où $S = \{k / r_k \geq r_j\}$. De cette façon, nous ne prenons pas en compte les tâches dont la date de disponibilité est inférieure à r_j.

Puisque tous les batch ont la même durée d'exécution, p, ils sont affectés consécutivement aux machines en ordre croissant de leurs date de disponibilité (la date de disponibilité d'un batch est égale à la plus grande date de disponibilité des tâches qu'il contient). Alors, les nb_j batch doivent être placés consécutivement sur M machines en ordre croissant de leur date de disponibilité. Dans ce cas, chaque machine peut contenir au plus $\left\lceil nb_j / M \right\rceil$ batch. De plus, dans le meilleur cas, l'exécution du premier batch parmi les nb_j batch va commencer à l'instant r_j. Alors, la valeur minimale du temps total d'exécution après la tâche j peut être donnée par : $r_j + \left\lceil nb_j / M \right\rceil * p$. En d'autres termes, la valeur minimale de la durée totale d'exécution après la tâche j est égale à la somme de la date de disponibilité de j et de la durée d'exécution d'un nombre minimal de batch affecté à la même machine que j. La valeur de C_{max} ainsi trouvée devient aussi une borne inférieure. ■

Théorème 4. La garantie de performance de l'algorithme *NFM* pour le problème $P \mid p\text{-}batch, r_j, p_j=p, v_j, B \mid C_{max}$ est 2.

Preuve. Nous allons utiliser le même raisonnement que la preuve précédente. Soient M le nombre de machines et M' ($1 \leq M' \leq M$) l'indice de la machine sur laquelle le dernier batch est exécuté. Si l'exécution du dernier batch commence à l'instant r_N, *i.e.* la dernière date de disponibilité des tâches, alors, le C_{max} est optimal. Sinon, il y a une chaîne de batch sur la machine M' sans temps mort.

Notons $b_{m'}$ le premier batch de la chaine sur la machine M'. Supposons que le dernier batch de exécuté sur la machine M' détermine la valeur du C_{max}.

Puisque nous avons *M* machines et que les batch sont affectés consécutivement, le deuxième batch affecté à la machine *M'* après le batch $b_{m'}$ est le batch d'indice $b_{m'+M}$. En effet, entre les batch $b_{m'}$ et $b_{m'+M}$, il y a *M*-1 batch affectés aux machines *M'*+1, *M'*+2,…, *M*, 1, …, *M'*-1, respectivement. Notons $b_{d'}$ le dernier batch affecté à la machine *M'*. Pour mieux comprendre la procédure d'affectation des batch par *NFM*, une illustration est donnée dans la figure 4.10.

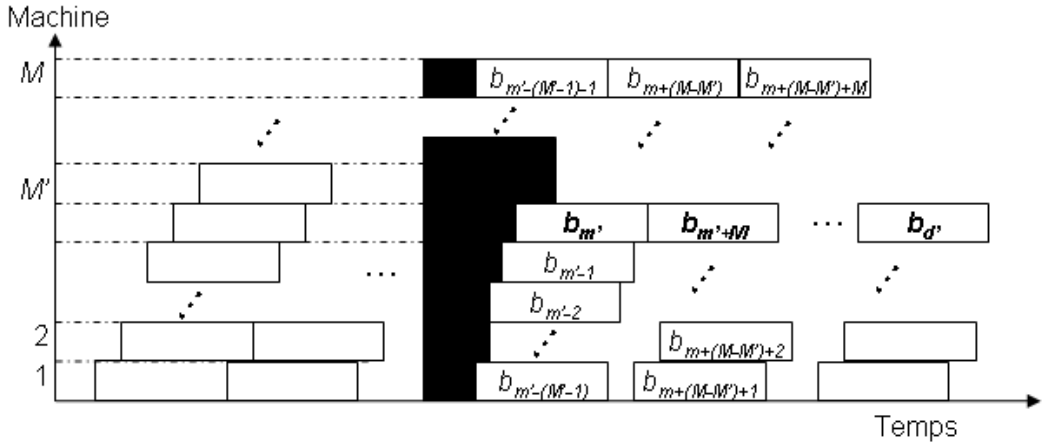

Figure 4.10. Une chaine de batch sans temps mort sur plusieurs machines

La date de début au plus tôt du premier batch de la chaine de batch sur la machine *M'* est forcément égale à la date de disponibilité d'une tâche, par exemple la tâche *l'* avec $r_{l'}$ sa date de disponibilité. Il peut y avoir d'autres tâches dans le batch $b_{m'}$ dont la date de disponibilité est inférieure ou égale à $r_{l'}$. Considérons seulement les tâches exécutées dans les batch $b_{m'+1}$, $b_{m'+2}$, …, $b_{d'}$. Notons $nb_{l'}$ le nombre minimal de batch que l'on peut créer avec ces tâches. Alors, le nombre minimal de batch que l'on peut obtenir avec les tâches exécutées dans les batch $b_{m'+1}$, …, $b_{d'}$ devient égal à $nb_{l'}$-1. Par le lemme 6, une borne inférieure pour le C_{max} optimal est comme le suivant :

$$C^*_{\max} \geq r_{l'} + \lceil (nb_{l'} - 1/M \rceil * p$$

D'après le lemme 5, *NFM* va créer $2nb_{l'}$-1 batch avec les tâches exécutées dans les batch $b_{m'+1}$, $b_{m'+2}$, …, $b_{d'}$. dans son pire cas. Notons $C^{NFM}_{\max}$ la valeur trouvée par *NFM*. Puisque nous créons un nombre maximal de batch, nous pouvons l'utiliser pour donner une borne supérieure pour $C^{NFM}_{\max}$. Nous obtenons donc

$$C^{NFM}_{\max} \leq r_{l'} + \lceil (2nb_{l'} - 1)/M \rceil * p$$

En composant le terme $\lceil (2nb_{l'} - 1)/M \rceil$, on a

$$C_{\max}^{NFM} \leq r_{l'} + \lceil (nb_{l'} - 1)/M \rceil * p + \lceil nb_{l'}/M \rceil * p$$

On sait que $C_{\max}^{*} \geq r_{l'} + \lceil (nb_{l'} - 1)/M \rceil * p$ et que $C_{\max}^{*} \geq \lceil nb_{l'}/M \rceil * p$. Alors,

$$C_{\max}^{NFM} \leq 2C_{\max}^{*} \blacksquare$$

4.4 Algorithmes optimaux pour deux cas particuliers

Dans cette section, nous étudions 2 cas particuliers et développons 2 algorithmes optimaux pour chacun des cas. Le premier cas suppose qu'il est possible de couper les tâches. Le deuxième cas considère une relation particulière entre les tailles des tâches.

4.4.1 Cas particulier où il est permis de couper les tâches

Dans cette section, nous étudions un cas particulier noté par $P|p\text{-}batch,r_j,p_j=p,v_j(split),B|C_{max}$. Le mot « split » rajouté à la notation du problème général signifie que les tâches peuvent être coupées. Notons que quand une tâche est coupée en deux, on obtient 2 tâches dont les dates de disponibilité et les durées d'exécution sont égales. Et bien entendu, la somme de leurs tailles donne la taille de la tâche originale. Notons que le problème avec des tâches de taille unitaire est une façon particulière de couper les tâches. Dans ce cas particulier, nous supposons que les tâches sont composées de sous-tâches de taille unitaire, et la somme des tailles de toutes les sous-tâches compose la tâche principale. De plus, toutes les sous-tâches ont la même date de disponibilité.

Le problème qui considère les tâches de taille unitaire en présence des dates de disponibilité différentes a été étudié par Ikura et Gimple [56]. Leur objectif est la minimisation du C_{max} sur une seule machine. Nous généralisons ce cas sur plusieurs machines et modifions un peu l'algorithme proposé par Ikura et Gimple [56]. Cette modification consiste à tout simplement affecter les batch consécutivement sur les machines en fonction du lemme 6.

L'algorithme proposé ici commence par créer une liste, *L*, des tâches triées en ordre décroissant des dates de disponibilité. Puis, une borne inférieure est calculée pour le nombre de batch que l'on peut créer avec les tâches de la liste *L* (cette borne inférieure est calculée en divisant la somme de toutes les tailles par la capacité des machines). Après avoir trouvé le nombre minimum de batch, on commence à remplir les batch en commençant par le premier élément de la liste *L*.

Un batch est fermé si et seulement s'il est 100% plein ou si toutes les tâches sont placées dans un batch. Si une tâche n'entre pas complètement dans un batch, elle est coupée, sa première partie est mise dans le batch (comme cela on obtient un batch 100% plein) et sa deuxième partie est traitée comme une nouvelle tâche ayant la même date de disponibilité que la tâche non-coupée. Enfin, les batch sont triés en ordre croissant des dates au plus tôt sur les machines.

Algorithme « *Split Job* »
1. Trier les tâches en ordre décroissant des dates de disponibilité : L
2. Calculer le nombre minimum de batch, nb, par la formule $\left\lceil \sum_{j=1}^{N} v_j / B \right\rceil$
3. Tant que $nb > 0$, ouvrir un batch : b_m
    3.1. Tant que b_m n'est pas 100% plein ou la liste L n'est pas vide,
        3.1.1. Mettre le premier élément de L dans le batch b_m. Mettre à jour L. Si le batch est 100% plein ou s'il n'y a plus d'élément dans L, fermer le batch et poser $nb = nb$ -1. Si la tâche n'entre pas complètement dans le batch, la couper. Mettre sa première partie dans le batch afin d'avoir un batch plein. Fermer le batch et poser $nb = nb - 1$. Mettre à jour la taille de la tâche coupée.
    Fin tant que
Fin tant que
4. Fixer pour chaque batch la date au plus tôt égale à la plus grande date de disponibilité des tâches qu'il contient.
5. Ordonnancer les batch consécutivement en ordre croissant des dates au plus tôt sur les machines.

Illustrons l'algorithme sur un exemple.

Exemple illustratif : Soient 3 machines de capacité 5. Soient 8 tâches, numérotées de 1 à 8, de tailles 3, 4, 2, 4, 1, 2, 4.5, 3 et de dates de disponibilité 10, 20, 40, 45, 50, 80, 100, 100. La durée d'exécution d'un batch est égale à 100.

Résolution de l'exemple : Après avoir trié les tâches en ordre décroissant des dates de disponibilité, on obtient la liste L contenant les tâche en ordre suivant : 8, 7, 6, 5, 4, 3, 2, 1. Les tâches 7 et 8 ont la même date de disponibilité, nous faisons notre choix arbitrairement.

Nous calculons ensuite la borne inférieure sur le nombre de batch : $\lceil (3+4+2+4+1+2+4.5+3)/5 \rceil = 5$ batch. Pour l'instant, tous ces batch sont vides. En commençant par le premier élément de la liste L, *i.e.* la tâche 8, on va remplir le premier batch. On met la tâche 8 dans le batch 1. La taille du batch est maintenant 3. La tâche 7 a une taille égale à 4.5, donc, elle ne peut pas entrer complètement dans le batch 1. Alors, la tâche 7 est coupée. Nous la coupons en

deux : la première partie a la taille 2 et la deuxième partie a la taille 2.5. Notons par 7_1 la première partie de la tâche 7 et par 7_2 sa deuxième partie. Nous mettons 7_1 dans le batch 1 afin d'avoir un batch 100% plein.

A chaque fois qu'une tâche est mise dans un batch, la liste *L* est mise à jour. Alors, après avoir placé les tâches 8 et 7_1 dans le batch 1, l'ordre des tâches qui reste dans *L* est : 7_2, 6, 5, 4, 3, 2, 1. On remplit maintenant le batch 2 avec les tâches 7_2, 6, 5_1. La tâche 5 est coupée en deux : 5_1 et 5_2 sont de même taille égale à 0.5. Le troisième batch contient les tâches 5_2, 4, 3_1, le batch 4 contient les tâches 3_2 et 2_1 et enfin les tâches 2_2 et 1 sont dans le batch 5.

Affectons une date au plus tôt à chaque batch. Nous obtenons donc pour les batch de 1 à 5 les dates au plus tôt 100, 100, 50, 40, 20. Enfin, nous affectons ces batch sur les machines par ordre croissant des dates au plus tôt. La figure 4.11 montre la solution finale.

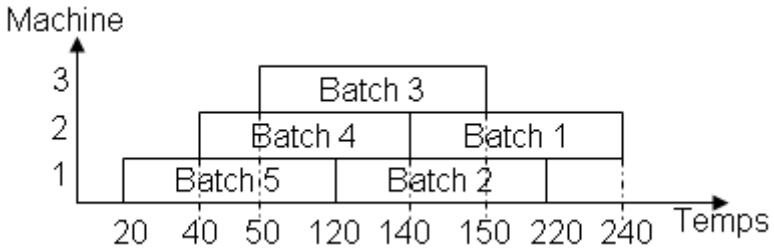

Figure 4.11. Résolution de l'exemple illustratif avec l'algorithme *split job*

Après avoir trié les tâches à l'étape 1, les batch sont créés avec des tâches consécutives. *N* étant le nombre de tâches, si chaque tâche est coupée, dans le pire cas, nous avons besoin de 2*N* itérations pour placer ces tâches dans les batch. Alors, la complexité de la procédure de création de batch est *0(N)*. La complexité de *split job* dépend donc de l'étape de tri qui est *0(NlogN)*.

Théorème 5. *Split job* est optimal pour le problème :
$P \mid p\text{-}batch, r_j, p_j=p, v_j(split), B \mid C_{max}$.

Preuve. L'étape 2 de l'algorithme calcule la borne inférieure sur le nombre de batch. La condition de fermer un batch est que, soit il est 100% plein, soit il n'y a plus de tâche dans la liste *L*. Cela implique que tous les batch sont pleins sauf, peut être, le dernier batch créé.

Maintenant, comme dans la preuve du théorème 4, considérons une chaîne de batch sans temps mort sur la machine contenant le dernier batch. Alors, la date au plus tôt du premier batch de cette chaîne est nécessairement égale à la date de

disponibilité d'une tâche, disons la tâche *k*. Puisque le reste des batch sont 100% plein, un nombre minimal de batch est créé après la tâche *k*. De plus, les batch sont affectés consécutivement sur les machines. Donc, le nombre de batch affectés après la tâche *k* sur la même machine est aussi minimal. Ainsi d'après le lemme 6, la valeur minimale du temps total d'exécution après la tâche *k* est atteinte et donc le C_{max} trouvé par *split job* est optimal. ■

Théorème 6. *Split job* fournit une borne inférieure pour le problème :
$P \mid p\text{-}batch, r_j, p_j=p, v_j, B \mid C_{max}$.

Preuve. La borne inférieure expliquée dans le lemme 6 est atteint avec l'algorithme *Split job*. Ainsi, la solution trouvée est une borne inférieure. ■

4.4.2 Cas particulier où les tailles des tâches sont « fortement divisibles »

Le cas particulier étudié ici est inspiré d'un cas particulier de bin packing où les tailles des tâches constituent un ordre fortement divisible (*strongly divisible sequence* en anglais). Notons ce problème :
$P \mid p\text{-}batch, r_j, p_j=p, v_j(fortement\ divisible), B \mid C_{max}$.

Le problème de bin packing est un problème *NP-dur* au sens fort [25]. Coffman et al. [24] ont montré que si la taille des tâches formait un ordre fortement divisible, les algorithmes First Fit (*FF*) et First Fit Decreasing (*FFD*) trouvent la solution optimale. L'algorithme *FF* a déjà été étudié dans les sections précédentes. Nous allons maintenant rappeler le principe de l'algorithme *FFD* et donner la définition d'un ordre fortement divisible.

FFD
1. Trier les tâches en ordre décroissant de leurs tailles
2. Appliquer *FF*

Ordre Fortement Divisible (Coffman et al. [24])

Soit *V* une liste contenant les différentes tailles des tâches du problème. Les tailles sont telles que $v_1 > v_2 > ... > v_i > v_{i+1} > ...$ Les tailles contenues dans la liste *V* constituent un ordre divisible si, pour tout *i*, v_{i+1} est un diviseur de v_i. Cet ordre est fortement divisible si, de plus, v_1 est un diviseur de la capacité du batch.

Avant de donner l'algorithme optimal pour le cas des tâches constituant un ordre fortement divisible, nous allons présenter quelques lemmes utilisés dans la procédure de démonstration.

Lemme 7. (Coffman et al. [24]) Si les tailles des tâches forment un ordre fortement divisibles, alors *FFD* ouvre un nouveau batch si et seulement si les batch précédents sont 100% pleins.

Le lemme ci-dessus a été démontré par Coffman et al. [24]. Donc, dans le cas où on a des tailles fortement divisibles, appliquer *FFD* permet de trouver la solution optimale du problème de bin packing, solution composée de batch %100 pleins sauf, peut être, le dernier batch.

Lemme 8. Soit une paire fortement divisible (*I*, *B*) où *I* est l'instance qui représente les tâches, et *B* la capacité des batch. Soit b_1 un batch partiellement rempli. Soit une tâche *j* de taille v_j qui n'entre pas complètement dans b_1. Alors, dans b_1, il y a au moins une tâche de taille inférieure à v_j.

Preuve. Soit b_1 un batch partiellement rempli. Soit v_{somme} la somme des tailles des tâches de b_1. Puisque chaque taille peut être exprimée comme la multiplication d'une plus petite taille (par exemple v_j) avec un nombre entier, on a $v_{somme}=coeff_1*v_j$ où $coeff_1$ est un entier. Soit *B* la capacité du batch. On sait que, comme toute les tailles, v_j est aussi un diviseur de la capacité du batch. Alors, la capacité *B* peut être exprimée comme suit : $B=coeff_2* v_j$ où $coeff_2$ est un entier. La place disponible dans b_1 est $B-v_{somme}=coeff_2*v_j - coeff_1*v_j \Leftrightarrow v_j(coeff2-coeff1)$. Si $coeff_1 = coeff_2$, alors le batch est 100% plein. Sinon, il y a assez d'espace pour la tâche *j* dans le batch b_1. Ainsi, si b_1 est un batch partiellement rempli dans lequel une tâche, *j*, ne peut pas être insérée, alors, il doit y avoir au moins une tâche dans b_1 dont la taille est inférieure à la taille de la tâche *j*. ■

Lemme 9. Soit une paire fortement divisible (*I*, *B*) où *I* est l'instance qui représente les tâches et *B* la capacité des batch. Soit b_1 un batch partiellement rempli. Soit une tâche *j* de taille v_j qui n'entre pas complètement dans b_1. Le batch b_1 peut être complètement rempli en retirant quelques tâches dont les tailles sont plus petites que v_j, et en mettant *j* dans b_1.

Preuve. Les tâches mises dans b_1 peuvent être classées dans 2 groupes : groupe 1 pour les tâches dont les tailles sont plus grandes ou égale à v_j, groupe 2 pour les tâches dont les tailles sont plus petites que v_j. Evidemment, $\sum_{l \in S} v_l = v_j*coeff$ avec S={*l* | *l* ∈ groupe 1}, *coeff* un entier et v_j la taille de la tâche *j* qu'on veut insérer dans le batch b_1. Les tâches du batch b_1 peuvent être facilement triées en ordre décroissant de leurs tailles. Après ce tri, on insère d'abord les tâches du groupe 1, ensuite les tâches du groupe 2 dans le batch b_1. *B* étant la capacité du batch, nous pouvons l'exprimer comme le suivant :

$$B = \sum_{l \in S} v_l + \sum_{l' \in S'} v_{l'} + place\_disponible$$

avec $S = \{l \mid \forall\ l \in$ groupe 1$\}$, $S' = \{l' \mid \forall\ l' \in$ groupe2$\}$ et *place_disponible* est le volume d'espace disponible dans b_1. Ou encore,

$$B = v_j * coeff + \sum_{l' \in S'} v_{l'} + place\_disponible.$$

Evidemment, $place\_disponible < v_j$. De plus, il y a un ensemble S'' de tâches dans le groupe 2 tel que $S'' = \{\ l'' \mid \exists\ l'' \in$ groupe2, $\sum_{\forall l''} v_{l''} + place\_disponible = v_j\}$. Une fois que nous trouvons les tâches de l'ensemble S'', les sortir du batch b_1 va générer un espace assez pour le placement de la tâche k dans b_1.

Supposons que le batch b_1 est composé de « sous-batch » de tailles égales à v_j. Alors, après le tri des tâches en ordre décroissant de leurs tailles, on obtient un certain nombre de sous-batch 100% pleins et 1 sous-batch pas 100% plein. Puisque la taille de chaque sous-batch est égale à v_j, le dernier sous-batch, qui n'est pas 100% plein, contient les tâches de l'ensemble S''. Alors, afin de remplir le batch b_1 complètement en mettant la tâche j dedans, il faut tout simplement enlever les tâches dans le dernier sous-batch, *i.e.* les tâches de l'ensemble S''. De cette façon, on crée un espace égale à v_j, et, la tâche j peut être mise dans le batch b_1.■

Exemple : Soit un batch de capacité 1, qui contient 5 tâches de tailles respectivement 1/8, 1/8, ½, 1/16 et 1/8. Supposons qu'on veuille remplir ce batch en plaçant une 6ième tâche de taille ¼. Alors, d'après ce qui est expliqué dans la preuve du lemme 6, le batch est composé des sous-batch de taille ¼. Ensuite, quand on trie les tâches en ordre décroissant des tailles, on obtient 3 sous-batch 100% pleins et un sous-batch partiellement plein (*cf.* figure 4.12). Il est maintenant facile de trouver les tâches à enlever afin de placer la tâche 6 dans le batch, il s'agit d'une tâche de taille 1/16 et d'une tâche de taille 1/8.

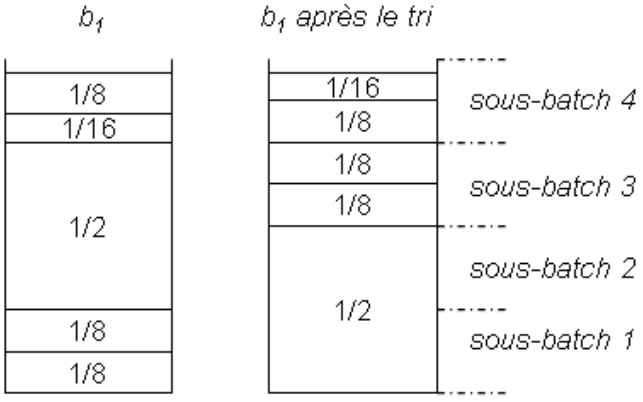

Figure 4.12. Batch b_1 avant et après le tri

Nous donnons maintenant les différentes étapes de l'algorithme optimal pour le cas des tâches fortement divisibles. Après avoir trié les tâches en ordre croissant des dates de disponibilité, l'algorithme calcule le nombre minimal de batch à créer à l'aide de l'algorithme *FF*. Ensuite, il commence à placer les tâches dans les batch. A chaque fois qu'une tâche est placée dans un batch, le nombre de batch restant à former est calculé.

Si, après avoir placé une tâche, le nombre de batch à former ne diminue pas, alors on ne ferme pas le batch et on continue avec l'élément suivant. Sinon, on ferme le batch. Si une tâche n'entre pas complètement dans un batch partiellement rempli, alors, on applique la procédure expliquée dans l'exemple illustratif du lemme 9 afin d'avoir un batch plein.

L'algorithme contient de nouvelles notations. Présentons-les avant de passer à l'algorithme.

L	liste contenant les tâches en ordre croissant des dates de disponibilité
nb_{old}	nombre minimum de batch qui peuvent être formés avec les tâches de L
j_{first}	première tâche dans L
b_b	batch actuellement ouvert
nb_{new}	nouveau nombre minimum de batch qui peuvent être formés avec les tâches de L après avoir enlevé j_{first} de L
space	dans le cas où une tâche, disons j, n'entre pas entièrement dans un batch partiellement rempli, *space* représente l'espace nécessaire pour que la tâche j soit placée entièrement dans le batch
j_{last}	dernière tâche dans le batch b_b après avoir trié les tâches du batch en ordre décroissant des tailles
v_{last}	taille de la tâche j_{last}

Donnons un exemple numérique pour mieux comprendre l'algorithme qui est présenté entièrement dans la page suivante.

Exemple illustratif : Soient 8 tâches de tailles 4, 1, 2, 1, 1, 4, 2, 4 et de dates de disponibilité 0, 5, 10, 10, 15, 15, 20, 60, respectivement. Soient 2 machines de capacité 8. La durée d'exécution d'un batch est égale à 100.

Résolution de l'exemple : Numérotons les tâches de 1 à 8 d'après l'ordre croissant des dates de disponibilité. De cette façon, nous créons la liste L. On calcule ensuite le nombre minimal de batch. En appliquant l'algorithme *FF*, nous trouvons un nombre minimal de batch égal à 3, donc $nb_{old} = 3$.

Algorithme « Séquence Fortement Divisible (*SFD*) »

1. Trier les tâches en ordre croissant des dates de disponibilité : L

2. Tant que L n'est pas vide, appliquer *FF* sur L afin de calculer le nombre minimum de batch, nb_{old}. Mettre la première tâche, $j_{first,}$ de L dans un nouveau batch, b_b. Effacer cette tâche de L et réappliquer *FF* sur L afin de trouver le nouveau nombre minimum de batch : nb_{new}. Si $nb_{old} > nb_{new}$, fermer le batch, Sinon,

    2.1. Tant que $nb_{old} = nb_{new}$, si la première tâche, j_{first}, de L entre dans le batch b_b, la mettre dedans. Mettre à jour L et réappliquer *FF* sur L pour calculer nb_{new}.

        2.1.1. Si la tâche, j_{first}, n'entre pas dans le batch b_b, calculer l'espace nécessaire, *space*, afin de placer cette tâche dans le batch. Trier les tâches de b_b dans un ordre décroissant des tailles.

            2.1.1.1. Tant que $space > 0$, enlever la dernière tâche, j_{last}, de b_b. Poser $space = space - v_{last}$. Mettre j_{last} dans L
            Fin tant que
            2.1.1.2. Mettre j_{first} dans b_b, l'effacer de L
            2.1.1.3. Poser $nb_{new} = nb_{new} - 1$
            2.1.1.4. Trier les tâches de L en ordre croissant de dates de disponibilité

        Fin si

    Fin tant que

    2.2. Fermer le batch

Fin tant que

3. Fixer pour chaque batch la date au plus tôt égale à la plus grande date de disponibilité des tâches qu'il contient.

4. Ordonnancer les batch consécutivement en ordre croissant des dates au plus tôt sur les machines.

Maintenant, on ouvre un batch, batch numéro 1, et on commence à le remplir avec la première tâche de L_1. La première tâche a une taille égale à 4. Après l'avoir placée dans le batch 1, nous l'enlevons de L_1. Nous recalculons le nouveau nombre minimal de batch, nb_l et trouvons que $nb_l = 2$. Alors, le batch 1 est fermé.

Le nouvel ordre des tâches dans L_1 est le suivant : 2, 3, 4, 5, 6, 7, 8. On place la tâche 2 dans un nouveau batch, batch 2, et on recalcule le nouveau nombre minimal de batch avec les tâches qui restent dans L_1. nb_{new} est toujours 2. On met donc les tâches 3, 4 et 5 dans le batch 2 (étape 2.1), mais nb_{new} est toujours 2. La première tâche de L est maintenant la tâche 6. Sa taille est égale à 4, la somme de la taille des tâches dans le batch 2 est égale à 5. Donc, la tâche 6 ne peut pas être placée dans le batch 2. On passe donc à l'étape 2.1.1.

Dans l'étape 2.1.1, nous calculons l'espace nécessaire pour placer entièrement la tâche dans le batch 2 : capacité utilisée du batch = 5, taille de la tâche 6 = 4. Alors, on a besoin d'une unité d'espace pour placer la tâche 6 dans le batch 2, donc *space* = 1. Ensuite, on trie les tâches du batch 2 en ordre décroissant des tailles et on obtient l'ordre des tâches suivant : 3, 2, 4, 5.

Dans l'étape 2.1.1.1, nous enlevons la dernière tâche du batch 2, *i.e.* la tâche 5. La taille de la tâche 5 est égale à 1 ($v_{last} = 1$) et donc la nouvelle valeur de *space* est maintenant 0 ($space = space - v_{last}$). Nous plaçons la tâche 5 dans la liste *L*. Maintenant la tâche 6 peut être placée dans le batch 2. On obtient un batch 100% plein et donc le nouveau nombre minimal de batch diminue de 1 (étape 2.1.1.2).

Le nouvel ordre des tâches dans la liste *L* est le suivant : tâches 5, 7, 8 qui ont les tailles 1, 2 et 4, respectivement. Le nombre minimal de batch est égal à 1. Donc, on met toutes ces tâches dans un nouveau batch, batch 3.

Après avoir appliqué les étapes 3 et 4, nous obtenons la solution montrée dans la figure 4.13.

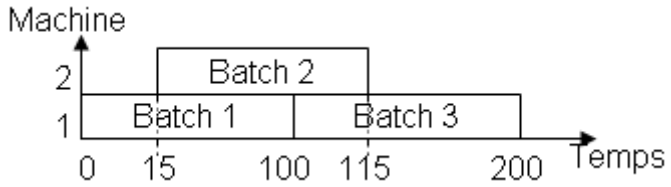

Figure 4.13. Résolution de l'exemple illustratif avec *SFD*

La complexité de l'algorithme est $0(N^3 logN)$. Les étapes 2 et 2.1 sont co-dépendantes et la complexité de SFD dépend largement de l'étape 2.1. L'étape 2.1 est exécutée au plus *N* fois. Dans chaque nouvelle itération dans 2.1, on sort un certaion nombre de tâche du batch actuel, ce qui est borné par *N*. De plus, après chque fermeture d'un batch, les tâches de *L* est triée par ordre croissant des dates de disponibilité, ce qui génére une complexité de $0(NlogN)$ dans le pire cas.Ainsi, la complexité de l'algorithme est-elle égale à $0(N^3 logN)$.

Théorème 7. L'algorithme *SFD* est optimal pour le problème :
P | p-batch, r_j, p_j=p, v_j(fortement divisible), B | minimiser le nombre de batch

Preuve. Le nombre de batch crées par *SFD* est égal au nombre de batch crées par l'algorithme *FF*. Puisque *FF* est optimal pour le cas où les tailles des tâches sont fortement divisibles [24], *SFD* minimise le nombre de batch.

Théorème 8. L'algorithme *SFD* est optimal pour le problème :
$P \mid p\text{-}batch, r_j, p_j=p, v_j(fortement\ divisible), B \mid C_{max}$.

Preuve. Quand l'algorithme essaie d'insérer une tâche, disons *k*, dans un batch, nous pouvons avoir 3 cas différents :
1- la tâche *k* entre dans le batch et le nombre de batch à former avec les tâches restantes diminue,
2- la tâche *k* entre dans le batch mais le nombre de batch à former avec les tâches restantes ne diminue pas,
3- la tâche *k* n'entre pas dans le batch.

Dans le cas 1, l'algorithme ferme le batch après avoir placé la tâche *k* dedans. Dans le cas 2, la tâche *k* sera mises dans le batch mais le batch va rester ouvert en attendant d'autres tâches. Dans le cas 3, l'algorithme va chercher les tâches à sortir du batch afin d'y insérer la tâche *k*. D'après le lemme 9, cette opération est possible. Même si un certain nombre de tâche est enlevé du batch, puisqu'on obtient un batch 100%, le nombre de batch à créer avec les tâches restantes diminue forcément. Donc, l'algorithme *SFD* ferme un batch si et seulement si, après avoir placé une tâche dans un batch, le nombre total de batch à former avec les tâches restantes diminue.

Supposons que, pour un problème avec *N* tâches et *M* machines, le nombre minimal de batch est égal à *nb*. Soit s_m le début d'exécution du batch *m*. Notons s_{nb} le début d'exécution du dernier batch et r_N la date de disponibilité de la dernière tâche. Si $r_N = s_{nb}$, alors la valeur de C_{max} est optimale. Si $r_N < s_{nb}$, alors, il n'y a pas de temps mort entre le batch b_{nb} et le batch qui précède b_{nb} sur la même machine. Puisque les tâches sont placées consécutivement sur les machines, le batch précédant le dernier batch sur la même machine est le batch $b_{(nb-M)}$. Considérons une chaîne de batch avec zéro temps mort entre les batch sur la machine qui contient le dernier batch : $b_{(nb-q*M)}\ldots\ b_{nb}$ où $q \geq 1$. Maintenant, considérons une tâche *k* dont la date disponibilité est égale au début d'exécution du batch $b_{(nb-q*M)}$, *i.e.* $r_k = b_{(nb-q*M)}$.

Avec l'algorithme *SFD*, un batch ne peut être fermé que lorsque le nombre de batch à former avec les tâches restantes a diminué de un par rapport au nombre de batch qu'il restait à former quand le batch précédent a été fermé. Ainsi, après avoir rajouté une dernière tâche à un batch, l'algorithme crée toujours un nombre

minimal de batch avec les tâches restantes. La condition donnée dans le lemme 6 est alors satisfaite. La solution est donc optimale et on a $C^*_{\max} = r_k + \lceil nb_k / M \rceil * p$. ■

4.5 Méthodes exactes et approchées pour le problème :

$P \mid p\text{-}batch, r_j, p_j=p, v_j, B \mid C_{max}$

Nous proposons dans cette section un modèle mathématique en nombres entiers et un algorithme de 2-approximation.

4.5.1 Modèle de programmation linéaire en nombres entiers (PLNE)

Nous avons vu dans le chapitre 1 que Chung et al. [23] ont proposé un modèle PLNE pour un problème très proche du notre. La différence est que Chung et al. [23] considèrent des durées d'exécution différentes pour les tâches. Leur modèle linéaire peut donc être utilisé pour notre problème, mais, il est possible de proposer un autre modèle PLNE plus efficace. Dans la section 4.6, nous comparons notre modèle à celui de Chung et al. [23]. Nous désignons notre modèle par *PLNE* et celui de Chung et al. [23] par $PLNE_{lit}$.

$PLNE_{Cmax}$

Indices :

j	1,…,N pour les tâches
k	1,…,N pour les batch (puisque au plus N batch peuvent être créés)
m	1,…,M pour les machines

Paramètres :

v_j	taille de la tâche j
r_j	date de disponibilité de la tâche j
N	nombre de tâches
M	nombre de machines
B	capacité de la machine
p	durée d'exécution
nb	borne inférieure pour le nombre de batch ($nb=\left\lceil \sum_{j=1}^{N} v_j / B \right\rceil$)

Variables de décision :

x_{jkm}	1 si la tâche j est exécutée dans le batch k et sur la machine m, 0 sinon
b_{km}	1 si le batch k est affecté à la machine m, 0 sinon
S_{km}	début d'exécution du batch k sur la machine m
C_{max}	makespan

Minimiser C_{max}

tq.

$$\sum_{k=1}^{N}\sum_{m=1}^{M} x_{jkm} = 1 \qquad \forall j \in [1, N] \qquad (1)$$

$$\sum_{k=1}^{N} v_j * x_{jkm} \le B * b_{km} \qquad \forall k \in [1, N], \forall m \in [1, M] \qquad (2)$$

$$\sum_{m=1}^{M} b_{km} \le 1 \qquad \forall k \in [1, N] \qquad (3)$$

$$S_{km} \ge x_{jkm} * r_j \qquad \forall j \in [1, N], \forall k \in [1, N], \forall m \in [1, M] \qquad (4)$$

$$S_{km} \ge S_{k-1,m} + p * b_{k-1,m} \qquad k = 2,..., N, \forall m \in [1, M] \qquad (5)$$

$$C_{max} \ge S_{Nm} + p * b_{Nm} \qquad \forall m \in [1, M] \qquad (6)$$

$$b_{k,(k \bmod M)+1} = 1 \qquad \forall k \in [1, nb] \qquad (7)$$

$$b_{k,m'} = 0 \qquad \forall k \in [nb + 1, N], \forall m \ne (k \bmod M) + 1 \qquad (8)$$

$$x_{jkm} \in \{0,1\}; b_{km} \in \{0,1\}; S_{km} \ge 0; C_{max} \ge 0$$

L'ensemble de contraintes (1) assure l'affectation de toutes les tâches à un batch exactement et à une machine exactement. L'ensemble de contraintes (2) représente la contrainte de capacité si le batch *k* est créé sur la machine *m*. L'ensemble de contraintes (3) affecte chaque batch à, au plus, une machine. L'ensemble de contraintes (4) impose à la date de début d'exécution d'un batch d'être supérieure ou égale aux dates de disponibilité des tâches de ce batch. Dans le cas où il y a plusieurs batch exécutés sur une même machine, l'ensemble de contraintes (5) assure une différence au moins égale à *p* entre les débuts d'exécution de ces batch. Si b_{km} est égal à 0, alors le batch *k* sur la machine *m* est un batch fictif, et donc sa date de début d'exécution est directement donnée au batch *k*+1 sur la machine *m*. Ainsi, la durée d'exécution du batch *k* devient implicitement égale à 0. Enfin, l'ensemble de contraintes (6) fixe la valeur du C_{max}. Les ensembles de contraintes (7) et (8) sont des inégalités valides permettant de diminuer l'espace de recherche du modèle mathématique. Grâce à ces contraintes, notre but est de diminuer le nombre de solutions équivalentes. Les différentes solutions considérées ici sont dues à l'affectation des batch aux machines. Par exemple, si les dates de début d'exécution de 2 batch sont égales, les machines sur lesquelles ces batch sont exécutés peuvent être échangées, ce qui correspond à 2 solutions équivalentes. Précisons maintenant l'objectif des ensembles de contraintes (7) et (8).

Notons que dans un ordonnancement optimal, comme les durées d'exécution sont égales, les batch peuvent être consécutivement affectés sur les machines. Donc, l'ensemble de contraintes (7) affecte les *nb* premiers batch

consécutivement sur les machines (ici, k mod M définit la machine sur laquelle le batch sera exécuté, (k mod M)+1 évite d'avoir 0 comme indice de machine, et, sans perte de généralité, nous plaçons le premier batch sur la machine d'indice 2). Une fois que tous ces batch sont affectés aux machines, nous continuons la procédure d'affectation des batch nb+1 à N. Par contre, nous ne pouvons pas savoir à l'avance si ces batch seront créés ou pas. Donc, la variable binaire $b_{k[(kmodM)+1]}$ (pour tout k > nb) peut valoir 0 ou 1. Mais, comme un batch peut être affecté à une seule machine, les variable binaires b_{km} (pour tout $k > nb$ et m différent de (k mod M) + 1) sont forcément égales à 0. Ainsi, nous introduisons l'ensemble de contraintes (8).

Nous pouvons illustrer cette procédure de pré affectation par un exemple. Supposons que pour un problème avec 10 tâches et 3 machines, la valeur de nb soit égale à 5. Alors, les batch d'indice 1 à 5 sont affectés machine par machine en commençant par la deuxième. Ensuite, le reste des batch, qui peuvent être ou non fictifs, est aussi pré affecté consécutivement aux machines. Ainsi, grâce à l'ensemble de contraintes (8), certains machines sont interdites aux batch 5 à 10. Par exemple, si le batch 6 est créé, il ne peut être affecté ni à la machine numéro 2 ni à la machine numéro 3 ($b_{62} = b_{63} = 0$ et $b_{61} \geq 0$).

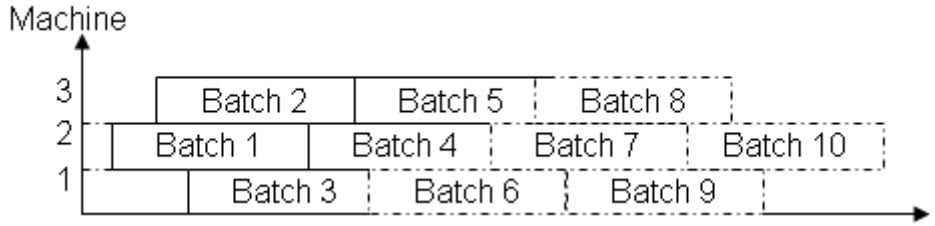

Figure 4.14. Procédure de pré affectation des batch avec les ensembles de contraintes (7) et (8)

Quand le nombre de machines M est égal à 1, l'ensemble de contraintes (8) est remplacé par :

$$b_{km} \geq \mathrm{b}_{\mathrm{k+1,m}} \qquad \forall k \in [nb+1, N-1] \qquad (8)$$

Grâce à cet ensemble de contraintes, nous imposons que les batch de petits indices soient prioritaires par rapport aux batch de grands indices. L'indice k varie de nb+1 à N-1 car les nb premier batch sont déjà créés, *i.e.* $b_k = 1$ pour $k = 1$, …, nb. Nous allons parler de l'efficacité des inégalités valides dans la partie 4.6.2.

Le nombre de variables du modèle est égal à $N^2M + 2NM + 1$. Le nombre de contraintes pour le cas où M > 1 est égal à $N^2M+3NM+N+nb(2-M)$. Si $M = 1$, le modèle contient $N^2 + 5N - 1$ contraintes.

4.5.2 Algorithme de 2-approximation

Dans la partie 4.4.1, nous avons proposé un algorithme exact (algorithme *split job*) qui est capable de trouver la solution optimale du problème lorsqu'il est permis de couper les tâches. Comme nous l'avons vu précédemment, cet algorithme est aussi un algorithme de borne inférieure du problème. Ainsi, dans le cas où il trouve une solution avec des tâches non-coupées, cette solution est la solution optimale du problème. Dans cette section, nous développons un algorithme de 2-approximation appelé « combine job » utilisant l'algorithme *split job* et cherchant à reformer les tâches originales.

Ce nouvel algorithme commence par exécuter *split job* afin de trouver une première solution. Ensuite, dans chaque batch, il cherche les tâches coupées. S'il en trouve, les tâches coupées sont enlevées des batch dans lesquels elles ont été mises et une liste contenant ces tâches en ordre décroissant des tailles est créée. L'étape suivante de l'algorithme est d'essayer de mettre ces tâches dans les batch déjà existants en respectant leurs dates de début d'exécution. C'est-à-dire que si une tâche peut être mise dans un batch en respectant la contrainte de capacité, sa date de disponibilité doit être inférieure ou égale au début d'exécution de ce batch. Enfin, s'il y a encore des tâches qui ne sont pas mises dans un batch, l'algorithme applique *FFD* sur ces tâches restantes et les batch ainsi formés sont ordonnancés après les batch formés par l'algorithme split job.

Algorithme « *Combine Job* »
1. Exécuter *split job*.
2. Enlever les tâches coupées des batch créés à l'étape 1 et les mettre dans une liste : L
3. Trier les batch par ordre croissant des dates au plus tôt
4. Trier les tâches de L selon l'ordre décroissant de leur taille
5. Pour tous les batch, b_b, créés dans l'étape 1 et pour toutes les tâches, j_L, de L
    5.1. Si la tâche j_L peut être mise dans le batch b_b et si sa date de disponibilité est inférieure ou égale à la date de début d'exécution de b_b, placer j_L dans b_b et l'effacer de L
6. Pour toutes les tâches qui n'ont pas encore été mises dans un batch, appliquer *FFD*. Trier les batch ainsi formés par ordre croissant des dates au plus tôt et les ordonnancer après les batch créés à l'étape 1.

La complexité de l'algorithme dépend de l'étape 5. Le nombre de batch trouvé par l'algorithme *split job* est borné par le nombre de tâche, N, dans le problème. De plus, si chaque batch contient une tâche coupée, on a N-1 tâches coupées. Alors, pour chaque tâche coupée, au plus N itérations sont faites dans l'étape 5. Ainsi, la complexité de l'algorithme est $O(N^2)$.

Illustrons sur un petit exemple la démarche suivie par *combine job*. Supposons que l'on a 10 tâches dans le problème, avec les tailles et dates de disponibilité précisées dans le tableau 4.3. Soit la capacité des machines égale à 6 et le nombre de machines fixé à 2. La durée d'exécution d'un batch est égale à 60 minutes. Nous allons détailler l'exécution de l'algorithme, étape par étape, sur des diagrammes de Gantt.

Tableau 4.3. Exemple illustratif pour l'algorithme *combine job*

tâche	1	2	3	4	5	6	7	8	9	10
date de disponibilité	0	0	10	20	30	40	50	60	70	80
taille	4	3	2	1	5	4	3	2	5	4

Etape 1 de combine job : Après avoir exécuté l'étape 1 de combine job, on obtient une solution initiale avec 6 batch et une durée de C_{max} égale à 190. Le diagramme de Gantt ci-dessous montre les batch ainsi que les tâches qu'ils contiennent. L'indice « c » précise quelles sont les tâches coupées.

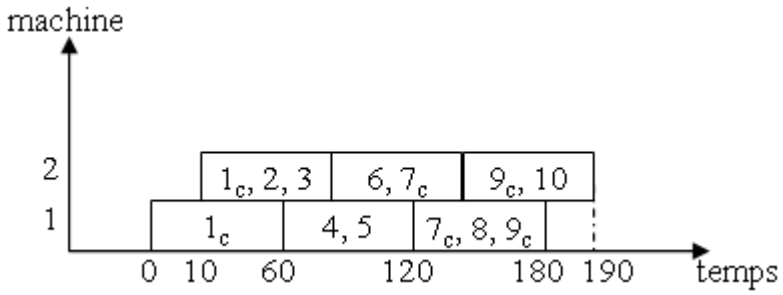

Figure 4.15. Etape 1 de l'algorithme *combine job*

Etape 2 de combine job : Les tâches coupées sont enlevées des batch, et donc, on obtient les batch avec des tâches non-coupées présentés sur la figure 4.16.

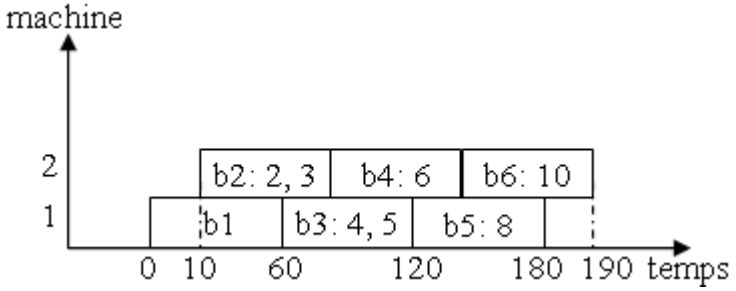

Figure 4.16. Etapes 2 et 3 de l'algorithme *combine job*

Etape 3 de combine job : Après avoir sorti les tâches coupées, on obtient une liste contenant les tâches 1, 7 et 9 dans l'ordre suivant : tâche 9, tâche 1 et tâche 7.

Etape 4 de combine job : Pour les tâches 9, 1 et 7, l'algorithme essaie maintenant de voir si les tâches peuvent être mises dans les batch créés précédemment en respectant la contrainte de capacité et les dates au plus tôt de ces batch.

De par sa taille, la tâche 9 peut être placée dans le batch 1 mais sa date de disponibilité est supérieure à la date de début d'exécution de ce batch. Le reste des batch n'ont pas assez d'espace pour accepter la tâche 9. Donc, la tâche 9 n'est mise dans aucun batch à cette étape.

La tâche 1, elle est mise dans le batch 1. Ensuite, la tâche 7 est placée dans le batch 5.

Etape 5 de combine job : FFD est appliqué sur la tâche 9, et un batch est créé pour cette tâche. Puis, ce batch est ordonnancé après les batch créés dans l'étape 1. Donc, ce nouveau batch (le 7ème batch) est exécuté juste après le batch 5.

La solution finale est présentée sur la figure 4.17.

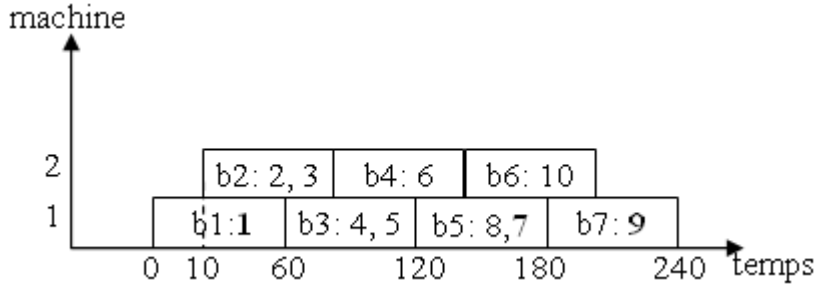

Figure 4.17. Solution obtenue par l'algorithme *combine job*

Théorème 9. *Combine job* est un algorithme de 2-approximation pour le problème $P \mid p\text{-}batch, r_j, p_j=p, v_j, B \mid C_{max}$.

Preuve. Soit $C_{\max}^{LB}$ le C_{max} trouvé par *split job* (*LB* pour « lower bound » en anglais), et $C_{\max}^{*}$ la solution optimale. Il est clair que $C_{\max}^{LB} \leq C_{\max}^{*}$. Posons $C_{\max}$ la solution trouvée par combine job. Nous voyons que jusqu'à la fin de l'étape 4, la valeur de $C_{\max}$ est la même que celle trouvée par split job. A partir de l'étape 5, *combine job* peut avoir un $C_{\max}$ plus grand.

Supposons que split job crée *nb* batch. Alors dans le pire cas, il y a au moins une tâche coupée entre chaque batch et donc le nombre de tâches coupées devient *nb*-1. Si, de plus, ces tâches sont de grande taille, elles ne peuvent ni être

mises dans des batch déjà existants, ni ensemble dans un nouveau batch. Ces nb-1 tâches donnent alors lieu chacune à la création d'un batch.

Supposons qu'il y ait M machines et que chaque machine m exécute nb_m batch, avec $1 \leq m \leq M$, dans la solution trouvée par *split job*. Alors, $\max_{m=1}^{M}\{nb_m\} * p \leq C_{\max}^{LB}$. Notons qu'après avoir ordonnancé les nb-1 batch nouvellement créés après les nb premiers batch, chaque machine peut avoir au plus $\max_{m=1}^{M}\{nb_m\}$ batch de plus dans la nouvelle solution. La relation entre $C_{\max}^{LB}$ et $C_{\max}$ devient :

$$C_{\max} \leq C_{\max}^{LB} + \max_{m=1}^{M}\{nb_m\} * p$$

En outre,

$$C_{\max}^{LB} + \max_{m=1}^{M}\{nb_m\} * p \leq 2C_{\max}^{LB} \leq 2C_{\max}^{*}$$

On obtient alors

$$C_{\max} \leq 2C_{\max}^{*} \blacksquare$$

4.6 Expérimentation numériques

4.6.1 Données utilisées

Nous avons testé l'efficacité de nos méthodes de résolution sur des instances créées en nous inspirant de données réelles, fournis.

Les données utilisées sont inspirées des données fournies par le service de stérilisation du Centre Hospitalier Privé Saint Martin de Caen (CHPSM), France. Plus précisément, le nombre de machines, la capacité des machines, et la durée d'exécution des tâches sont des données réelles : Il y a 4 laveurs dans ce service de stérilisation, nous avons donc 4 machines dans notre problème d'ordonnancement. Les laveurs sont identiques et la capacité d'un laveur est égale à 6 DIN où DIN est une mesure standard de capacité pour les laveurs dans les services de stérilisation. La durée de lavage d'un ensemble de DMR est de 60 minutes.

Les tailles et les dates de disponibilité des tâches ont été générées après plusieurs observations sur le terrain (CHPSM, CHU de Grenoble). Suite à ces observations, nous avons défini les tailles comme des multiples de 1/36ème de la

capacité du laveur. Il y a ainsi un maximum de 36 tailles différentes pour les ensembles. Nous avons donc choisi de les générer en utilisant une distribution uniforme U [1, 36]. Plus précisément, on estime la taille v_j de la tâche j est estimée par : v_j = U [1, 36]/Capacité laveur.

En ce qui concerne la date de disponibilité d'une tâche, elle correspond à la date d'arrivée d'un ensemble de DMR au service de stérilisation. En supposant qu'il n'y ait pas de rinçage manuel, un ensemble de DMR est disponible pour le lavage dès son arrivée dans le service, s'il a suffisamment trempé dans le liquide de pré désinfection. Nous considérons ici que lorsque les DMR arrivent au service de stérilisation, ils sont suffisamment trempés.

D'après les informations que nous avons pu récolter, il peut y avoir entre 0 et 40 minutes entre deux arrivées successives de deux ensembles de DMR dans le service. Ainsi, nous avons échantillonné les dates de disponibilité à l'aide d'une distribution uniforme telle que la différence entre les dates de disponibilité de 2 tâches consécutives est égale à X minutes avec X~U [0; 40] (*i.e.* X suit une distribution uniforme discrète dans l'intervalle [0; 40]). Ce type d'arrivée est appelé par la suite « arrivée aléatoire ».

D'autre part dans certains services de stérilisation, le ramassage des DMR est fait de manière régulière auprès les blocs opératoires. Dans ce cas, il y a un personnel de l'établissement qui est chargé d'aller récupérer les ensembles de DMR régulièrement, à heure fixes, dans la journée. Les DMR arrivent donc régulièrement dans le service de stérilisation. Nous considérons alors deux types d'inter-arrivées pour les tâches : 20 minutes et 40 minutes d'inter-arrivée. Nous supposons aussi que le nombre de tâches arrivant au service en même temps est échantillonné suivant une distribution uniforme qui est U[0;2] pour le cas d'inter-arrivée de 20 minutes, et U[1;3] pour les inter-arrivées de 40 minutes (plus la durée entre deux ramassages est important, plus la chance d'avoir des ensembles de DMR arrivant en même temps au service de stérilisation augmente). Nous avons donc défini 3 types de dates de disponibilité en fonction du type d'arrivée de DMR dans le service. Nous regroupons les instances créées en fonction du type de dates de disponibilité des tâches. 3 types d'instances sont ainsi considérées : 1er type pour l'arrivée aléatoire, 2ème type pour le ramassage régulier de 20 minutes, et enfin, 3ème type pour le ramassage régulier de 40 minutes. Le deuxième regroupement des instances est fait en fonction du nombre de machines et du nombre de tâches. Le nombre de machines varie de 1 à 4, tandis que le nombre de tâches varie de la façon suivante : 10, 15, 20, 25 pour les petites instances, 50 pour les grandes instances. Pour chaque combinaison du nombre de tâches et du nombre de machines, nous testons 90 instances, et donc, pour chaque type de ramassage, *i.e.* 1er type, 2ème type, 3ème type, 30 instances sont testées. Un ordinateur Intel Corel 2 Duo, 3 Ghz CPU avec une ram de 3.25 GB est utilisé

pour les tests. Les algorithmes sont codés en C++, le modèle PLNE est testé avec CPLEX 10.2. La limite de la durée de résolution est fixée à une heure.

Notons que la procédure de création des instances donnée ci-dessus sera utilisée pour la création d'autres instances dans les sections suivantes.

4.6.2 Performance du modèle PLNE et qualité de l'algorithme de la borne inférieure

A. Comparaison de notre PLNE avec le PLNE de Chung et al. [23]

Nous comparons notre modèle au modèle PLNE proposé par Chung et al. [23]. Rappelons que dans leur modèle, les tâches peuvent avoir des durées d'exécution différentes. Ils ont donc posé des contraintes permettant de calculer les durées d'exécution des batch. Puisque toutes les durées d'exécution sont égales dans notre problème, nous avons enlevé ces contraintes de leur modèle afin de l'adapter au mieux au cas des durées d'exécution égales. Rappelons que nous notons notre modèle *PLNE*, et celui de Chung *et al.* [23] $PLNE_{lit}$. Le tableau 4.4 montre, pour *PLNE* et $PLNE_{lit}$, la durée de résolution moyenne et le pourcentage des instances résolues à l'optimal pour les 90 instances testées pour chaque combinaison du nombre de machines et du nombre de tâches considérées.

Il est clair que notre modèle est plus rapide que celui de Chung et al. [23]. Notre PLNE permet de résoudre optimalement l'ensemble des instances comportant jusqu'à 25 tâches/2 machines et 20 tâches/4 machines, alors que celui de Chung *et al.* [23] ne permet de résoudre à l'optimal que l'ensemble des instances comportant 10 tâches.

Naturellement la durée de résolution augmente avec le nombre de machines, à la fois pour *PLNE* et $PLNE_{lit}$. Pour certaines instances par contre, nous avons observé que la durée de résolution était plus élevée pour les tests avec 3 machines que la durée de résolution des tests avec 4 machines. Cette observation est liée aux dates de disponibilité dans nos instances. Pour certaines instances, l'écart entre la date de disponibilité des tâches est assez élevé. Dans ce cas, avoir un grand nombre de machine, comme par exemple 4 machines au lieu de 3, rend plus facile l'affectation des tâches aux machines pour minimiser le C_{max}. Ainsi, la durée de résolution obtenue est plus courte.

Tableau 4.4. Limites de résolution pour *PLNE* et $PLNE_{lit}$

Nbr. tâche	*Nbr. machine*	*PLNE*		$PLNE_{lit}$	
		Durée de résol.	*% d'optimalité*	*Durée de résol.*	*% d'optimalité*
10	1	<1 sec.	100%	<232 sec.	100%
10	2	<1 sec.	100%	<341 sec.	100%
10	3	<1 sec.	100%	<473 sec.	100%
10	4	<1 sec.	100%	<178 sec.	100%
15	1	<1 sec.	100%	>3600 sec.	0%
15	2	<1 sec.	100%	>3600 sec.	0%
15	3	≈10 sec.	100%	>3600 sec.	≈40%
15	4	≈7 sec.	100%	≈711 sec.	≈81%
20	1	≈1 sec.	100%	>3600 sec.	0%
20	2	≈242 sec.	100%	>3600 sec.	0%
20	3	≈450 sec.	100%	>3600 sec.	0%
20	4	≈341 sec.	100%	≈1349 sec.	≈66%
25	1	≈4.5 sec.	100%	>3600 sec.	0%
25	2	≈568 sec.	100%	>3600 sec.	0%
25	3	≈1147 sec.	≈89%	>3600 sec.	0%
25	4	≈933 sec.	≈88%	>3600 sec.	0%

B. Evaluation de l'impact des ensembles de contraintes 7 et 8

Quand nous avons décrit notre PLNE, nous avons dit que les deux derniers ensembles de contraintes étaient des inégalités valides dont l'objectif était d'éviter les solutions équivalentes. Enlever ces contraintes ne change pas la valeur optimale mais augmente la durée de résolution. Sans ces contraintes, il n'y a aucune instance résolue en une heure pour les cas ayant un nombre de tâches supérieur à 15, sauf dans le cas d'une machine avec 15 tâches, qui est résolue en 300 secondes en moyenne. Avec les ensembles de contraintes (7) et (8), ces instances sont résolues en moins d'une seconde en moyenne (*cf.* tableau 4.4).

C. Evaluation de l'algorithme split job en tant que borne inférieure

Afin d'évaluer la qualité de l'algorithme *split job* pour la borne inférieure, nous comparons les résultats trouvés par *split job* aux résultats optimaux trouvés par le *PLNE*. Pour le faire, nous montrons la différence moyenne entre les deux solutions pour différentes combinaisons du nombre de machines/nombre de tâches. Le gap entre la solution optimale et celle de *split job* est calculé par $(C^*_{\max} - C^{LB}_{\max}) * 100 / C^{LB}_{\max}$ où $C^*_{\max}$ est la solution optimale et $C^{LB}_{\max}$ est la

solution de borne inférieure trouvée par *split job*. Nous expérimentons les instances jusqu'à 25 tâches/2 machines, car, au-delà de cette combinaison, les instances ne sont pas 100% résolues à l'optimal. Sur le tableau 4.5, nous montrons les gaps moyens.

Tableau 4.5. Comparaison entre split job et les résultats optimaux

Nbr. tâches	*Nbr. machines*	*Gap moyen*
10	1	≈ 13 %
10	2	≈ 4,5 %
10	3	≈ 4,5 %
10	4	≈ 0,6 %
15	1	≈ 19,5 %
15	2	≈ 9,13 %
15	3	≈ 3,17 %
15	4	≈ 0,45 %
20	1	≈ 14 %
20	2	≈ 6 %
20	3	≈ 3,5 %
20	4	≈ 0,6 %
25	1	≈ 14 %
25	2	≈ 6,3 %

Le comportement de *split job* comme une borne inférieure est indépendant du type d'instance. Ainsi, nous regroupons toutes les instances pour montrer sa qualité. Comme nous pouvons le constater dans le tableau 4.5, la performance de *split job* est assez bonne en tant qu'algorithme de borne inférieure, sauf pour les cas à une machine. Pour le cas d'une machine, couper les tâches permet d'affecter à la machine un nombre de batch beaucoup plus petit que dans le cas où elles ne sont pas coupées. Ainsi, la différence entre la valeur de la borne inférieure et la solution optimale est élevée. Nous allons utiliser l'algorithme *split job* en tant que borne inférieure afin de tester la qualité des autres algorithmes proposés sur des grandes instances, pour lesquelles le PLNE n'est pas efficace.

4.6.3 Performances des heuristiques

Dans cette thèse, notre objectif était de proposer sur le plan théorique des heuristiques à performance de garantie et sur le plan pratique des heuristiques simples qui puissent être aisément comprises par le personnel des services de stérilisation. Malgré tout, nous avons souhaité évaluer la performance empirique de nos heuristiques.

D'après les recherches bibliographiques que nous avons faites, il n'y a pas d'algorithme ayant une garantie de performance pour le problème $P \mid p\text{-}batch, r_j, p_j=p, v_j, B \mid C_{max}$, ni pour un cas plus général où les durées d'exécution ne sont pas égales, i.e. le problème $P \mid p\text{-}batch, r_j, p_j, v_j, B \mid C_{max}$. La meilleur heuristique pour ce dernier problème est celle proposée par de Damodaran et Velez-Gallego [27]. Le principe de cette heuristique est d'abord de déterminer un horizon de temps. Puis, un problème de sac-à-dos (*knapsack problem* en anglais) est résolu, sans prendre en compte la date de disponibilité des tâches, pour créer un batch. Cette opération est répétée jusqu'à ce que toutes les tâches sont affectées dans un batch. Une fois que tous les batch sont formés, ils sont affectés aux machines avec une heuristique de complexité polynomiale (Puisqu'on a des durée d'exécution identiques, les batch sont affectés consécutivement en ordre croissant des dates au plus tôt). Pour la résolution du sac-a-dos, Damodaran et Velez-Gallego [27] utilisent l'algorithme dynamique proposé par Martello et Toth [93]. La complexité de cet algorithme est $O(\min(B*N,2^N)$, qui fait que l'heuristique de Damodaran et Velez-Gallego [27] est de complexité pseudo-polynomiale. Nous souhaitons évaluer la performance de nos heuristiques vis-à-vis de l'heuristique de Damodaran et Velez-Gallego [27] sur quelques petites instances. Nous notons leur heuristique H_{lit}.

Pour chaque instance testée, nous calculons son gap par rapport à la solution optimale avec la formule $(C^A_{\max} - C^{LB}_{\max}) * 100 / C^{LB}_{\max}$ où $C^A_{\max}$ représente la valeur trouvée par des algorithmes (avec A = *Combine job, FFM, BFM, WFM, NFM* et H_{lit}, respectivement) et $C^*_{\max}$ la solution optimale. Le nombre d'instance testée dans chaque combinaison du nombre de machines et de type d'instance est égal à 30.

De ces résultats, il apparaît que H_{lit} est plus performante que les algorithmes proposés dans ce chapitre. Mais ce que nous remarquons grâce au tableau 4.6 est que la performance de nos algorithmes augmente lorsque le nombre de machines croit. De plus, les 3 types qu'on a définit pour les dates de disponibilité ont aussi une influence sur la performance des algorithmes. Nous voyons qu'en général les $2^{ème}$ et $3^{ème}$ types d'instances sont résolus plus efficacement. Pour ce qui concerne la performance de nos heuristiques, il n'y a pas une grande différence entre elles. Mais *NFM* et *combine job* sont légèrement meilleure que les autres. Puisque la performance de nos algorithmes augmente avec le nombre de machine, il vaut la peine de tester leur performance sur un cas réel où nous avons beaucoup plus de tâches.

Tableau 4.6. Performance des algorithmes proposés sur les instances avec 10 tâches

Nbr. machines	*Type d'inst.*	H_{lit}	*Combine Job*	*FFM*	*BFM*	*WFM*	*NFM*
1	1	0,15	14	10	7,1	15,8	15,8
1	2	0,07	9,4	6,1	9,5	14,7	2,4
1	3	0,7	11,9	8,8	8,7	16,4	5,5
2	1	1,6	10	11,5	10,7	15,1	11,9
2	2	0,3	5,5	11,8	11,8	12,4	8,2
2	3	0,7	6,6	11,4	7,4	13,2	4,6
3	1	2,3	5,6	8,5	8,9	11,3	6,8
3	2	0,1	0,9	1,1	1,1	1,6	0,5
3	3	0	0,9	3,4	2,5	4,7	0,9
4	1	0,9	2,4	10	7,1	15,8	15,8
4	2	0	0,2	6,1	9,5	14,7	2,4
4	3	0	0,2	8,7	8,7	16,4	5,5

En ce qui concerne le cas réel, les services de stérilisation que nous avons étudiés appartiennent à des grands hôpitaux. Nous avons vu qu'il y a entre 45 et 50 ensembles de DMR qui arrivent au service de stérilisation par jour. De plus un service de stérilisation capable de traiter 40 à 50 ensembles de DMR par jour possède souvent 3 ou 4 laveurs [36]. Ainsi, nous testons nos heuristiques sur des instances contenant 50 tâches en présence de 3 et 4 machines. Pour ces instances, le modèle PLNE n'est plus capable de trouver la solution optimale en moins d'une heure. Nous utilisons donc l'algorithme de borne inférieure *split job* comme référence de comparaison. Le gap présenté correspond à $(C^{A}_{\max} - C^{LB}_{\max}) * 100 / C^{LB}_{\max}$ où $C^{A}_{\max}$ représente la valeur trouvée par les algorithmes (avec A = *Combine job, FFM, BFM, WFM, NFM* et H_{lit}, respectivement) et $C^{LB}_{\max}$ représente la solution de borne inférieure obtenue par *split job*. Le nombre d'instance testée dans chaque combinaison du nombre de machines et de type d'instance est égal à 30.

Tableau 4.7. Qualité des algorithmes proposés en termes de gap d'optimalité par rapport à la borne inférieure sur les instances avec 50 tâches

Nbr. machine	*Type d'inst,*	H_{lit}	*Combine Job*	*FFM*	*BFM*	*WFM*	*NFM*
3	1	4,5	12,5	18	17	23	16
3	2	0,5	1	2	4,3	4,2	0,8
3	3	0,6	0,9	1,8	1,8	1,8	0,7
4	1	0,6	1,2	2,2	4	4,5	1,1
4	2	0	0	0	0,2	0,2	0
4	3	0	0,1	0,5	0,6	0,4	0,1

Quand le nombre de machine augmente, nos méthodes de résolution s'améliorent. Leur performance est meilleure surtout pour les instances de type 2 et 3. La raison de ce fait est que quand plusieurs tâches sont disponibles en même temps, il est possible de bénéficier plus de la capacité des machines.

Même si H_{lit} est meilleur que nos algorithmes pour les instances inspirées du cas réel, nos algorithmes sont polynomiaux et chacun a une garantie de performance. L'heuristique H_{lit} est un algorithme pseudo-polynomial à cause de l'algorithme dynamique qu'elle utilise pour la création des batch. Notons que l'algorithme dynamique qu'ils utilisent pour résolution du problème de sac-à-dos considère des tailles entières pour les tâches alors que nous considérons des tailles fractionnelles. Nous avons donc multiplié les tailles (et la capacité des batch) de nos instances avec un coefficient pour les convertir en nombres entiers. La résolution des instances avec 50 tâches dure quelques mili-secondes avec nos méthodes. H_{lit} résout ces instances en une minute en moyenne. Notons que pour la résolution avec H_{lit}, nous avons considéré la capacité d'un batch égale à 600 et la taille des tâches comme inférieure ou égale à 600 pour qu'elles soient conformes à la résolution du problème de sac-à-dos. Si nous avions choisi d'arrondir nos tailles de façon à ce que la capacité d'un batch soit 6000 et les tailles inférieures ou égale à 6000, la durée d'exécution avec H_{lit} serait beaucoup plus élevé qu'une minute par instance.

4.7. Conclusion

Dans ce chapitre, nous avons étudié le problème de minimisation du C_{max} qui correspond à la minimisation de la durée totale de lavage dans notre problème de stérilisation. Puisque l'étape de lavage est souvent un goulet d'étranglement pour les services de stérilisation, la minimisation du C_{max} est un objectif important pour empêcher le goulet dans cette étape. Pour notre problème de stérilisation, conformément à la caractéristique des laveurs et des ensembles de DMR, nous avons considéré des machines identiques à traitement par batch et des tâches

ayant des tailles et dates de disponibilité différentes. Notons que, les tâches ayant toutes la même durée d'exécution, notre problème est un cas particulier des problèmes étudiés par [21], [23], [26], [27] et [138] et reste quand même *NP-dur* au sens fort.

Nous nous sommes concentrés sur des méthodes de résolution rapides. Après avoir travaillé sur deux cas particuliers et proposé des algorithmes exacts, nous avons traité le problème de départ. Premièrement, nous avons proposé un modèle PLNE et nous l'avons comparé à un autre modèle donné par Chung et al. [23]. Nous avons deuxièmement travaillé sur des méthodes de résolution rapides, ayant une complexité polynomiale. Nous avons proposé 5 algorithmes polynomiaux et démontré leurs garanties de performance. A notre connaissance dans la littérature, il n'y a pas d'algorithme de complexité polynomiale pour notre problème, ni un algorithme d'approximation.

Chapitre 5 : Minimisation des encours

5.1. Introduction

L'étude de la minimisation des encours a pour but de minimiser l'attente des DMR dans le stock de lavage pour notre problème de stérilisation. Les longues attentes dans le stock de lavage peuvent avoir plusieurs conséquences parmi lesquelles on trouve l'accumulation d'un grand nombre de DMR attendant le lavage. Or, il peut être possible d'utiliser les laveurs plus efficacement pour accélérer le processus de stérilisation dans les étapes en aval. Une autre conséquence serait l'augmentation de la durée de pré désinfection des DMR. Nous avons expliqué dans la partie 1 qu'une bonne durée de pré désinfection est importante pour un bon lavage et pour la durée de vie des DMR. Ainsi, la minimisation des encours est un objectif important pour notre problème de lavage.

La fonction objectif correspondant à notre problème d'ordonnancement est la minimisation de la $\sum C_j$. En nous inspirant de la notation de Graham [51], nous proposons la notation suivante pour le problème étudié :
$P \mid p\text{-}batch, r_j, p_j=p, v_j, B \mid \sum C_j$. Dans cette notation, *P* correspond aux machines identiques, *p-batch* à l'ordonnancement à traitement par batch, r_j et v_j sont respectivement les dates de disponibilité et tailles/volumes des jobs, $p_j = p$ signifie que les durées d'exécution sont égales et *B* correspond à la capacité des machines. Finalement, $\sum C_j$ est l'objectif d'optimisation où C_j représente la fin d'exécution de la tâche *j*. A notre connaissance, ce problème n'a pas encore été étudié dans la littérature. Nous allons d'abord étudier quelques cas particuliers et puis proposer des heuristiques pour le problème général. Nous allons ensuite modifier légèrement la fonction objectif. Nous étudierons alors la minimisation de la durée moyenne d'attente des ensembles de DMR, que nous noterons $(1/N)*\sum f_j$ où f_j signifie la durée d'attente de la tâche *j* avant d'être exécutée et *N* est le nombre de tâches. Nous allons d'abord proposer un modèle *PLNE* pour ce problème et ensuite une heuristique de complexité pseudo-polynomiale.

Les études effectuées dans cette partie ont donné lieu aux publications [107] et [108].

5.2. Complexité du problème P | p-batch, r_j, p_j=p, v_j, B | $\sum C_j$

Un cas particulier de notre problème est celui où toutes les dates de disponibilité des tâches sont égales (*i.e.* $r_j = r_{j'}$ pour tous j et j', sachant que $j \neq j'$). Ce cas a été démontré comme étant *NP*-dur au sens fort, même en présence d'une seule machine ([133]). Donc, le problème $P \mid p\text{-}batch, r_j, p_j=p, v_j, B \mid \sum C_j$ est également *NP*-dur au sens fort.

5.3. Cas particuliers où toutes les dates de disponibilité sont égales

Dans cette section, nous étudions deux cas particuliers pour lesquels les dates de disponibilité sont égales. Le premier cas considère des tâches dont les tailles sont fortement divisibles. Pour ce problème, nous donnons un algorithme simple qui trouve la solution optimale. Le second cas est un peu plus général et considère des machines parallèles avec des tâches de tailles arbitraires. Pour ce dernier, nous proposons un algorithme avec une garantie de performance de 3.

5.3.1 Cas particulier où les tailles des tâches sont « fortement divisibles » et les dates de disponibilité sont égales

Dans la partie 4.4.2, nous avons proposé un algorithme exact pour le problème $P \mid p\text{-}batch, r_j, p_j=p, v_j(fortement\ divisible), B \mid C_{max}$. Dans cette section aussi, nous nous inspirons du cas particulier d'un problème de bin packing où les tailles des objets sont fortement divisibles. Rappelons que les algorithmes *FFD* et *FF* sont optimaux pour minimiser le nombre de batch si les tailles sont fortement divisibles et ce quel que soit l'ordre des tâches.

Le problème est noté $P \mid p\text{-}batch, p_j=p, v_j(fortement\ divisible), B \mid \sum C_j$. L'algorithme proposé tri d'abord les tâches en ordre croissant des tailles, puis applique *FF* sur ces tâches. Enfin, les batch créés sont ordonnancés consécutivement sur les machines.

Algorithme *SFD2* (*SFD* pour « séquence fortement divisible »)
1. Trier les tâches en ordre croissant des tailles : *L*
2. Appliquer *FF* sur *L*
3. Ordonnancer les batch créés à l'étape 2 consécutivement sur les machines.

Donnons d'abord quelques lemmes afin de montrer l'optimalité de *SFD2* pour le problème $P \mid p\text{-}batch, p_j=p, v_j(fortement\ divisible), B \mid \sum C_j$.

Lemme 10. Dans la solution optimale du problème, les batch doivent être affectés aux machines par ordre décroissant du nombre de tâches qu'ils contiennent.

Preuve. Trivial. ■

Lemme 11. Considérons un cas particulier d'un problème de sac à dos[2] où les objets ont des tailles différentes mais des valeurs/profits identiques. Alors, il suffit de trier les objets en ordre croissant des tailles, puis, de les mettre dans le sac-à-dos dans cet ordre afin de trouver la solution optimale.

Preuve. Trivial. ■

En effet, le lemme 11 est équivalent à maximier le nombre d'objet mis dans un sac-à-dos, ou un batch pour notre problème. Ce lemme sera utilisé pour montrer que l'algorithme *SFD2* place un nombre maximum d'objet dans chaque batch.

Théorème 10. L'algorithme *SFD2* est optimal pour le problème :
$P \mid p\text{-}batch, p_j=p, v_j(fortement\ divisible), B \mid \sum C_j$

Preuve. L'algorithme *FF* est optimal pour minimiser le nombre de batch quand la taille des tâches est conforme à une séquence fortement divisible [24]. Notons *nb* le nombre de batch créés. D'après le lemme 11, *SFD2* va maximiser le nombre de tâches mises dans le premier batch. Ainsi, après la fermeture du premier batch, il va aussi essayer de maximiser le nombre de tâches mises dans le deuxième batch avec les tâches restantes, puis la même procédure pour le troisième batch, etc. Ensuite, comme indiqué dans le lemme 10, les batch sont affectés aux machines dans l'ordre décroissant du nombre de tâches qu'ils contiennent. Si chaque batch est 100% plein, sauf peut être le dernier, alors chaque batch contient un nombre maximum de tâches. Ainsi, la solution donnée par *SFD2* est optimale.

Considérons le cas où les batch ne sont pas forcément 100% plein. Dans ce cas, il pourrait être possible de faire un échange de tâches entre certains batch pour augmenter la capacité utitlisée, ainsi pour augmenter le nombre de tâches qu'ils contiennent. Soit b_j le batch d'indice j tel que $1 \leq j \leq nb$. Soit g_j l'espace disponible dans le batch j. Notons $v_j^{\min}$ la taille minimale et $v_j^{\max}$ la taille

[2] Un problème du sac à dos (en anglais, *Knapsack Problem*) est un problème qui modélise une situation analogue au remplissage d'un sac à dos, ne pouvant supporter plus d'une certaine capacité, avec tout ou partie d'un ensemble donné d'objets ayant chacun une taille et une valeur. Les objets mis dans le sac à dos doivent maximiser la valeur totale, sans dépasser la capacité du sac à dos [93].

maximale dans le batch *j*. On sait par [24] que $\sum_{1\le i\le k} g_i < v_{k+1}^{\min}$, *i.e.* la somme de tous les espaces disponibles dans les batch 1, 2, …, *k* est inférieure à la taille de la plus petite tâche dans le batch *k* +1. Supposons que tous les espaces dans les batch 1, 2, …, *k* soient tellement bien arrangés que les *k*-1 premiers batch sont maintenant 100% plein et que le batch *k* ait un espace disponible qui est égal à $\sum_{1\le i\le k} g_i$. Cet espace ne suffira pas si nous essayons de placer la plus petite tâche du batch *k*+1 dans le batch *k*. Pour augmenter la capacité utilisée du batch *k*, il faut enlever au moins une tâche de ce batch. Si, au lieu d'essayer d'insérer la tâche de taille $v_{k+1}^{\min}$, nous désirons insérer une tâche de taille encore plus grande, le nombre de tâches à enlever du batch *k* sera encore plus grand. Cela veut dire que malgré toute modification, il est impossible d'augmenter le nombre de tâches mises dans chaque batch. Ainsi, chaque batch contient un nombre maximum de tâches et la solution donnée par *SFD2* est optimale.■

Le tri effectué à l'étape 1 impose que la complexité de l'algorithme est égale à O(*NlogN*).

5.3.2 Cas particulier où les dates de disponibilité sont égales en présence de machines parallèles

Zhang *et al.* [150] étudie le problème de minimisation de $\sum C_j$ avec des tâches de tailles différentes, sur une seule machine, et lorsque toutes les tâches sont disponibles en même temps. Les durées d'exécution sont unitaires, *i.e.* égales à 1. Ils donnent 3 algorithmes d'approximation avec une garantie de performance 4, 2, et 3/2, respectivement. Nous généralisons ce problème et proposons un algorithme avec une garantie de performance 3 en présence de plusieurs machines et de durées d'exécution identiques mais pas forcément unitaires. Notre algorithme est composé de 2 étapes. La première étape trouve une borne inférieure où il peut y avoir des tâches coupées entre deux batch. La deuxième étape enlève les tâches coupées et essaie de les mettre dans les batch existants sans les couper. Enfin, *FFD* est appliqué sur toutes les tâches qui ne sont pas encore mises dans un batch. Ces derniers batch sont ordonnancés sur des machines à la suite les batch précédents. Commençons par l'algorithme de borne inférieure (nous l'appelons *Split Job 2*). Il construit d'abord une liste *L* contenant les tâches en ordre croissant des tailles. Tant que *L* n'est pas vide, les batch sont remplis successivement en affectant les tâches selon l'ordre de *L*.

On ouvre un batch et on le remplit avec les éléments de *L* en commençant par son premier élément. Si une tâche n'entre pas complètement dans un batch, elle est coupée en deux, de façon à ce que sa première partie est placée dans le

batch pour avoir un batch 100% plein. La deuxième partie de la tâche reste dans la liste *L* et devient le premier élément d'un nouveau batch.

Comme nous cherchons une borne inférieure pour le calcul de $\sum C_j$, une tâche coupée comptera uniquement dans le premier batch auquel elle est affectée.

Algorithme *split job 2* (*SJ2*)
1. Trier les tâches en ordre croissant de tailles : *L*
2. Tant que *L* n'est pas vide
    2.1. Ouvrir un batch,
    2.2. Tant que le batch n'est pas complètement plein ou *L* n'est pas vide
        2.2.1. S'il y a assez d'espace dans le batch, mettre la première tâche de *L* dans le batch, l'effacer de *L*. Sinon,
        2.2.2. Couper la tâche en deux de façon à ce que sa première partie entre dans le batch. Fermer le batch. Mettre à jour la taille de la tâche coupée.
        Fin si
    Fin tant que
Fin tant que
3. S'il y une tâche coupée et mise dans deux batch successifs, $bathc_k$ et $bathc_{k+1}$, alors diminuer le nombre de tâches mises dans $bathc_{k+1}$ de 1.
4. Affecter les batch consécutivement sur les machines.

La complexité de l'algorithme dépend de l'étape de tri, et est donc égale à $O(NlogN)$.

Théorème 11. *SJ2* donne une borne inférieure pour $P \mid p\text{-}batch, p_j=p, v_j, B \mid \sum C_j$.

Preuve. Zhang et al. [150] ont démontré que *SJ2* donne une borne inférieure pour le problème $1 \mid p\text{-}batch, p_j=1, v_j, B \mid \sum C_j$. Evidemment, si nous considérons le cas où les durées d'exécution sont égales, mais pas forcément unitaires, *SJ2* est toujours un algorithme de borne inférieure en présence d'une seule machine. La preuve de Zhang et al. [150] porte sur le fait que l'algorithme crée un nombre minimal de batch et, de plus, chaque batch contient un nombre maximal de tâche. Alors, affecter les batch par ordre décroissant du nombre de tâches qu'ils contiennent sur des machines identiques donnera aussi une borne inférieur.■

Algorithme *Combine job 2* (*CJ2*)
1. Exécuter *SJ2* pour obtenir une borne inférieure.
2. Sortir les tâches coupées des batch précédemment créés. Leur attribuer leur taille initiale. Créer une liste L_1 contenant ces tâches en ordre croissant de tailles.
3. Pour chaque élément de L_1, et pour chaque batch créé par *SJ2* en ordre de leur affectation sur les machines,
    3.1. Si la tâche peut entrer dans le batch, la mettre dedans.
    Fin si
4. Appliquer *FFI* sur les tâches qui ne sont pas encore mises dans un batch.
5. Affecter les batch créés à l'étape 4 aux machines à la suite des batch créés par *SJ2*.

La complexité de l'algorithme dépend de l'étape 3. Le nombre de batch créés à l'étape 1 est borné par le nombre de tâches N. La taille des tâches est inférieure ou égale à la capacité des machines. Alors, la première tâche du premier batch n'est pas coupée. Il peut y avoir alors au plus N-1 tâches coupées, et donc, N-1 élément dans L_1. Ainsi, *CJ2* a une complexité $O(N^2)$.

Théorème 12. La garantie de performance de *CJ2* est 3.

Preuve. A la fin de l'étape 1, nous pouvons avoir 2 cas : 1- aucun batch ne contient une tâche coupée, 2- certains batch contiennent des tâches coupées. Considérons le cas 1. Si, à la fin de l'étape 1, aucun batch ne contient une tâche coupée, alors chaque batch, sauf peut être le dernier, sont 100% plein et ils contiennent un nombre maximal de tâches. De plus, un nombre minimal de batch est crée. La solution est donc optimale.

Considérons maintenant le deuxième cas. Notons *BI* la valeur de la $\sum C_j$ donnée par *SJ2* à la fin de l'étape 1. Soit *nb* le nombre de batch créés par *SJ2*. Puisque les batch sont affectés consécutivement sur les machines, l'exécution de tous les batch se termine à l'instant $\lceil nb/M \rceil$ avec M le nombre de machines dans le problème. Alors, les M premiers batch sont exécutés à l'instant 0, les M batch suivant sont exécutés à l'instant p, etc. Montrons une affectation possible des batch créés par *SJ2* sur une figure (*cf* figure 5.1).

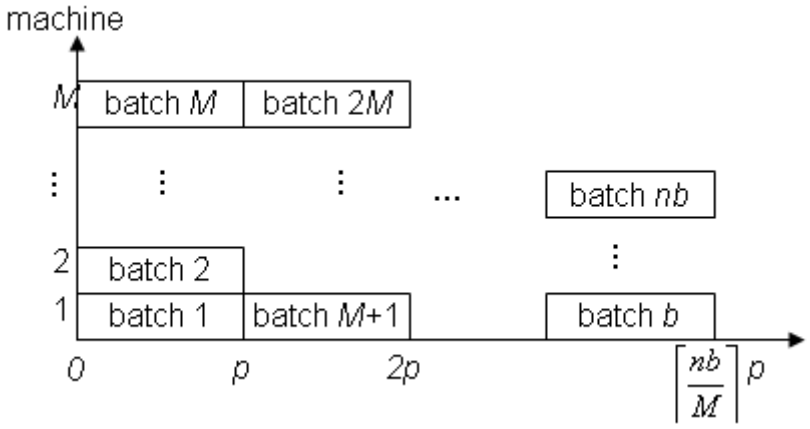

Figure 5. 1. Placement des batch avec *SJ2*

Notons c_k le nombre de tâches coupées dans les batch dont la date de fin d'exécution est k avec $k = p, 2p, \ldots, \lceil nb/M \rceil p$. Evidemment, $0 \leq c_k \leq M \ \forall k$. Quand les tâches coupées sont enlevées des batch à l'étape 2, nous allons avoir une diminution de $p^* \sum_{\forall k} k^* c_k$ dans la fonction objectif. Ainsi, la valeur de la $\sum C_j$ à la fin de l'étape 2 est égale à $BI - p^* \sum_{\forall k} k^* c_k$.

Après avoir affecté leur taille initiale, l'algorithme essaie de mettre les tâches coupées dans les batch créés par *SJ2*. S'il reste encore des tâches, *FFI* est appliqué et les batch ainsi créés sont ordonnancés suivant les batch créés par *SJ2*. Dans le pire des cas, chaque tâche initialement coupée est exécutée toute seule dans un batch. De plus, pour rendre encore plus mauvais la valeur de la $\sum C_j$, les c_k tâches seront exécutées entre les instants $k\text{-}p+\lceil nb/M \rceil p$ et $k+\lceil nb/M \rceil p$ avec $k = p, 2p, \ldots, \lceil nb/M \rceil p$, même si c_{k-1} est 0. Cela veut dire qu'une tâche initialement coupée sera exécutés $\lceil nb/M \rceil p$ unités de temps plus tard par rapport à sa date d'exécution déterminée par *SJ2* même s'il peut être possible de l'exécuter plus tôt avec *CJ2*. Notons $\sum C_j^{CJ2}$ la valeur de la $\sum C_j$ donnée par l'algorithme *CJ2*. Dans le pire des cas, nous obtenons l'inégalité suivante :

$$\sum C_j^{CJ2} \leq BI - p^* \sum_{\forall k} k^* c_k + c_1^*(\lceil nb/M \rceil p + p) + \ldots + c_l^*(\lceil nb/M \rceil p + \lceil nb/M \rceil p)$$

avec $l = \lceil nb/M \rceil$ et $k = 1, 2, \ldots, \lceil nb/M \rceil$.

$$\Leftrightarrow \sum C_j^{CJ2} \leq BI - p^* \sum_{\forall k} k^* c_k + p^* \sum_{\forall k} k^* c_k + \lceil nb/M \rceil p^* \sum_{\forall k} c_k$$

$$\Leftrightarrow \sum C_j^{CJ2} \leq BI + \lceil nb/M \rceil p^* \sum_{\forall k} c_k \text{ …….....(A)}$$

Considérons une solution où une seule tâche est exécutée dans chaque batch créés par *SJ2*. Evidemment, cette solution aurait été la solution optimale pour *CJ2*. Alors, dans le cas où il y a des tâches coupées, la valeur de la solution donnée par *SJ2* est forcément supérieure au cas où il n'y a qu'une tâche dans chaque batch. Alors,

$$BI > M^*p + M^*2p + \ldots + M' * \lceil nb/M \rceil p$$

avec M' le nombre de tâche/batch dont la date de fin d'exécution est égale à $\lceil nb/M \rceil p$.

$$\Leftrightarrow BI > M^*p(1+2+\ldots+ \lceil nb/M \rceil -1) + M' * \lceil nb/M \rceil p$$

$$\Leftrightarrow BI > M^*p(\lceil nb/M \rceil -1)\ (\lceil nb/M \rceil)/2 + M' * \lceil nb/M \rceil p$$

$$\Leftrightarrow 2BI > p^*\lceil nb/M \rceil^*[M^*(\lceil nb/M \rceil -1) + 2M']$$

Afin de maximiser la partie droite de l'inégalité (*A*), nous imposons $c_k = M$ pour tout k et $c_l = M'$ avec $l = \lceil nb/M \rceil p$. D'où, nous obtenons,

$$2BI > p^*\lceil nb/M \rceil^*[M^*(\lceil nb/M \rceil -1) + M'] = \lceil nb/M \rceil p^* \sum_{\forall k} c_k$$

Enfin,

$$\sum C_j^{CJ2} < BI + 2BI$$

$$\Leftrightarrow \sum C_j^{CJ2} < 3BI$$

■

5.4. Cas général : Heuristiques inspirées des algorithmes classiques de bin packing

Dans cette section, nous allons proposer des méthodes rapides pour la résolution du problème $P \mid p\text{-}batch,\ p_j=p,\ r_j,\ v_j,\ B \mid \sum C_j$. Ces heuristiques sont aussi basées sur les algorithmes classiques du bin packing. Après avoir trié les tâches en ordre croissant des dates de disponibilité, nous définissons des intervalles de temps dans lesquelles nous cherchons les tâches pour former des batch. La longueur de l'horizon du temps est comprise entre les dates de disponibilité de la première et de la dernière tâche. Ensuite, cet horizon est divisé

en intervalles de longueur *p*. Pour chaque intervalle, nous appliquons une version modifiée des heuristiques *FF*, *BF* et *WF*. La raison de former des batch avec des tâches appartenant à un même intervalle de temps est d'éviter les attentes longues. Par conséquent, nous essayons d'empêcher de mettre dans un même batch les tâches de date de disponibilité petites avec les tâches de date de disponibilité grandes. Nous supposons que les tâches sont préalablement triées en ordre croissant des dates de disponibilité.

5.4.1. Intervalle First Fit Modifié

Cette heuristique fonctionne d'abord en divisant l'horizon de temps en intervalles de longueur égale à *p*. Ensuite, pour chaque intervalle de temps, l'heuristique crée des listes avec des tâches dont les dates de disponibilité sont dans l'intervalle correspondant. Pour les tâches de chaque intervalle, *FF* est appliqué aux tâches qui sont triées en ordre croissant des dates de disponibilité.

Intervalle First Fit Modifié (*IFFM*)
1. Trier les jobs en ordre croissant des dates de disponibilité r_j
2. Soit le nombre d'intervalles, *nbr_intervalle*, égal à $\lceil (r_N - r_1)/p \rceil$
3. Mettre dans une liste, L_1, les tâches dont les dates de disponibilité sont égales à 0.
4. Pour *i* de 1 jusqu'à *nbr_intervalle*
    4.1. S'il y a au moins une tâche dans l'intervalle] (*i*-1)**p*, *i***p*],
        4.1.1. Mettre dans L_1 les tâches dont les dates de disponibilité sont dans] (*i*-1)**p*, *i***p*]
        4.1.2. Trier les tâches de L_1 en ordre croissant des dates de disponibilité
        4.1.3. Appliquer *FF* sur L_1 pour créer des batch. Effacer de L_1 les tâches qui sont mises dans un batch.
    Fin si
Fin pour
5. Fixer pour chaque batch la date au plus tôt égale à la plus grande date de disponibilité des tâches qu'il contient.
6. Ordonnancer les batch consécutivement en ordre croissant des dates au plus tôt sur les machines.

Pour calculer la compléxité de cette algorithme, notons *D* le nombre d'intervalles trouvé à l'étape 2. Alors, on entre dans la boucle « pour » à l'étape 4 au plus *D* fois. *N* étant le nombre de tâches total, soit *N/c* le nombre de tâche dans l'intervalle *d* avec *c* une constante et *d* = 1, 2,…, *D*. L'étape 4.1 est exécutée pour chaque différente valeur de *d*. La complexité de l'étape 4.1 dépend de l'étape 4.1.2 (ou également 4.1.3). Ainsi, pour *N*/*c* tâches, l'étape 4.1 a une complexité

qui est égale à (N/c)log(N/c). Enfin, la complexité finale d'*IFFM* peut être exprimée par O(*DNlogN*).

5.4.2. Intervalle Best Fit Modifié

Intervalle Best Fit Modifié (*IBFM*)

1. Trier les jobs en ordre croissant des dates de disponibilité r_j
2. Soit le nombre d'intervalles, *nbr_intervalle*, égal à $\lceil (r_N - r_1)/p \rceil$
3. Mettre dans une liste, L_1, les tâches dont les dates de disponibilité sont égales à 0.
4. Pour i de 1 jusqu'à *nbr_intervalle*
    - 4.1. S'il y a au moins une tâche dans l'intervalle] (i-1)*p, i*p],
        - 4.1.1. Mettre dans L_1 les tâches dont les dates de disponibilité sont dans] (i -1) * p, i * p]
        - 4.1.2. Trier les tâches de L_1 en ordre croissant des dates de disponibilité
        - 4.1.3 Créer une liste, L_2, contenant les tâches de L_1 en ordre décroissant des tailles
        - 4.1.4. Tant que L_1 n'est pas vide
            - 4.1.4.1. Mettre la première tâche, j_k, de L_1 dans un nouveau batch, B_k
            - 4.1.4.2. Effacer j_k des listes L_1 et L_2
            - 4.1.4.3. Pour toutes les tâches, j_l, de L_2
                - 4.1.4.3.1. Si j_l entre dans B_k
                    - $B_k = B_k \cup j_l$
                    - Effacer j_l de L_1 et L_2
                    - Fin si
            - Fin pour
        - Fin tant que
    - Fin si
    - 4.2. Poser $i = i + 1$
- Fin pour
5. Fixer pour chaque batch la date au plus tôt égale à la plus grande date de disponibilité des tâches qu'il contient.
6. Ordonnancer les batch consécutivement en ordre croissant des dates au plus tôt sur les machines.

Après avoir déterminé les intervalles de temps, cette heuristique crée 2 listes pour chaque intervalle. La première liste contient des tâches en ordre croissant des dates de disponibilité. Cette liste fournit toujours le premier élément des batch. Ensuite, la deuxième liste est parcourue pour remplir un batch ouvert.

La deuxième liste contient des tâches en ordre décroissant des tailles. Le but de trier la deuxième liste de cette façon est qu'on range les tâches dans les boîtes les mieux remplies qui puissent les contenir, donc *BF*. *D* étant le nombre d'intervalles, la boucle « pour » à l'étape 4 est exécutée *D* fois. La boucle de l'étape 4.1.4 est exécutée au plus *N* fois. La boulce de l'étape 4.1.4.3 est exécutée au plus *N*-1 fois. La complexité de l'algorithme est donc de $O(DN^2)$.

5.4.3. Intervalle Worst Fit Modifié

Intervalle Worst Fit Modifié (*IWFM*)

1. Trier les jobs en ordre croissant des dates de disponibilité r_j
2. Soit le nombre d'intervalles, *nbr_intervalle*, égal à $\lceil (r_N - r_1)/p \rceil$
3. Mettre dans une liste, L_1, les tâches dont les dates de disponibilité sont égales à 0.
4. Pour *i* de 1 jusqu'à *nbr_intervalle*
    - 4.1. S'il y a au moins une tâche dans l'intervalle] (*i*-1)**p*, *i***p*],
        - 4.1.1. Mettre dans L_1 les tâches dont les dates de disponibilité sont dans] (*i* -1) * *p*, *i* * *p*]
        - 4.1.2. Trier les tâches de L_1 en ordre croissant des dates de disponibilité
        - 4.1.3 Créer une liste, L_2, contenant les tâches de L_1 en ordre croissant des tailles
        - 4.1.4. Tant que L_1 n'est pas vide
            - 4.1.4.1. Mettre la première tâche, j_k, de L_1 dans un nouveau batch, B_k
            - 4.1.4.2. Effacer j_k des listes L_1 et L_2
            - 4.1.4.3. Pour toutes les tâches, j_l, de L_2
                - 4.1.4.3.1. Si j_l entre dans B_k
                - $B_k = B_k \cup j_l$
                - Effacer j_l de L_1 et L_2
                - Fin si
            - Fin pour
        - Fin tant que
    - Fin si
    - 4.2. Poser $i = i + 1$
- Fin pour
5. Fixer pour chaque batch la date au plus tôt égale à la plus grande date de disponibilité des tâches qu'il contient.
6. Ordonnancer les batch consécutivement en ordre croissant des dates au plus tôt sur les machines.

Cette heuristique est la même que *IBFM* sauf l'étape de tri dans l'étape 4.1.3. La complexité de l'heuristique est la même que celle de *IBFM*, soit O(DN^2).

5.4.4. Exemple illustratif

Pour mieux comprendre la démarche des heuristiques *IFFM*, *IBFM* et *IWFM*, nous donnons ici un exemple illustratif. La seule différence entre ces heuristiques est l'application des heuristiques classiques de bin packing (*FF*, *BF* et *WF*) dans l'étape 4.1. Puisque toutes ces heuristiques se ressemblent beaucoup, l'exemple ne sera expliqué que sur *IBFM*.

Exemple illustratif : Nous avons 6 tâches dont les dates de disponibilité et les tailles sont données dans le tableau 5.1. Deux machines de capacité 6 sont disponibles à l'instant 0. La durée d'exécution d'un batch est de 60 minutes.

Tableau 5.1. Données de l'exemple illustratif

tâche	1	2	3	4	5	6
rj	0	10	50	70	130	150
vj	1	3	4	2	4	1

Résolution de l'exemple illustratif

Etape1 : Les tâches sont déjà triées dans l'ordre croissant des dates de disponibilité.

Etape2 : *nbr_intervalle*=$\lceil (150-0)/60 \rceil = 3$.
(L'étape 4.1 va calculer ces 3 intervalles comme]0, 60],]60, 120] et]120, 180].)

Etape 3 : Pour que les tâches dont la date de disponibilité est 0 soient incluses dans l'intervalle 1, l'algorithme les met dans la liste L_1.

Etape 4 : On a 3 intervalles. Donc, cette étape va être exécutée 3 fois.

Etapes 4.1, 4.1.1 et 4.1.2 : L_1 contient les tâches 1, 2 et 3.

Etape 4.1.3 : Les tâches de L_1 sont copiées dans une 2ème liste, L_2, en ordre décroissant des tailles. Ainsi, L_2 contient les tâches 3, 2 et 1.

Etape 4.1.4 : Cette étape consiste à former des batch jusqu'à ce qu'il n'y ait plus d'éléments dans L_1.

Etapes 4.1.4.1 et 4.1.4.2. : Pour être sensible à la date de disponibilité des tâches, quand l'algorithme ouvre un nouveau batch, L_1 fournit le premier élément de ce batch. Dans notre exemple, le batch numéro 1 est créé, et le premier élément de L_1, la tâche 1, est mise dedans. Ensuite, cette tâche est effacée des listes L_1 et L_2.

Etapes 4.1.4.3 et 4.1.4.3.1 : L'étape 4.1.4.3 a pour but de remplir le batch actuellement ouvert avec la logique de *BF*. Pour mieux remplir le batch, la liste L_2 est parcourue en commençant par le premier élément, et on essaie de mettre le plus grand élément dans le batch. Dans notre exemple, la tâche 3 est mise dans le batch qui a maintenant une taille égale à 5. Il n'y a plus assez d'espace pour la tâche 2 dans ce batch (car sa capacité est 6), donc le batch numéro 1 est fermé. La tâche 3 est effacée des listes L_1 et L_2.

Mais, il y a encore un élément dans L_1 (tâche 2). Donc, un deuxième batch est créé. Pour l'intervalle 1, nous avons donc formé 2 batch.

Enfin, il n'y a plus de tâche dans l'intervalle 1, toutes ont été mises dans un batch. L'algorithme effectue les mêmes opérations pour les intervalles 2 et 3, respectivement. Une fois que toutes les tâches sont mises dans un batch, les étapes 5 et 6 déterminent la date au plus tôt des batch et les ordonnance sur les machines. Un diagramme de Gantt est donné dans la figure 5.2 pour montrer la solution de l'exemple illustratif avec *IBFM*.

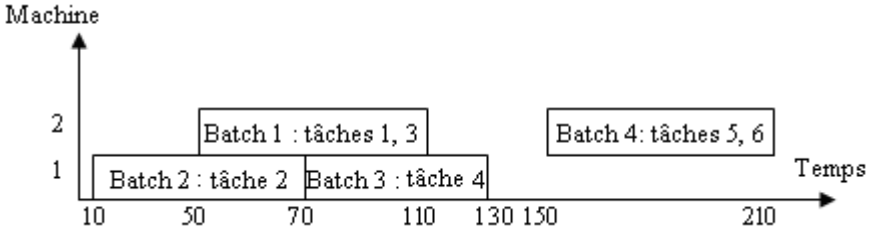

Figure 5.2. Résolution de l'exemple avec illustratif *IFBM*

Nous obtenons la valeur 840 pour la fonction objectif. Nous pouvons calculer l'attente des tâches avant leur exécution pour mieux évaluer les attentes dans le stock de lavage. Il n'y a que les tâches 1 et 6 qui attendent 50 minutes et 20 minutes, respectivement, dans le stock de lavage. Alors, nous avons une durée d'attente totale qui est égale à 70 minutes, ou encore une attente moyenne qui est égale à 11.6 minutes (70 minutes / N avec N le nombre de tâches).

5.5. De $\sum C_j$ vers la minimisation du temps moyen d'attente (*TMA*)

A partir de cette section, nous allons appeler différemment notre fonction objectif. Même si d'un point de vue optimisation, la minimisation des encours est identique à la minimisation du temps moyen d'attente, la mesure du temps entre la date de disponibilité et le début de lavage des ensembles de DMR a plus d'intérêt pratique pour notre problème de lavage. Donnons alors tout de suite ce que veut dire en terme mathématiques la minimisation du temps moyen d'attente. f_j étant la durée d'attente de la tâche j dans le stock de lavage, f_j est égale à C_j - r_j. Alors, $\sum f_j = \sum C_j - r_j$. Puisque $\sum r_j$ est nombre constant, $\sum C_j \equiv \sum f_j$. Notre but ici est de calculer l'attente moyenne, nous divisons ensuite $\sum f_j$ par le nombre de tâche, N.

Nous proposons la notation suivante pour notre problème :
$P \mid p\text{-}batch, r_j, p_j = p, w_j, B \mid \sum f_j/N$. Le dernier terme cette indique fois-ci la somme des « temps d'attente » divisée par le nombre de tâches.

La minimisation de la durée d'attente est importante d'un point de vue organisationnel pour les DMR. Puisque l'attente de DMR est élevée dans le stock de lavage, mesurer $(1/N)*\sum f_j$ nous donnera une information plus visuelle que $\sum C_j$.

Nous posons pour ce nouveau problème d'abord un modèle *PLNE* et une heuristique. Suivant ces méthodes de résolution, nous testons leurs efficacités.

5.5.1. Modèle de programmation linéaire en nombres entiers

Nous allons légèrement modifier le *PLNE* que l'on a proposé pour la minimisation du C_{max} afin d'avoir ce nouveau modèle mathématique pour la minimisation du temps moyen d'attente. Nous appelons ce modèle $PLNE_{TMA}$.

$PLNE_{TMA}$
Indices :
j 1,…,N pour les tâches
k 1,…,N pour les batch (puisque au plus N batch peuvent être créés)
m 1,…,M pour les machines
Paramètres :
v_j taille de la tâche j
r_j date de disponibilité de la tâche j
N nombre de tâches
M nombre de machines
B capacité de chaque machine
Q un très grand nombre nombre
p durée d'exécution

nb borne inférieure sur le nombre de batch ($nb=\left\lceil \sum_{j=1}^{N} v_j / B \right\rceil$)

Variables de décision :

x_{jkm} 1 si la tâche *j* est exécutée dans le batch *k* et sur la machine *m*, 0 sinon

b_{km} 1 si le batch *k* est affecté à la machine *m*, 0 sinon

S_{km} début d'exécution du batch *k* sur la machine *m*

s'_j début d'exécution de la tâche *j*

f_j temps d'attente pour le job *j*

Minimiser $1/N \sum_{j=1}^{N} f_j$

tq.

$$\sum_{k=1}^{N}\sum_{m=1}^{M} x_{jkm} = 1 \qquad \forall j \in [1, N] \qquad (1)$$

$$\sum_{k=1}^{N} v_j * x_{jkm} \le B * b_{km} \qquad \forall k \in [1, N], \forall m \in [1, M] \qquad (2)$$

$$\sum_{m=1}^{M} b_{km} \le 1 \qquad \forall k \in [1, N] \qquad (3)$$

$$S_{km} \ge x_{jkm} * r_j \qquad \forall j \in [1, N], \forall k \in [1, N], \forall m \in [1, M] \qquad (4)$$

$$S_{km} \ge S_{k-1,m} + p * b_{k-1,m} \qquad k = 2,..., N, \forall m \in [1, M] \qquad (5)$$

$$s'_j \ge S_{km} - Q(1 - x_{jkm}) \qquad \forall j \in [1, N], \forall k \in [1, N], \forall m \in [1, M] \qquad (6)$$

$$f_j \ge s'_j - r_j \qquad \forall j \in [1, N] \qquad (7)$$

$$b_{k,(k \bmod M)+1} = 1 \qquad \forall k \in [1, nb] \qquad (8)$$

$$b_{km} = 0 \qquad \forall k \in [nb+1, N], \forall m \ne (k \bmod M) + 1 \qquad (9)$$

$$x_{jkm} \in \{0,1\}; b_{km} \in \{0,1\}; S_{km} \ge 0; f_j \ge 0; s'_j \ge 0$$

Quand le nombre de machines *M* est égal à 1, l'ensemble de contraintes (9) est remplacé par :

$$b_{km} \ge b_{k+1,m} \qquad \forall k \in [nb+1, N-1] \qquad (9)$$

Nous n'avons pas besoin de réexpliquer toutes les contraintes de ce modèle, car il y a très peu de différences entre $PLNE_{TMA}$ et $PLNE_{Cmax}$. Les seules différences sont la fonction objectif et les contraintes (6) et (7). L'ensemble de contraintes (6) fixe le début d'exécution de la tâche *j* égal au début d'exécution du batch dans lequel elle est mise. Si la tâche *j* est placée dans le batch *k* qui est affecté à la machine *m*, son début d'exécution s'_j devient égal à S_{km}. Enfin, l'ensemble de contraintes (7) calcule la durée d'attente de la tâche *j*.

Le nombre de variables du modèle est $N^2M + 2NM + 2N$. Si le nombre de machines est plus grand que 1, le nombre de contraintes est $2N^2M + 3NM – M + 2N – nbM + 2nb$, et si $M = 1$, $2N^2 + 6N – 2$. Le modèle étant capable de résoudre exclusivement des instances de petite taille, nous allons proposer une heuristique dans la section suivante. Cette nouvelle heuristique va aussi fonctionner en formant des batch dans des intervalles de temps, mais contrairement aux heuristiques précédentes, elle va créer des intervalles de longueur différentes.

5.5.2. Heuristique d'horizon glissant : Intervalle First Fit (*IFF*)

Rappelons que les heuristiques proposées dans la section 5.4 créent des batch en déterminant des intervalles de temps. Seules les tâches dans un même intervalle peuvent être mises dans un même batch. De plus, la longueur des intervalles est fixe et égale à la durée d'exécution des tâches, *p*.

Cette fois-ci, nous proposons une autre heuristique qui fonctionne sur des horizons glissants, *i.e.* sur des intervalles de longueurs différentes. Nous définissons les intervalles de temps de manière dynamique. Nous utilisons un paramètre pour déterminer la longueur des intervalles de temps. Ce paramètre, appelé α, fixe la valeur de la longueur des intervalles, égale à 0, $p/4$, $p/2$ et p minutes à chaque nouvelle exécution de l'heuristique. Le choix de ces paramètres s'explique de la façon suivante : nous avons d'abord déterminé deux extrémités : 0 et p pour imposer une attente minimale et maximale, respectivement. De cette façon, nous avons deux stratégies différentes qui font que, soit une tâche n'attend pas du tout dans le stock de lavage, soit elle attend une durée assez longue, ce qui peut aider à faire des batch avec plus de tâches. Les paramètres $p/4$ et $p/2$ sont des valeurs intermédiaires au cas où les paramètres 0 et p ne fonctionnent pas bien. Une fois qu'un intervalle est créé, nous appliquons ensuite une procédure basée sur l'algorithme *First Fit*. Le but de cette procédure est de créer un seul batch avec les tâches dans l'intervalle de temps. Après avoir déterminé les tâches à mettre dans un intervalle, l'algorithme essaie de mettre les tâches dans l'ordre croissant de leur date de disponibilité dans le batch. Nous donnons maintenant notre heuristique appelée *IFF* après avoir détaillé la *Procédure First Fit* (*PFF*) qu'elle utilise pour les batch.

Procedure first fit (*PFF*)
1. Trier les tâches en ordre croissant des dates de disponibilité : *L*
2. En commençant par la première tâche de *L*, parcourir la liste *L* : mettre la tâche dans le batch s'il y a assez d'espace. Sinon, continuer avec la tâche suivante jusqu'à ce que *L* soit complètement parcouru. Fermer le batch.

Notons que *PFF* crée un seul batch avec les tâches d'un problème, alors que l'algorithme *FF* en crée plusieurs. Après avoir exécuté *PFF*, le batch est fermé et la date au plus tôt du batch est fixée à la plus grande date de disponibilité des tâches contenues dans ce batch. Une autre particularité d'*IFF* pour la création des intervalles de temps est que la disponibilité des machines est prise en compte. S'il n'y a pas de machine disponible, la longueur de l'intervalle de temps prend la valeur maximale entre le moment de disponibilité d'une machine et la longueur d'horizon d'attente pour déterminer la longueur de l'intervalle de temps. La raison pour laquelle nous nous sommes inspiré de l'heuristique *FF*, mais pas de *BF* ni *WF*, pour proposer *IFF* est que *FF* crée les batch en respectant la date de disponibilité des tâches plus que les autres heuristiques classiques de bin packing.

Intervalle First Fit (IFF)
1 Trier les tâches en ordre croissant des r_j : L
2 Tant que L n'est pas vide
    2.1 Mettre la première tâche, j_1, de L dans un batch et mettre à jour L
    2.2 Définir le moment de décision :
    t = max ($r_1 + \alpha * p$, prochaine libération parmi toutes les machines)
    2.3 Appliquer *PFF* pour ajouter les tâches j de L telles que $r_j \leq t$, dans le batch actuel
    2.4 Mettre à jour L,
    2.5 Fixer t' = max (date au plus tôt du batch, prochaine libération parmi toutes les machines)
    2.6 Dès que t' est atteint, commencer l'exécution avec le batch précédemment créé, calculer la nouvelle date de disponibilité de la machine
Fin tant que

Le paramètre α prend les valeurs 0, ¼, ½ et 1 pour avoir des intervalles de longueur 0, p/4, p/2 et p. L'heuristique est exécutée pour ces différentes valeurs et nous choisissons le meilleur résultat. Un batch est créé à chaque passage dans la boucle « tant que ». Lorsqu'un nouveau batch est ouvert, la première tâche de la liste L est mise dedans. La fin de l'intervalle de temps devient égale à t, comme calculé à l'étape 2.2. Notons que le début des intervalles de temps est toujours 0. Par contre, t n'est pas forcément la date de début d'exécution du batch. Imaginons le cas où il y a une tâche et une machine disponible à l'instant 0, α est ¼ et la plus grande date de disponibilité des tâches mises dans le premier batch égale à p/5. La fenêtre de temps correspond à l'intervalle [0, p/4]. Mais comme il y a une machine disponible et le batch est fermé avant p/4, nous pouvons l'exécuter à sa date au plus tôt, *i.e.* à l'instant p/5. L'étape 2.5 fait ce type d'ajustement afin d'éviter les attentes inutiles.

Donnons un exemple illustratif pour mieux comprendre le fonctionnement de l'heuristique.

Exemple illustratif : Soit 6 tâches dont les dates de disponibilité et les tailles sont données dans le tableau 5. 2. Soient deux machines de capacité 6 qui sont disponibles à l'instant zéro et la durée d'exécution d'un batch, *p*, 60 minutes.

Tableau 5. 2. Données de l'exemple illustratif

tâche	1	2	3	4	5	6
rj	0	10	30	40	60	70
vj	1	3	4	3	4	1

Pour différentes valeurs de α, *IFF* est exécuté 4 fois (α=0, ¼, ½ et 1) et le meilleur résultat est choisi. Pour comprendre l'heuristique, il suffit de faire une explication en considérant une seule valeur pour α. Soit α égale à ¼.

Etape 1 : Les tâches sont déjà triées en ordre croissant des dates de disponibilité et mises dans une liste *L*.

Etape 2.1 et 2.2 : En respectant l'ordre croissant des dates de disponibilité dans la liste *L*, son premier élément est mis dans un nouveau batch et la date de disponibilité de cet élément est utilisé pour déterminer un horizon de temps en prenant en compte la disponibilité des machines. Alors, la tâche 1 est mise dans le batch numéro 1 et *t* est calculé et vaut 15 minutes. Nous allons donc considérer les tâches dont la date de disponibilité est inférieure ou égale à 15 pour mettre dans le premier batch.

Etape 2.3 : Cette étape déclenche la procédure *PFF*. Ainsi, la liste *L* est parcourue depuis son premier élément jusqu'à ce qu'une tâche dont la date de disponibilité est plus grande que *t* soit trouvée, alors on arrête. Nous mettons donc la tâche 2 dans le batch numéro 1.

Etape 2.5 et 2.6 : Cette étape a pour but de fixer la date de début du le batch précédemment formé. t' est fixé à 10. La nouvelle disponibilité de la machine 1 est 70.
En suivant les mêmes instructions, nous obtenons les batch suivants :
- batch 2 qui contient la tâche 3,
- batch 3 qui contient les tâches 4 et 6,
- batch 4 qui contient la tâche 5,

Un diagramme de Gantt est donné dans la figure 5.3 pour montrer la solution de l'exemple illustratif avec *IFF*.

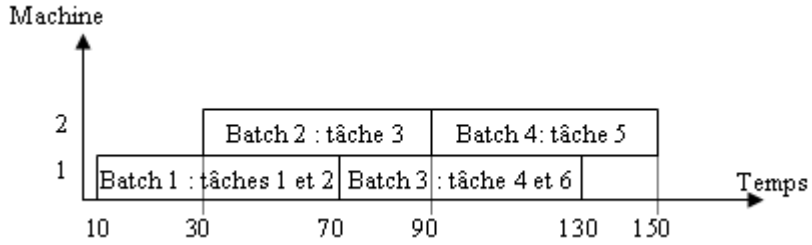

Figure 5. 3. Résolution de l'exemple avec illustratif *IFBM*

La complexité d'exécution est O($N^2 logN$) (La boucle tant que est exécutée dans le pire des cas N fois et *PFF* a une complexité O($NlogN$**)** à cause du tri à l'étape 1). L'algorithme est exécuté 4 fois au total pour les différentes valeurs de α. Donc, il est de complexité O($N^2 logN$). Par contre, si p fenêtres de temps sont considérées, *i.e.* un nombre égal à la durée d'exécution des tâches. Alors, l'algorithme est exécuté pour p différentes valeurs, et donc p fois. Dans ce cas, sa complexité n'est plus polynomiale et sa nouvelle complexité devient O($pN^2 logN$).

5.6. Expérimentations numériques

Plutôt que de considérer la $\sum C_j$, nous considérons $\sum f_j/N$, *i.e.* le temps moyen d'attente (ou *TMA*) pour les tests numériques. Nous consacrons cette partie à tester l'efficacité des heuristiques par rapport aux solutions optimales trouvées par le modèle linéaire ($PLNE_{TMA}$). Les instances que nous utilisons sont les instances testées dans la section 5.6. Rappelons que les tailles et les dates de disponibilité des tâches sont inspirées du cas réel ainsi que la capacité des machines. Nous avons encore 3 types d'instances en fonction des dates de disponibilité : 1er type pour l'arrivée irrégulière des ensembles de DMR au service de stérilisation, 2ème et 3ème types pour les arrivées régulières de 20 et 40 minutes.

5.6.1. Limites de résolution de $PLNE_{TMA}$

Le modèle PLNE a été codé en CPLEX 10.2. Nous avons fixé la durée maximale de résolution à une heure. Même si une instance n'est pas résolue à l'optimal, nous gardons la meilleure solution trouvée.

Nous avons observé que les types de date de disponibilité, *i.e.* les 1er, 2ème et 3ème types d'instances, ont une influence sur la durée de résolution du $PLNE_{TMA}$. Ainsi, nous divisons notre analyse en fonction du type d'instance. Le tableau 5.3 montre les durées moyennes de résolution pour chaque combinaison de nombre de tâches/nombre de machines pour lesquelles nous avons testé 90 instances (30 instances ont été testées pour chaque type d'instance). La colonne « % opt » montre le pourcentage des instances résolues à l'optimal en une heure.

La colonne « *dmr* » montre la durée moyenne de résolution, en secondes, pour les instances résolues à l'optimal.

Tableau 5.3. Durées de résolution et % d'optimalité avec $PLNE_{TMA}$

		1st instance type		2nd instance type		3rd instance type	
Nbr.de tâche	Nbr. de machine	*%opt*	*dmr*	*%opt*	*dmr*	*%opt*	*dmr*
10	1	100%	<1	100%	≈ 3.2	100%	<1
10	2	100%	<1	100%	<1	100%	<1
10	3	100%	<1	100%	<1	100%	<1
10	4	100%	<1	100%	≈1	100%	≈1
15	1	100%	595	100%	373	100%	906
15	2	100%	1100	100%	923	50%	2560
15	3	100%	1350	70%	1373	0%	>3600
15	4	100%	1699	0%	>3600	0%	>3600

Nous voyons que toutes les instances du 1er type sont résolues à l'optimal dans une limite de temps d'une heure. Par contre, pour les instances de type 2 et 3, le modèle commence à avoir du mal à trouver la solution optimale dans une courte durée. De plus, les instances testées sur 4 machines pour le 2ème type et sur 3/4 machines pour le 3ème ne sont jamais résolues jusqu'à l'optimalité. Quand on teste des instances contenant 16 tâches pour les instances de type 1, le modèle commence à avoir du mal à trouver la solution optimale en une heure en présence d'une ou plus de machines. Donc, la limite de résolution avec le modèle $PLNE_{TMA}$ peut être estimée à 15 tâches.

Nous allons maintenant discuter de l'impact des contraintes d'inégalité valides (8) et (9) du modèle mathématique sur la durée de résolution. Rappelons qu'enlever ces contraintes ne change pas la valeur de la solution optimale, mais, peut amener à un temps de résolution plus élevé. Nous avons choisi aléatoirement quelques instances de tous les 3 types et les avons testées avec $PLNE_{TMA}$ sans ces deux ensembles de contraintes. Toutes les instances de 10 tâches ont été résolues à l'optimal avec une durée d'exécution égale à 30 secondes pour les cas de 1 et 2 machines, plus de 100 secondes pour les cas de 3 et 4 machines. Aucun cas n'a été résolu d'une manière optimale pour les instances de 15 tâches. Malgré l'amélioration avec les contraintes d'inégalité valides, la limite de résolution n'est pas autant élevée que celle du modèle mathématique proposé pour la minimisation du C_{max}. La raison principale est l'utilisation d'un grand nombre, Q, dans l'ensemble de contraintes 6. La présence de grand Q dégrade la qualité de la relaxation continue qui devient très fractionnaire. Le modèle est donc utilisable pour les petites instances.

5.6.2. Qualité des heuristiques par rapport aux résultats de $PLNE_{TMA}$ sur les petites instances

Dans cette section, nous comparons les valeurs moyennes du « temps moyen d'attente (*TMA*) » fournies par les heuristiques et $PLNE_{TMA}$. Nous avons testé les mêmes instances que le chapitre 4 avec les heuristiques proposées. Notre but est d'évaluer la qualité des heuristiques d'un point de vue opérationnel. Nous avons généré des instances inspirées des données réelles. Nous cherchons à savoir combien attendraient en moyenne les ensembles de DMR dans le stock de lavage avant le lavage, si on utilisait l'une des heuristiques proposées dans cette partie.

Nous montrons les résultats en fonction du type d'instance. Les tableaux 5.4, 5.5 et 5.6 comparent les résultats de $PLNE_{TMA}$ aux solutions trouvées par les heuristiques. Nous montrons pour les 30 instances testées dans chaque combinaison de nombre de machines/nombre de tâches, la valeur moyenne des *TMA* trouvés par les diverses méthodes de résolution. Ce temps est exprimé en minutes. La colonne « % opt » des tableaux suivants représente le pourcentage des instances pour lesquelles les heuristiques trouvent le même résultat que $PLNE_{TMA}$.

Ces résultats montrent qu'*IFF* est beaucoup plus performant que les autres heuristiques. Les résultats donnés par les heuristiques *IBMF*, *IMFF* et *IMWF* sont loin de l'optimal. Nous comprenons donc qu'ils ne sont très adaptés à notre problème. D'un point de vue opérationnel, l'attente moyenne des tâches dans le stock de lavage n'est pas très éloignée de la durée moyenne d'attente trouvée par le modèle linéaire. Nous voyons aussi que le gap d'optimalité d'*IFF* est meilleur pour les $2^{ème}$ et $3^{ème}$ type d'instance que pour les instances de type 1. Dans ces instances, les tâches ont tendance à arriver en grande quantité, surtout dans les instances de type 3. Dans ce cas, *IFF* est capable de maximiser la capacité des batch. Outre cela, plus le nombre de machine est grand, plus les batch sont affectés aux machines sans attente excessive. Nous pouvons dire que le seul cas pour lequel *IFF* se comporte mauvais est le cas où nous avons une seule machine. Mais, même les résultats du modèle linéaire pour le cas d'une machine sont très élevés à cause de manque de ressource, ce qui veut dire qu'une seule machine n'est pas suffisante pour avoir des attentes considérablement petites dans le stock de lavage avec les données considérées ici.

Tableau 5.4. Moyenne de *TMA* par les différentes méthodes pour le 1er type d'instances

		$PLNE_{TMA}$	*IFF*		*IMBF*		*IMFF*		*IMWF*	
Nbr. tâche	Nbr. mach.	*TMA*	*TMA*	% opt	*TMA*	% opt	*TMA*	% opt	*TMA*	% opt
10	1	29	31,5	57	66,8	0	66,2	0	72	0
10	2	7,1	8,5	36	20,5	0	20,2	0	22,4	0
10	3	2,6	4	13	14,3	0	14,2	0	15,8	0
10	4	1,05	3,1	6	13,5	0	13,1	0	14,6	0
15	1	43	53,3	3	84	0	96	0	99	0
15	2	8,7	11,2	7	22	0	18	0	24	0
15	3	3,2	5,2	3	14	0	11	0	15	0
15	4	1,3	3,8	0	14	0	10,9	0	14,7	0

Tableau 5.5. Moyenne de *TMA* par les différentes méthodes pour le 2ème type d'instances

		$PLNE_{TMA}$	*IFF*		*IMBF*		*IMFF*		*IMWF*	
Nbr. tâche	Nbr. mach.	*TMA*	*TMA*	% opt	*TMA*	% opt	*TMA*	% opt	*TMA*	% opt
10	1	20	21	66	37,1	3	36,4	3	37,1	3
10	2	2,7	3,4	80	20,5	0	20,3	0	20	0
10	3	0,06	0,06	100	19,1	0	19,2	0	19,3	0
10	4	0	0	100	19,3	0	19,2	0	19,3	0
15	1	23,1	27,5	13	57	0	55	0	57	0
15	2	3,7	5	17	15	0	13	0	14	0
15	3	0,6	0.8	96	13,4	6	12,1	6	13,1	6
15	4	0.5	0.7	98	13,4	6	12,1	7	13,1	6

Tableau 5.6. Moyenne de *TMA* par les différentes méthodes pour le 3ème type d'instances

		$PLNE_{TMA}$	*IFF*		*IMBF*		*IMFF*		*IMWF*	
Nbr. tâche	Nbr. mach.	*TMA*	*TMA*	% opt	*TMA*	% opt	*TMA*	% opt	*TMA*	% opt
10	1	15	21	36	90	0	90	0	92	0
10	2	0,8	1,2	86	22,5	0	22,3	0	23,1	0
10	3	0	0	100	12,6	0	12,1	0	12,7	0
10	4	0	0	100	10,3	0	9,9	0	10,3	0
15	1	18,4	26,4	13	38	0	36	0	40	0
15	2	1	1,6	70	10	0	9	0	10	0
15	3	0.6	0.7	97	8,7	13	7,9	13	9,1	13
15	4	0.5	0.6	98	8,7	12	7,9	13	9	13

5.6.3. Performance d'*IFF* sur des instances de taille réelle

La section précédente nous a permis de voir qu'*IFF* est assez performante pour minimiser la durée d'attente des tâches sur des petites instances. Nous souhaitons maintenant voir si la même performance est obtenue pour des instances de taille réelle. Comme expliqué dans la partie 4.6.4, les hôpitaux que nous avons étudiés ont en moyenne 50 ensembles de DMR à traiter par jour. Donc, nous testons les instances définies dans la section 4.6.4 pour évaluer le comportement d'*IFF*. Nous le comparons aussi à une stratégie naturelle appliquée dans les services de stérilisation, la stratégie *FIFO online* (*cf.* section 2.3).

Nous considérons un service de stérilisation capable de traiter 50 ensembles de DMR par jour et qui dispose de 3 ou 4 laveurs. Nous testons donc des instances contenant 50 tâches en présence de 3 et 4 machines. Nous distinguons les tests en fonction du type d'arrivée des ensembles de DMR au service de stérilisation, *i.e.* en fonction des instances de type 1, 2 et 3, respectivement.

Le tableau 5.7 montre les valeurs minimales, maximales et moyennes, en minutes, de *TMA* pour les 30 instances testées pour chaque combinaison de nombre de machine/type d'instance.

Tableau 5.7. Comparaison entre *IFF* et *FIFO online* sur 3 et 4 machines

		IFF			*FIFO online*		
Nbr. de machine	Type d'inst.	min	max	moyenne	min	max	Moyenne
3	1	3,2	8,3	4,9	25	39,7	33,7
3	2	0	7,2	1,4	25,2	39,6	32,9
3	3	6,4	9,6	8,3	30,4	44,4	36,7
4	1	2	4,3	3,3	25	39	33,7
4	2	0	2,4	0,2	24,4	39,6	32,2
4	3	1,6	2,8	2	24,4	38,4	30,9

Bien entendu, une borne inférieure naturelle pour la durée d'attente est 0 minute pour les tâches. D'un point de vue pratique, nous observons que, dans la plupart des cas, *IFF* est capable de diminuer l'attente des tâches dans le stock de lavage. L'attente moyenne donnée par *IFF* est très faible.

Nous comprenons aussi par le tableau 5.7 que *FIFO online* provoque une attente énorme dans le stock de lavage. Cette observation nous montre combien

nous pouvons gagner en connaissant à l'avance l'arrivée des ensembles de DMR. Une telle connaissance va permettre aux responsables de stérilisation de mieux gérer les ressources de lavage tout en diminuant l'attente de DMR dans le stock de lavage.

5.7. Conclusion

Nous avons étudié dans ce chapitre la minimisation des encours à l'étape de lavage. L'étape de lavage est toujours modélisée comme un problème d'ordonnancement par batch où les tâches représentent les ensembles de DMR. Nous avons commencé par étudier quelques cas particuliers, afin de proposer un algorithme exact et un algorithme d'approximation. L'algorithme exact considère des dates de disponibilité identiques et des tailles « fortement divisibles ». Les algorithmes d'approximation, par contre, considèrent des tailles arbitraires. Dans la littérature d'ordonnancement par batch, Zhang et al. [150] proposent trois algorithmes d'approximation pour le problème $1 \mid p\text{-}batch,\ p_j = 1,\ v_j,\ B \mid \sum C_j$. Leurs algorithmes ont les garanties de performance asymptotiques 4, 2 et 3/2. Nous avons considéré un cas plus général où le nombre de machines est supérieur à 1. Pour le problème $P \mid p\text{-}batch,\ p_j = p,\ v_j,\ B \mid \sum C_j$, nous avons proposé un algorithme avec une garantie de performance de 3.

Après avoir étudié les cas particuliers, nous avons proposé 3 heuristiques qui sont des adaptations des heuristiques classiques de bin packing et nous avons étudié la minimisation de la $\sum C_j$ qui est en fait équivalente à la minimisation du temps moyen d'attente, qu'on note $1/N*\sum f_j$. Ce dernier objectif a pour but d'exprimer l'attente des DMR plus, ce qui est plus intéressant en pratique que la $\sum C_j$. Nous avons ensuite proposé un modèle de type *PLNE* et proposé une autre heuristique *IFF*.

D'après les expérimentations numériques, *IFF* est assez performante pour la minimisation de la durée d'attente des ensembles de DMR dans le stock de lavage. Nous avons également comparé *IFF* à la stratégie classiquement appliquée dans les services de stérilisation, *i.e. FIFO online*. *FIFO online* conduit à de grandes durées d'attente alors qu'*IFF* conduit à des temps d'attente assez petits.

La minimisation du temps d'attente à l'étape de lavage nous a encouragé à étudier un autre objectif. Rappelons-nous que dans certains services de stérilisation, un rinçage manuel est appliqué avant le lavage. Cette étape n'étant pas obligatoire, il a pour but de terminer la pré désinfection des DMR et les faire attendre avant le lavage sans risque de corrosion dû au liquide pré désinfectant. Dans le chapitre suivant, nous étudierons l'optimisation de la durée de trempage.

Chapitre 6 : Minimisation du dépassement moyen de la durée idéale de pré désinfection

6.1. Introduction

Dans les services de stérilisation que nous avons étudiés, le rinçage manuel est souvent appliqué alors que les laveurs effectuent aussi un rinçage automatique avant le lavage. L'une des raisons pour ce double rinçage est qu'il permet aux DMR d'attendre le lavage sans risque de corrosion due au liquide de pré-désinfection (ou trempage en d'autres termes). Comme l'étape de lavage est généralement le goulet d'étranglement du processus de stérilisation, l'attente des DMR dans le stock de lavage peut être assez longue (par exemple, 30 minutes ou plus). Par conséquent, le rinçage manuel enlève le liquide pré désinfectant et les DMR attendent le lavage après avoir été pré désinfectés. Notons que de longues durées de pré désinfection risquent de diminuer la durée de vie des DMR. Les responsables des services de stérilisation que nous avons étudiés fixent la durée idéale de pré désinfection à 20 minutes alors que 15 minutes de pré désinfection sont obligatoires et que 50 minutes correspondent à la limite supérieure. En ce qui concerne l'effet de corrosion du liquide désinfectant, il n'y a pas à notre connaissance de travail spécifique étudiant ce fait. Les responsables de services de stérilisation considèrent que les DMR sont soumis à une corrosion faible depuis le début de pré désinfection. La durée minimale de pré désinfection étant incontournable, l'effet de corrosion du liquide désinfectant augmente avec le temps.

Si la durée d'attente des DMR dans le stock de lavage est assez petite, la durée idéale de pré désinfection peut être respectée en supprimant le rinçage manuel et en conservant uniquement le rinçage automatique (*i.e.* rinçage dans les laveurs). L'opérateur chargé du rinçage peut alors être transféré à un autre poste de travail (par exemple à l'étape de conditionnement qui est toujours manuelle). En considérant que la durée idéale de trempage est 20 minutes, nous définissons le « dépassement moyen de la durée idéale de pré désinfection (*DMP*) » pour un ensemble de DMR comme la différence entre sa durée de trempage et la durée idéale de trempage.

Notre but principal est ainsi de minimiser le dépassement moyen de la durée idéale de pré désinfection. L'un des buts de cet objectif est d'examiner

l'opportunité de supprimer le rinçage manuel. Notons que nous considérons 20 minutes comme étant la durée idéale de pré désinfection mais cette valeur peut être changée dans nos méthodes de résolution si besoin. Outre la minimisation du *DMP*, nous introduisons un critère secondaire qui est la minimisation du nombre de cycles de lavage, qui est un critère économique et environemental. Ce deuxième critère n'est pas aussi important que le premier. Nous donnons d'abord la complexité et la notation du problème. Ensuite, pour la résolution, nous proposons d'abord des méthodes exactes et puis des heuristiques. Nous terminerons ce chapitre par des expérimentations numériques.

Les travaux effectués dans cette partie ont donné lieu à la publication [109].

6.2. Complexité et notation

La minimisation du dépassement moyen de la durée idéale de pré désinfection est une extension de la minimisation du temps moyen d'attente qui est la fonction objectif étudié dans le chapitre précédent. Un ensemble de DMR a un début de pré désinfection et a aussi une date d'entrée dans le service de stérilisation. Sachant qu'il doit être pré désinfecté au moins 15 minutes, le maximum entre son début de pré désinfection plus 15 minutes et son arrivée au service de stérilisation détermine sa date de disponibilité pour le lavage. Alors pour notre problème d'ordonnancement, nous considérons les paramètres t_j qui correspond au début de pré désinfection de la tâche j, et r'_j qui représente sa date d'entrée dans le service de stérilisation, *i.e.* le stock de lavage. Alors, la date de disponibilité r_j, d'une tâche est donnée par : $max(t_j+15 ; r'_j)$. Un cas particulier du problème s'obtient lorsque t_j+15 est égal à r'_j pour toutes les tâches. Alors, la minimisation du *DMP* se réduit à la minimisation du temps moyen d'attente (problème étudié dans le chapitre précédent) qui est *NP-dur* au sens fort. Donc, le nouveau problème est également *NP-dur* au sens fort.

Nous notons par f'_j la valeur du dépassement de la durée idéale de pré désinfection pour la tâche j. Ainsi, la fonction objectif que nous minimisons est $\sum f'_j/N$. Pour faire un parallèle avec les notations utilisées pour les problèmes précédents, nous pouvons utiliser la notation suivante :
$P \mid p\text{-}batch, r_j, p_j = p, v_j, B \mid \sum f'_j/N$.

6.3. Méthodes de résolution

Dans cette section, nous proposons quatre méthodes de résolution pour l'optimisation du *DMP*. La première méthode est un modèle PLNE inspiré du

modèle $PLNE_{TMA}$ que nous avons proposé dans la partie 5. Ensuite, nous proposons un modèle de *ε-contrainte* ayant un objectif secondaire qui est la minimisation du nombre de batch. Troisièmement, nous proposons une méthode qui fonctionne d'une façon opposée à celle de *ε-contrainte*. Enfin, nous proposons une heuristique de complexité polynomiale.

6.3.1 Modèle PLNE pour la minimisation du *DMP* : $PLNE_{DMP}$

Nous commençons par un modèle *PLNE* qui minimise le *DMP*. Nous appelons le nouveau modèle $PLNE_{DMP}$. Ce modèle est inspiré du modèle linéaire que nous avons présenté dans la section 5.5.1., *i.e.* $PLNE_{TMA}$.

Rappelons-nous que la date de disponibilité d'une tâche dépend de deux paramètres :

- la durée minimale de pré désinfection,
- la date d'entrée dans le stock de lavage.

La durée minimale de pré désinfection est fixée à 15 minutes. La pré désinfection d'un ensemble de DMR, j, commence à l'instant t_j, puis, cet ensemble entre dans le stock d'attente devant les laveurs à l'instant r'_j. Bien entendu t_j est forcément plus petit que r'_j. Mais, « t_j+15 » peut être plus grand ou plus petit que r'_j. C'est-à-dire qu'un ensemble de DMR peut être mis dans un laveur dès son arrivée au service de stérilisation s'il a été pré désinfecté au moins 15 minutes. Deux autres paramètres sont donc rajoutés au $PLNE_{TMA}$: t_j et r'_j qui correspondent au début de pré désinfection et à la date d'entrée dans le stock de lavage de la tâche j, respectivement. Deux autres ensembles de contraintes sont donc rajoutés au modèle : « $r_j \geq r'_j$ » et « $r_j \geq t_j + 15$ ».

Grâce à la contrainte « $r_j \geq t_j + 15$ », les 15 minutes minimales de pré désinfection sont toujours respectées. Afin de déterminer l'écart entre le temps de trempage réel et le temps de trempage idéal, qui est de 20 minutes, l'ensemble de contrainte 7 du $PLNE_{TMA}$ devient « $f'_j \geq s'_j - t_j - 20$ pour tout $j \in [1, N]$» avec t_j le début de pré désinfection, s'_j le début d'exécution et enfin f'_j la pénalisation pour la tâche j. Notons que si le temps de trempage se situe entre 15 et 20 minutes, f'_j est égal à 0. Le modèle linéaire est présenté ci-dessous.

$PLNE_{DMP}$

Indices :

j	1,…,N pour les tâches
k	1,…,N pour les batch (puisque au plus N batch peuvent être créés)
m	1,…,M pour les machines

Paramètres :

v_j taille de la tâche j
t_j début de pré désinfection de la tâche j
r'_j date d'entrée dans le stock de lavage pour la tâche j
r_j date de disponibilité de la tâche j
N nombre de tâches
M nombre de machines
B capacité de chaque machine
Q un très grand nombre nombre
p durée d'exécution
nb borne inférieure sur le nombre de batch ($nb = \left\lceil \sum_{j=1}^{N} v_j / B \right\rceil$)

Variables de décision :

x_{jkm} 1 si la tâche j est exécutée dans le batch k et sur la machine m, 0 sinon
b_{km} 1 si le batch k est affecté à la machine m, 0 sinon
S_{km} début d'exécution du batch k sur la machine m
s'_j début d'exécution de la tâche j
f'_j dépassement de la durée idéale de pré désinfection pour le job j

$$\textit{Minimiser} \sum_{j=1}^{N} f'_j$$

tq.

$$\sum_{k=1}^{N}\sum_{m=1}^{M} x_{jkm} = 1 \qquad \forall j \in [1, N] \qquad (1)$$

$$\sum_{k=1}^{N} v_j * x_{jkm} \le B * b_{km} \qquad \forall k \in [1, N], \forall m \in [1, M] \qquad (2)$$

$$\sum_{m=1}^{M} b_{km} \le 1 \qquad \forall k \in [1, N] \qquad (3)$$

$$S_{km} \ge x_{jkm} * r_j \qquad \forall j \in [1, N], \forall k \in [1, N], \forall m \in [1, M] \qquad (4)$$

$$r_j \ge t_j + 15 \qquad \forall j \in [1, N] \qquad (5)$$

$$r_j \ge r'_j \qquad \forall j \in [1, N] \qquad (6)$$

$$S_{km} \ge S_{k-1,m} + p * b_{k-1,m} \qquad k = 2,..., N, \forall m \in [1, M] \qquad (7)$$

$$s'_j \ge S_{km} - Q(1 - x_{jkm}) \qquad \forall j \in [1, N], \forall k \in [1, N], \forall m \in [1, M] \qquad (8)$$

$$f'_j \ge s'_j - r_j - 20 \qquad \forall j \in [1, N] \qquad (9)$$

$$b_{k,(k \bmod M)+1} = 1 \qquad \forall k \in [1, nb] \qquad (10)$$

$$b_{k,m'=0} \qquad \forall k \in [nb + 1, N], \forall m \ne (k \bmod M) + 1 \qquad (11)$$

$$x_{jkm} \in \{0,1\}; b_{km} \in \{0,1\}; S_{km} \ge 0; f'_j \ge 0; s'_j \ge 0$$

6.3.2 Modèle de *ε-contrainte* pour la minimisation d'un critère secondaire

T'kindt et Billaut [132] étudient plusieurs méthodes pour l'ordonnancement multicritère, dont *ε-contrainte*. Elle consiste à minimiser un critère sachant que les autres critères sont bornés supérieurement. Dans cette section, nous proposons une telle méthode minimisant un deuxième critère : nous souhaitons minimiser le nombre de batch créés. L'équivalent de ce critère dans notre problème de stérilisation est la minimisation du nombre de cycles de lavage lancés. Ce modèle, tout en respectant la valeur optimale de *DMP*, essaie donc de diminuer le nombre de batch utilisés dans le problème. Nous commençons donc par résoudre $PLNE_{DMP}$ afin de trouver la valeur optimale de *DMP*. Puis, en respectant la valeur optimale de *DMP*, nous diminuons le nombre de batch utilisés d'une unité et résolvons de nouveau $PLNE_{DMP}$. Nous continuons cette opération jusqu'à ce que la valeur optimale de *DMP* ne soit plus respectée, ou, que la borne inférieure pour le nombre de batch, *nb*, soit atteinte.

Notons que la borne inférieure triviale pour le nombre de batch, *nb*, définie comme un paramètre dans les modèles *PLNE*, n'est pas toujours atteignable. Pour illustrer ce fait, considérons 5 tâches dont les tailles sont égales à « $B/2 + \alpha$ » avec B la capacité d'un batch et α un nombre très petit. Nous pouvons facilement voir que le nombre minimal de batch pour ces 5 tâches est 5 (*i.e.* une tâche par batch). Mais, la formule de *nb* utilisée dans les modèles *PLNE* conduit à 3 batch ($nb = \left\lceil \sum_{j=1}^{5} (B/2+\alpha)/B \right\rceil$), qui n'est pas une solution atteignable. Par conséquent, nous utilisons un autre modèle *PLNE* pour calculer une borne inférieure atteignable pour le nombre de batch. Ce modèle que nous appelons $PLNE_{batch}$ correspond au modèle de Kantorovich dans la littérature de bin-packing ([136]). Commençons par présenter $PLNE_{batch}$.

$PLNE_{batch}$

Indices :

j	1,…,N pour les tâches
k	1,…,N pour les batch

Paramètres :

v_j	taille de la tâche j
N	nombre de tâches
B	capacité de la machine

Variables de décision :

x_{jk}	1 si la tâche j est exécutée dans le batch k, 0 sinon

b_k 1 si le batch k est créé, 0 sinon

$$Minimiser \sum_{k=1}^{N} b_k$$

tq.

$$\sum_{k=1}^{N} x_{jk} = 1 \qquad \forall j \in [1, N] \qquad (1)$$

$$\sum_{j=1}^{N} v_j * x_{jk} \leq B * b_k \qquad \forall k \in [1, N] \qquad (2)$$

$$x_{jk} \in \{0,1\}; b_k \in \{0,1\}$$

La fonction objectif a pour but de minimiser le nombre de batch utilisés. L'ensemble de contraintes (1) assure l'affectation de toutes les tâches à des batch. L'ensemble de contraintes (2) traduit les contraintes de capacité des batch.

Nous pouvons maintenant expliquer la méthode *ε-contrainte*. Dans chaque itération, la valeur optimale de *DMP* est respectée. Après la première résolution du $PLNE_{DMP}$, il peut être possible de diminuer le nombre de batch utilisés dans l'ordonnancement. Les étapes de l'algorithme sont les suivantes :

ε_{DMP}
1- Résoudre $PLNE_{DMP}$ pour trouver la solution optimale pour *DMP*.
2- Poser *NB* égal au nombre de batch créés dans l'étape 1.
3- Calculer la borne inférieure pour le nombre de batch, nb_{opt}, avec $PLNE_{batch}$
4- Si $NB = nb_{opt}$, arrêter l'algorithme. Sinon, diminuer *NB* d'une unité et le définir comme paramètre dans $PLNE_{DMP}$ pour la valeur maximale des indices de batch (*i.e.* l'indice *k* du $PLNE_{DMP}$ est maintenant borné par *NB*). Résoudre à nouveau $PLNE_{DMP}$
5- Si la fonction objectif primaire (*DMP*) augmente, alors arrêter l'algorithme. Sinon, aller à l'étape 4.

Un schéma montrant la procédure de résolution est proposé en figure 6.1 Dans la section sur les expérimentations numériques, nous discuterons de l'efficacité de cette méthode de résolution.

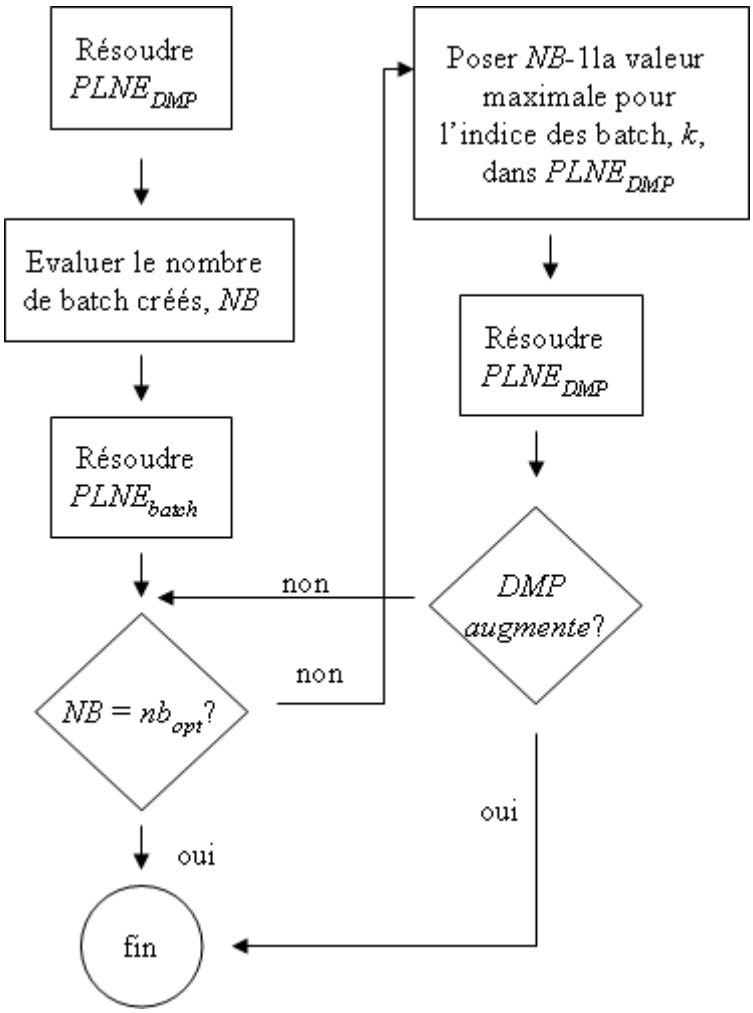

Figure 6.1. Procédure de résolution pour la méthode de *ε-contrainte*

Une fois que l'algorithme s'arrête, si le nombre de batch trouvé n'est pas le minimum possible, chercher une autre solution possèdant moins de batch va provoquer une détérioration selon le critère de *DMP*. Ainsi, la solution obtenue par ε_{DMP} est pareto-optimale.

6.3.3. Algorithme composé : *CompA*

Nous présentons, dans cette partie, une méthode basée sur le schéma de résolution présenté par Chung et al. [23]. Il est clair que le nombre de batch utilisés dans un ordonnancement appartient à l'intervalle tel que $1 \leq NB \leq N$ avec N le nombre de tâches dans le problème. Par contre, trouver la solution optimale pour *DMP* avec le modèle *PLNE* en explorant tous les nombres de batch atteignables peut augmenter considérablement la durée d'exécution. Dans cette

partie donc, nous proposons un algorithme qui fonctionne à l'inverse du modèle *ε-contrainte*.

Nous appelons cet algorithme *CompA*. D'abord, nous résolvons le $PLNE_{batch}$ pour trouver une borne inférieure pour le nombre de batch (nb_{opt}) et nous l'utilisons comme paramètre dans $PLNE_{DMP}$, *i.e.* l'indice des batch, *k*, varie de 1 jusqu'à nb_{opt}. Puis, nous permettons de dégrader le deuxième critère d'optimisation afin d'améliorer la valeur du *DMP*. Pour le faire, nous augmentons la borne supérieure du nombre de batch d'une unité, et résolvons à nouveau $PLNE_{DMP}$. Ces opérations continuent jusqu'à ce qu'il n'y ait plus d'amélioration pour le premier critère d'optimisation.

Notre but avec cette méthode est d'avoir une durée de résolution plus courte comparée à celle du modèle *ε-contrainte*. Fixer la borne supérieure des batch à nb_{opt} plus tôt qu'à *N* permet d'avoir un espace de solution plus petit. L'algorithme est donné ci-dessous.

CompA
1. Fixer la solution initiale pour *DMP* égale à un très grand nombre
2. Résoudre $PLNE_{batch}$ pour obtenir une borne inférieure sur le nombre de batch, notée nb_{opt}
3. Fixer $N'=nb_{opt}$
4. Définir *N'* comme la borne supérieure des indices de batch dans $PLNE_{DMP}$ (*i.e.* l'indice *k* dans $PLNE_{DMP}$ varie de 1 à *N'*)
5. Résoudre $PLNE_{DMP}$
6. Si la fonction objectif, *DMP*, baisse, fixer $N'= N' + 1$, aller à l'étape 4. Sinon, arrêter l'algorithme et garder la solution d'avant avec $N'= N' – 1$ batch.

Un schéma montrant la procédure de résolution est proposé sur la figure 6.2.

$PLNE_{batch}$ assure un nombre de batch est atteignable. En dehors de la restriction sur le nombre de batch, il n'y a pas d'autre restriction sur un autre indice ou sur une contrainte. Par conséquent, a priori, $PLNE_{DMP}$ ne fournit pas de solution non-réalisable. Par contre, pour un problème de grand taille, par ex. une instance contenant 100 tâches, $PLNE_{batch}$ peut ne pas être pas en mesure de trouver une solution dans le délai impartie (1 heure pour nos expérimentations). Donc, l'algorithme est arrêté avant d'avoir trouvé une solution pour *DMP*.

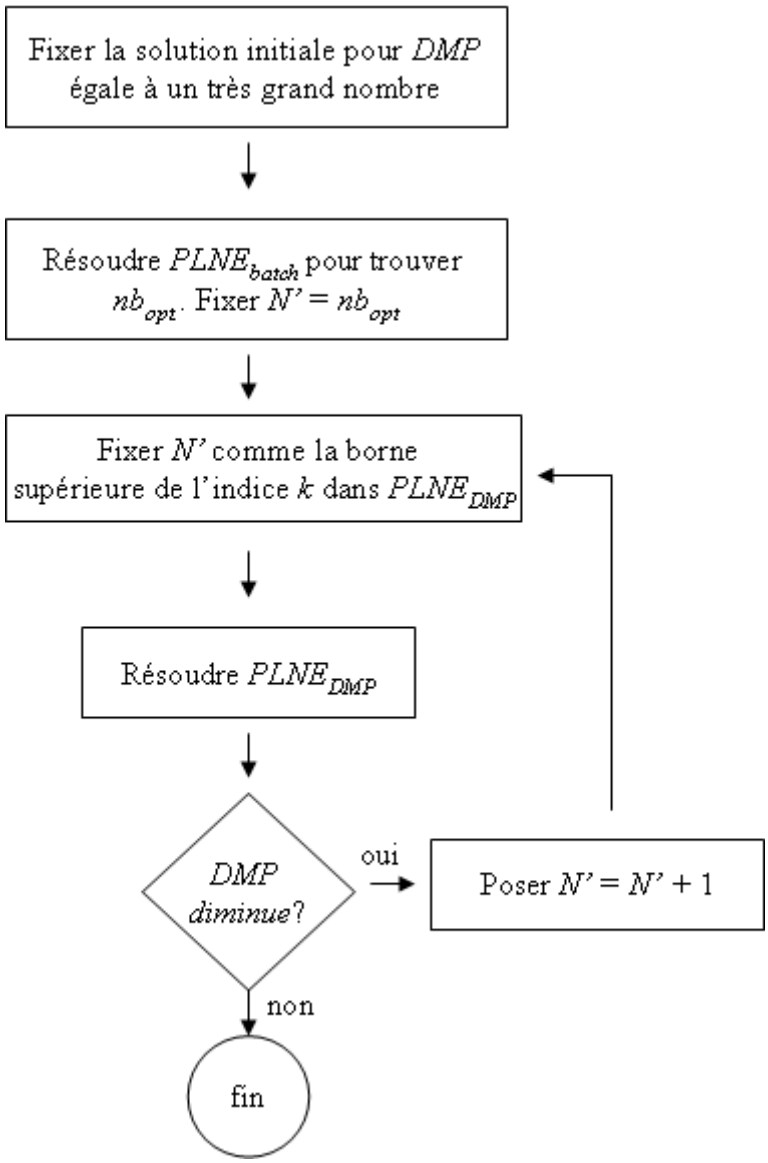

Figure 6.2. Procédure de résolution pour *CompA*

En ce qui concerne la qualité de solution, *CompA* ne trouve pas forcément la meilleure valeur pour *DMP*. Pour la résolution d'une instance quelconque, l'algorithme peut s'arrêter dans un optimum local pour *DMP*. Considérons 2 cas avec $N' = k$ et $N' = k+1$ pour lesquels $PLNE_{DMP}$ donne la même solution. Alors, $PLNE_{DMP}$ s'arrête car il n'y a plus d'amélioration pour le critère *DMP*, mais une meilleure valeur de *DMP* pourrait être trouvée avec un nombre de batch supérieure à $k+1$. Cependant, il vaut mieux noter qu'il n'y a aucune solution qui domine la solution donnée par *CompA*. Une solution dominante est une solution dont la valeur des deux critères est meilleure que celles données par *CompA*. Puisque *CompA* admet une dégradation pour le nombre de batch créé, même si la

valeur de *DMP* est améliorée, il est impossible de trouver une solution où le nombre de batch et le *DMP* sont inférieurs à ceux données par *CompA*.

Concernant la durée de résolution avec *CompA*, elle doit être a priori plus rapide que ε_{DMP}. Nous allons tester son efficacité à la fois pour la durée de résolution et pour la valeur de *DMP* dans la partie des tests numériques.

6.3.4. Heuristique d'intervalle de temps : *TIH*

Pour les problèmes de petite taille, les modèles *PLNE*, *ε-contrainte* et *CompA* peuvent fournir des solutions avec une durée de calcul raisonnablement courte. Mais, pour des grandes instances, ces méthodes risquent d'être moins efficaces et de nécessiter une longue durée de calcul. Par conséquent, nous proposons ici une heuristique de complexité polynomiale.

L'heuristique proposée commence par définir une fenêtre de temps [0, *t*], puis résout un problème de sac-à-dos, avec les tâches dont les dates d'arrivée appartiennent à la fenêtre de temps et en considérant qu'elles ont le même poids. Pour la résolution du problème de sac-à-dos, nous utilisons la *procédure first fit* présentée dans la section 5.5.2.

Concernant la détermination de la fenêtre de temps, sa longueur, *t*, est déterminée par *max*(r'_k, première disponibilité de machine) où r'_k est la date d'entrée de la tâche *k* dans le système (sa date de disponibilité par contre sera calculée par l'algorithme) parmi les tâches qui ne sont pas encore affectées à une machine. *k* est un paramètre qui varie de 1 à *N* où *N* est le nombre de tâches dans le problème. Le début de la fenêtre de temps est toujours 0. Ainsi, ce qui change dans l'intervalle [0, *t*], c'est seulement la variable *t*.

L'algorithme définit la longueur de la fenêtre de temps dans l'étape 3.1.2. Chaque fois qu'un batch est créé en utilisant *PFF*, le paramètre *t'* est calculé pour le lancement du cycle de lavage. L'étape 3.1.5 sert à respecter la durée minimale de pré désinfection, qui est de 15 minutes pour toutes les tâches d'un batch, et aussi à redéfinir le début d'exécution du batch. Après chaque itération de la boucle « tant que » avec une valeur différente pour la variable *k*, une nouvelle valeur pour la fonction objectif (*DMP*) est calculée. L'étape 3.3 choisit ainsi la plus petite valeur de *DMP*.

TIH

1. Trier les tâches en ordre croissant des dates d'arrivé r'_j : L_0. Poser $L_1 = L_0$
2. Fixer la valeur initiale de *DMP*, sol_{in}, égale à un très grand nombre
3. Pour k = 1 à N
    - 3.1. Tant que L_1 n'est pas vide
        - 3.1.1. Si le nombre de tâches dans L_1 est plus petit que k, fixer l = nombre de tâche dans L_1, else $l = k$,
        - 3.1.2. Définir la longueur de la fenêtre de temps comme t = max(date d'arrivée de la $l^{ième}$ tâche, première disponibilité de machine)
        - 3.1.3. Appliquer *PFF* sur les tâches dont les dates d'arrivée sont inférieures ou égales à t. Effacer de L_1 les tâches mises dans un batch
        - 3.1.4. Parmi les tâches mises dans le batch dans l'étape 3.1.3, trouver la tâche dont le début de pré désinfection est le plus grand : pre_{max}
        - 3.1.5. Fixer t' = max(t, pre_{max} + 15)
        - 3.1.6. Dès que t' est atteint, lancer un cycle de lavage avec le batch précédemment créé. Calculer la nouvelle disponibilité de la machine sur laquelle le batch est exécuté
    - Fin tant que
    - 3.2. Soit sol_{DMP} la valeur obtenue pour *DMP*
    - 3.3. Si $sol_{in} > sol_{DMP}$, poser $sol_{in} = sol_{DMP}$
    - 3.4. Fixer $L_1 = L_0$

Fin pour

4. Fixer la solution finale, sol_{finale}, égale à sol_{in}

Exemple illustratif : Soient 3 tâches dont les tailles, les dates d'arrivées et les débuts de pré désinfection sont les suivants :

- $v_1 = 3$, $t_1 = 5$, $r_1 = 10$,
- $v_2 = 2$, $t_2 = 5$, $r_2 = 20$,
- $v_3 = 4$, $t_3 = 20$, $r_3 = 45$.

Soient deux machines de capacité 6 qui sont disponibles à l'instant 0 et soit la durée d'exécution d'un batch 60 minutes.

Résolution de l'exemple illustratif :

Pour k = 1 :

Les tâches dans L_1 sont : 1, 2, 3

Alors, la limite de la fenêtre de temps est la date d'arrivée de la tâche qui se trouve au premier indice de la liste L_1. Donc, la fenêtre de temps est limitée à l'instant 10 (t = 10). On applique *PFF* sur les tâches dont la date de d'arrivée est inférieure ou égale à 10. Ainsi, la tâche 1 est mise dans un batch. A l'étape 3.1.5, on calcule le moment, t', pour lancer l'exécution de ce batch en prenant en compte la durée minimale de pré désinfection. Ainsi, on obtient t' = 20. Le batch est exécuté sur la première machine disponible, la machine 1, à l'instant 20.

Les tâches dans L_1 sont : 2, 3
La valeur du *k* est toujours 1 et il y a une machine disponible à l'instant 0. Donc, on crée un batch avec une seule tâche qui se trouve au premier indice de la liste L_1. Ainsi, un batch est créé avec la tâche 2 et est exécuté sur la machine 2 à l'instant 20.

Les tâches dans L_1 sont : 3
La tâche 3 est exécutée toute seule dès qu'il y a une machine disponible, donc à l'instant 80.

Calcul pour DMP
Les tâches 1 et 2 ont 15 minutes de pré désinfection. Donc, aucune pénalité pour elles. La durée de pré désinfection de la tâche 3 est 60, ce minutes ce qui implique une pénalité de 40 minutes. Alors, la valeur du *DMP* est 40/3 pour *k* égal à 1.

Pour k = 2 :
Les tâches dans L_1 sont : 1, 2, 3
La fenêtre du temps est calculée en fonction de la date d'arrivée de la tâche au deuxième indice de la liste L_1. Alors, à l'étape 3.1.3, on applique *PFF* sur les tâches 1 et 2. En prenant en compte la durée minimale de pré désinfection à l'étape 3.1.4, la date de début d'exécution du batch qui contient les tâches 1 et 2 est 20.

Les tâches dans L_1 sont : 3
La tâche 3 est exécutée toute seule dès qu'elle est suffisamment pré désinfectée. Elle est exécuté sur la machine 2 à l'instant 45.

Calcul pour DMP
Seule la tâche 3 a une pénalité de 5 minutes. Donc, la valeur du *DMP* est mise à jour et on trouve 5/3.

Pour k = 3 :
Les tâches dans L_1 sont : 1, 2, 3
La limite supérieure de la fenêtre du temps est la date d'arrivée de la tâche au $3^{ième}$ indice de la liste L_1. On applique *PFF* sur toutes les tâches. Ainsi, les tâches 1 et 2 sont mises dans un batch, mais il ne reste plus assez d'espace pour la tâche 3. Le batch est exécutée dès que toutes les tâches sont pré désinfectées au moins 15 minutes.

Les tâches dans L_1 sont : 3
La valeur de *k* est toujours égale à 3. Mais d'après l'étape 3.1.1, nous fixons *l* à 1. La tâche 3 est exécutée toute seule dans un deuxième batch.

Calcul pour DMP
La valeur finale du *DMP* est 5/3.

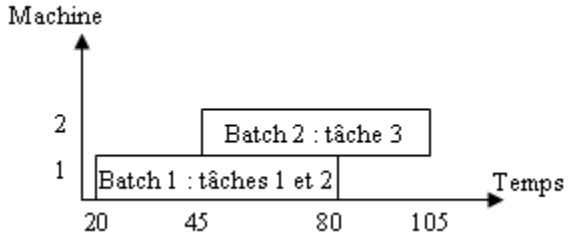

Figure 6.3. Résolution de l'exemple illustratif avec *TIH*

La complexité de l'algorithme est bornée par $0(N^3 logN)$. *k* varie de 1 à *N*. Pour chaque valeur différente de *k*, une boucle de « tant que » est exécutée *N* fois dans le pire cas. L'étape dominante pour le calcul de complexité est l'étape où *PFF* est appliqué. *PFF* a une complexité de $0(NlogN)$ dans son pire cas. Donc, la complexité de pire cas de *TIH* est $0(N^3 logN)$.

6.4. Expérimentations numériques

Notre but dans cette partie est de tester l'efficacité des méthodes présentées dans cette section. Rappelons que nous avons deux critères à tester : le *DMP* et le nombre de batch créés. La taille et la date d'arrivée des tâches dans les instances sont générées en fonction des explications données dans la section 4.6.1. La seule différence des instances dans cette partie est l'affectation d'une date de début de pré désinfection pour toutes les tâches. Pour les types d'arrivée 1 et 2, *i.e.* arrivée aléatoire et ramassage de 20 minutes, nous avons généré la date de début de pré désinfection des tâches suivant la formule suivante : $t_j = r'_j$ – U[5 ; 25] avec t_j la date de début de pré désinfection et r'_j la date d'arrivée dans le service de stérilisation pour la tâche *j*, et U une distribution uniforme. En fonction de nos observations, nous avons supposé que l'écart maximal et minimal entre le début de pré désinfection d'une tâche et son arrivée dans le service de stérilisation peut être 25 minutes et 5 minutes, respectivement. Pour le type d'arrivée 3, *i.e.* ramassage de 40 minutes, l'écart entre le début de prédesinfection et l'arrivée d'une tâche dans le service peut être un peu plus grande. Nous avons donc généré la date de début de pré désinfection des tâches suivant la formule suivante : $t_j = r'_j$ – U[5 ; 40] pour les instances de type 3. Nous avons codé les modèles *PLNE* en C++ pour faire appel à CPLEX. *TIH* est également codé en C++.

6.4.1. Performance des méthodes de résolution pour le *DMP* sur les petites instances

Cette partie est consacrée à évaluer les résultats d'un point de vue pratique. Pour toutes les instances, nous voulons voir si la valeur de la durée de pré désinfection dépasse les 20 minutes, *i.e.* la durée idéale de pré désinfection. D'abord, nous nous intéressons à la valeur moyenne du dépassement moyen de la durée idéale de pré désinfection pour chaque combinaison de nombre tâches/nombre de machines. Grâce au regroupement des instances en fonction des arrivées des ensembles de DMR, *i.e.* instances de type 1, 2 et 3 (*cf.* partie 4.6.1), les caractéristiques des instances sont similaires dans un groupe. Ainsi, il est raisonnable de calculer la moyenne du *DMP* dans un groupe donné pour voir la performance des différentes méthodes de résolution.

Dans le tableau 6.1, nous comparons ε_{DMP}, *CompA* et *TIH* entre eux. Pour chacune de ces méthodes, nous présentons la moyenne du *DMP* et les durées de résolution moyennes. Puisque ε_{DMP} donne la solution optimale pour le critère *DMP*, nous montrons aussi le gap moyen des solutions trouvées par *CompA* et *TIH* par rapport aux résultats de ε_{DMP}. La formule utilisée pour le gap sur une instance est la suivante : (solution de X – solution de ε_{DMP}) / (solution de X) où X représente *CompA* et *TIH*, respectivement. Puis, nous calculons la moyenne sur toutes les instances résolues pour toutes les combinaisons de nombre de tâches/nombre de machines. D'après les tests numériques, *CompA* trouve une solution dans le délai de résolution (fixé à 1 heure comme précédemment) alors qu'il y a quelques instances pour lesquelles ε_{DMP} ne peut pas trouver la solution optimale dans un délai d'une heure.

D'après le tableau 6.1, toutes les instances sont résolues à l'optimal pour le critère *DMP* par ε_{DMP} et *CompA* dans le délai d'une heure pour les instances du 1er type. Quand on regarde les moyennes de *DMP* et son gap par rapport à ε_{DMP}, nous voyons que *TIH* donne de bons résultats. De plus, son gap par rapport à ε_{DMP} a tendance à diminuer quand le nombre de machines croît. Notons que toutes les durées d'exécution sont quelques millisecondes avec *TIH*.

Pour les 2ème et 3ème types d'instances, *CompA* trouve toujours une solution dans le délai d'une heure. Par contre, pour les instances de 15 tâches et le 2ème type, l'optimalité avec ε_{DMP} n'est jamais garantie en présence de 4 machines. La même observation est valide pour les cas de 3 et 4 machines et le 3ème type d'instances. Une fois que le délai d'exécution est atteint, nous arrêtons CPLEX et nous récupérons les résultats de ε_{DMP}. Même si l'algorithme est arrêté avant d'avoir trouvé la solution optimale, toutes les solutions trouvées pour *DMP* sont encore une fois les mêmes que celles trouvées par *CompA*. Concernant les

résultats de *TIH*, nous voyons que le gap moyen entre *TIH* et ε_{DMP} est assez petit et même égal à 0 dans les cas de 4 machines et le type d'instance 3.

Tableau 6.1. Comparaison du DMP et durée de résolution avec les différentes méthodes de résolution

Type inst.			*CompA*		ε_{DMP}		*TIH*		*Gap CompA*	*Gap TIH*
	Nbr. tâches	Nbr. mach.	*Moyenne de DMP*	*dmr*	*Moyenne de DMP*	*dmr*	*Moyenne de DMP*	*dmr*		
1	10	1	19.3	< 1	19.3	< 1	21.4	≈ 0	0	0.18
	10	2	2	< 1	2	< 1	2.3	≈ 0	0	0.05
	10	3	0.2	< 1	0.2	< 1	0.3	≈ 0	0	0.09
	10	4	≈ 0	< 1	≈ 0	< 1	≈ 0	≈ 0	0	0.03
	15	1	30.9	101	30.9	640	34.8	≈ 0	0	0.11
	15	2	6.7	324	6.6	1395	8.2	≈ 0	0	0.14
	15	3	1.7	812	1.7	1975	2.9	≈ 0	0	0.11
	15	4	0.4	2104	0.4	2866	0.5	≈ 0	0	0.09
2	10	1	32.5	< 1	32.5	< 1	44.2	≈ 0	0	0.2
	10	2	5.8	< 1	5.8	< 1	7.8	≈ 0	0	0.16
	10	3	0.3	< 1	0.3	< 1	0.4	≈ 0	0	0.07
	10	4	≈ 0	< 1	≈ 0	< 1	≈ 0	≈ 0	0	0.03
	15	1	36	196	36	441	41	≈ 0	0	0.13
	15	2	9.5	312	9.5	1050	12	≈ 0	0	0.11
	15	3	5.9	551	5.9	1859	6.3	≈ 0	0	0.02
	15	4	5.3	1368	5.3	>3600	5.7	≈ 0	0	0.03
3	10	1	17	< 1	17	< 1	18.8	≈ 0	0	0.07
	10	2	4.5	< 1	4.5	< 1	5	≈ 0	0	0.05
	10	3	3.11	< 1	3.11	< 1	3.1	≈ 0	0	0.01
	10	4	3	< 1	3	< 1	3	≈ 0	0	**0**
	15	1	43	81	43	1344	53	≈ 0	0	0.15
	15	2	16	185	16	3289	16.7	≈ 0	0	0.07
	15	3	13	386	13	>3600	13.3	≈ 0	0	0.02
	15	4	12.8	543	12.8	>3600	12.8	≈ 0	0	**0**

Type inst. : Type d'instance, ***Moyenne de DMP*** : Moyenne de *DMP* en minutes, ***dmr*** : durée moyenne de résolution, ***Gap CompA*** *:* Gap moyen pour *CompA*, ***Gap TIH*** *:* Gap moyen pour *TIH*

La durée d'exécution de ε_{DMP} et *CompA* dépend non seulement de $PLNE_{DMP}$, mais aussi de $PLNE_{batch}$. En ce qui concerne la durée d'exécution avec $PLNE_{batch}$, il n'est, en effet, pas pratique pour la résolution de grandes instances, par exemple, des instances contenant 100 tâches. Mais il peut résoudre les cas à 10 et 15 tâches en quelques millisecondes.

Nous voyons que *TIH* donne de bons résultats comparé aux résultats optimaux (ou aux meilleurs résultats au cas où l'optimalité n'est pas assurée). Dans 60% des cas en moyenne, *TIH* donne les mêmes résultats que *CompA* et ε_{DMP} pour le *DMP*.

Nous allons maintenant élargir notre analyse sur les gaps et les répartir dans les déciles. Pour ce faire, nous prenons la différence entre le gap maximum et le gap minimum et divisons la différence par 10 afin de trouver la longueur d'un décile. Ce calcul est fait pour toutes les combinaisons de nombre de tâches/nombre de machines. Evidemment, l'intervalle du premier décile commence par le gap minimum et va jusqu'à « minimum gap + 1**longueur* » avec *longueur* la longueur des déciles. De la même façon, l'intervalle du seconde décile commence par la dernière valeur du gap précédent et va jusqu'à « minimum gap + 2**longueur* », et ainsi de suite. Notre but est de montrer le nombre d'instances dont le gap tombe dans ces intervalles.

Dans le tableau 6.2, nous montrons pour chaque combinaison de nombre de tâches/nombre de machines d'abord le gap minimal et le gap maximal entre *TIH* et ε_{DMP}. Les intervalles de décile qui suivent la colonne de gap maximal montrent le nombre d'instances pour lesquelles le gap est inclus dans ces intervalles de déciles. Il est clair que la plupart des gaps de *TIH* sont dans le premier décile. Nous pouvons maintenant examiner si nous obtenons des bonnes durées de pré désinfection avec *CompA*, ε_{DMP} et *TIH*.

Même si la durée idéale de pré désinfection est de 20 minutes pour les ensembles de DMR, 20 à 30 minutes de pré désinfection est une durée considérée comme correcte dans les services de stérilisation. Ainsi, un dépassement de 10 minutes de la durée optimale de pré désinfection, *i.e.* 30 minutes de pré désinfection, est accepté d'un point de vue opérationnel. Nous pouvons alors essayer d'observer la distribution des différentes durées de pré désinfection pour les instances testées. Après avoir montré le dépassement moyen de la durée idéale de pré désinfection, nous montrons maintenant les durées moyennes de pré désinfection des instances testées. Nous définissons 4 intervalles de temps (en minutes) qui sont $\leq$30, >30 et $\leq$40, >40 et $\leq$50, > 50. Nous indiquons le nombre d'instances ayant une durée moyenne de pré désinfection correspondant à chaque intervalle sur le tableau 6.3.

Tableau 6.2. Nombre d'instances dans les déciles par rapport au gap entre *TIH* et ε_{DMP}

Type Inst.	Nbr. tâches	Nbr. mach.	*min*	*max*	*dec1*	*dec2*	*dec3*	*dec4*	*dec5*	*dec6*	*dec7*	*dec8*	*dec9*	*dec10*
1	10	1	0	0.35	15	1	2	4	1	2	3	1	0	1
	10	2	0	0.65	25	0	1	0	2	1	0	0	0	1
	10	3	0	1	26	0	1	1	0	0	1	0	0	1
	10	4	0	1	28	1	0	0	0	0	0	0	0	1
	15	1	0	0.26	9	1	1	2	3	4	1	4	1	4
	15	2	0	0.43	7	6	3	2	5	2	1	2	1	1
	15	3	0	0.34	15	0	0	3	0	3	3	3	0	3
	15	4	0	0.46	18	3	3	0	0	0	0	3	0	3
2	10	1	0	0.56	10	0	4	2	0	1	12	0	0	1
	10	2	0	0.51	11	0	2	2	0	12	1	0	1	1
	10	3	0	1	26	1	0	0	1	0	1	0	0	1
	10	4	0	1	28	1	0	0	0	0	0	0	0	1
	15	1	0	0.37	6	1	4	7	2	1	5	3	0	1
	15	2	0	0.43	14	1	2	3	2	5	2	0	0	1
	15	3	0	0.27	27	0	0	0	0	0	0	0	0	3
	15	4	0	0.36	29	0	0	0	0	0	0	0	0	1
3	10	1	0	0.56	21	0	1	1	3	0	1	1	0	2
	10	2	0	0.38	22	1	0	0	1	0	2	3	0	1
	10	3	0	0.2	28	0	0	0	0	0	0	0	1	1
	10	4	0	0	30	0	0	0	0	0	0	0	0	0
	15	1	0	0.41	10	4	2	2	7	0	1	0	3	1
	15	2	0	0.36	12	1	4	2	3	4	0	1	0	3
	15	3	0	0.17	28	0	0	0	0	0	0	0	0	2
	15	4	0	0	30	0	0	0	0	0	0	0	0	0

D'après le tableau 6.3, les résultats de *TIH* ne sont pas très différents que ceux de *CompA*. Par contre, il y a un point qui attire notre attention. Pour tous les types d'instance, il n'y a pas de tâche ayant une durée de pré désinfection plus petite que 30 minutes s'il y a une seule machine dans le système. Ceci peut s'expliquer par le fait que les données ont été déterminées à partir d'un cas avec 4 machines, et ainsi, avoir une seule machine est insuffisant pour traiter correctement les DMR qui arrivent au service de stérilisation. La durée de pré désinfection plus élevée est observée pour les instances de 3ème type. Avec un ramassage régulier de 40 minutes, l'attente d'un ensemble de DMR est assez longue avant d'atteindre le service de stérilisation, et donc, la durée de trempage des tâches est assez élevée même lors de leur entrée dans le stock de lavage.

Tableau 6.3. Nombre d'instances dans les intervalles en fonction de la valeur de pré désinfection moyenne

Type. inst.			*CompA*				*TIH*			
	Nbr. tâches	Nbr. mach.	[15, 30]	]30, 40]	]40, 50]	]50, ∞[	[15, 30]	]30, 40]	]40, 50]	]50, ∞[
1	10	1	0	20	5	5	0	18	5	7
	10	2	27	3	0	0	25	5	0	0
	10	3	30	0	0	0	30	0	0	0
	10	4	30	0	0	0	30	0	0	0
	15	1	0	10	5	15	0	8	4	18
	15	2	25	5	0	0	24	6	0	0
	15	3	30	0	0	0	28	2	0	0
	15	4	30	0	0	0	29	1	0	0
2	10	1	0	7	8	15	0	5	7	18
	10	2	15	10	5	0	15	10	5	0
	10	3	29	1	0	0	29	1	0	0
	10	4	30	0	0	0	30	0	0	0
	15	1	0	4	5	21	0	4	2	24
	15	2	18	12	0	0	17	13	0	0
	15	3	22	8	0	0	20	10	0	0
	15	4	24	6	0	0	22	8	0	0
3	10	1	0	2	3	25	0	0	2	28
	10	2	5	22	2	1	3	20	5	2
	10	3	8	22	0	0	6	22	2	0
	10	4	6	24	0	0	6	24	0	0
	15	1	0	0	0	30	0	0	0	30
	15	2	0	15	15	0	0	13	15	2
	15	3	0	14	16	0	0	14	15	1
	15	4	1	20	9	0	1	20	9	0

Dans les 1er et 2ème types d'instance, les durées de pré désinfection sont meilleures que le 3ème type pour le cas de 2 machines. Mais, ce résultat n'est pas très bon d'un point de vue opérationnel. Comme notre but est de supprimer l'étape de rinçage manuel, nous pouvons conclure que quand il y a moins de 3 laveurs dans le système, la suppression du rinçage manuel peut engendrer de longues durées de pré désinfection.

Comme montré sur le tableau 6.3, *CompA* et *TIH* donnent de bons résultats quand il y 3 machines ou plus dans le système sauf pour le 3ème type d'instances. Pour ces cas, il est alors possible de supprimer le rinçage manuel. De

plus d'après nos visites des services de stérilisation, il y a toujours plus de 2 laveurs dans l'étape de lavage.

Rappelons que d'un point de vue économique, nous avons un deuxième critère d'optimisation qui est la minimisation du nombre de cycles de lavage. Ce critère n'étant pas autant important que le premier critère, nous allons maintenant essayer d'examiner la performance de nos méthodes de résolution pour ce critère secondaire.

6.4.2. Performance des méthodes de résolution pour le nombre de batch créés sur les petites instances

Le nombre de cycles de lavage lancés est un critère économique. A chaque fois qu'un cycle de lavage est lancé, un coût est généré, dû à l'utilisation d'eau, d'électricité, etc. Cependant, le coût de lancement d'un cycle de lavage est négligeable comparé au coût d'achat des DMR. D'après nos contacts avec les services de stérilisation, le coût d'un cycle de lavage est estimé être autour de 1 euro[3]. Nous traitons donc la minimisation du nombre de batch comme un critère secondaire. Nous allons maintenant discuter les résultats donnés par nos méthodes de résolution.

Les expérimentations numériques montrent que le nombre de batch créés par *CompA* est toujours égal au nombre de batch créés par ε_{DMP} pour toutes les instances résolues à l'optimal par ε_{DMP}. Pour les autres instances qui ne sont pas résolues jusqu'à l'optimalité par ε_{DMP}, par contre, le nombre de batch trouvés par *CompA* est plus petit que le nombre de batch donnés par ε_{DMP} (Rappelons que même si une instance n'est pas résolue jusqu'à l'optimalité par ε_{DMP} dans un délai d'une heure, l'algorithme est arrêté et la solution trouvée est enregistrée).

Pour chaque combinaison de nombre de tâches/nombre de machines, nous montrons sur le tableau 6.4 le nombre moyen de batch créés par $PLNE_{batch}$, $PLNE_{DMP}$, ε_{DMP}, *CompA* et *TIH*. Nous comparons ces résultats aussi aux nombres de batch donnés par $PLNE_{batch}$. Nous calculons le gap sur le nombre de batch avec la formule : (nombre de batch formés par *X* - nombre de batch formés par $PLNE_{batch}$)/ nombre de batch par *X* où *X* signifie ε_{DMP}, *CompA* et *TIH*, respectivement.

[3] Pour calculer le coût d'un cycle de lavage, nous pouvons utiliser la formule suivante :
Puissance du laveur*tarif d'électricité heures pleines + consommation d'eau.
Application numérique pour un laveur de marque *GETINGE* avec une puissance de 9800W qui consomme 15 litres d'eau par cycle de lavage: 9.8kw * 0.1154€/kw + 15*0.003 € = 1.1792 €.

Tableau 6.4. Comparaison pour le nombre de batch formé

Type inst.	Nbr. tâches	Nbr. mach.	$PLNE_{batch}$	$PLNE_{DMP}$	ε_{DMP}	*CompA*	*TIH*	*Gap* ε_{DMP}	*Gap CompA*	*Gap TIH*
	10	1	3	4.4	3.5	3.5	3.5	0.14	0.14	0.14
	10	2	3	6.3	3.8	3.8	4.9	0.21	0.21	0.38
	10	3	3	7.9	7.3	7.3	6.2	0.58	0.58	0.51
	10	4	3	8.1	8	8	7.6	0.62	0.62	0.6
	15	1	5	6.9	5.2	5.2	5.1	0.03	0.03	0.01
	15	2	5	9.5	7	7	7.6	0.28	0.28	0.34
	15	3	5	11.8	10	10	9.9	0.5	0.5	0.5
1	15	4	5	13.1	11.6	11.6	11.5	0.56	0.56	0.56
	10	1	3.1	4.5	3.8	3.8	4.2	0.18	0.18	0.26
	10	2	3.1	6.6	4	4	4.9	0.22	0.22	0.36
	10	3	3.1	8.3	4.5	4.5	5.5	0.28	0.28	0.41
	10	4	3.1	8.7	5	5	6.1	0.34	0.34	0.45
	15	1	4.9	7.2	5.2	5.2	5.2	0.05	0.05	0.05
	15	2	4.9	8.8	6.8	6.8	7.2	0.27	0.27	0.31
	15	3	4.9	11.2	8.7	8.4	8.4	0.43	0.41	0.41
2	15	4	4.9	12.6	12.6	8.5	8.6	0.61	0.42	0.44
	10	1	3	4.4	3.5	3.5	3.6	0.14	0.14	0.16
	10	2	3	6.5	3.3	3.3	4.3	0.09	0.09	0.3
	10	3	3	7.8	3.6	3.6	4.5	0.16	0.16	0.33
	10	4	3	8.7	3.6	3.6	4.5	0.16	0.16	0.33
	15	1	4.9	6.4	5	5	5.4	0.02	0.02	0.09
	15	2	4.9	8	6.5	6.2	6.5	0.24	0.2	0.24
	15	3	4.9	11.6	11.6	6.7	6.8	0.57	0.26	0.27
3	15	4	4.9	12.3	12.3	6.7	6.8	0.6	0.26	0.27

Comme montré sur le tableau 6.4, le nombre de batch est amélioré considérablement par *CompA*, comparé à $PLNE_{DMP}$. Pour les autres instances, le nombre de batch trouvé est souvent plus petit avec *CompA*. Pour la comparaison avec *TIH*, *CompA* est généralement plus performant, mais la différences n'est jamais plus grande que 1,1 batch. Les 3 dernières colonnes du le tableau 6.4 montrent la différence moyenne pour ε_{DMP}, *CompA* et *TIH* par rapport aux solutions optimales pour le nombre de batch. Nous voyons que le gap diminue quand le nombre de machines diminue aussi. Il est clair que s'il y a peu de machines dans le système, la probabilité d'avoir une machine disponible diminue. Ainsi, avoir peu de machines implique d'augmenter l'attente des tâches avant leur exécution. Dans ce cas, quand un batch est ouvert, plusieurs tâches peuvent être disponibles pour l'exécution simultanément. Par conséquent, un problème de sac-à-dos est résolu et le nombre de tâches à mettre dans le batch est maximisé. Plus on met de tâches dans un batch, mieux on utilise la capacité de la machine et

donc, le nombre de batch créé s'approche de la borne inférieure. Cette observation est aussi valide pour les instances du 3ème type. Comme pour *CompA* et *TIH*, le gap moyen est plus proche de la borne inférieure sur les instances de type 3 que les instances de type 1 et 2. En effet, quand il y a un ramassage régulier de 40 minutes des ensembles de DMR, la probabilité d'avoir davantage de tâches arrivant simultanément est plus grande que pour les arrivées dans les instances de type 1 et 2. Par conséquent, si un grand nombre de tâches arrivent en même temps, une bonne solution serait de maximiser le remplissage des machines.

Dans ces deux dernières parties, nous avons vu que *TIH* donnait de bon résultats pour le *DMP* et pouvait concurrencer *CompA* pour le nombre de batch. Nous allons maintenant tester *TIH* sur des instances de taille réelle pour lesquelles ni *CompA* ni ε_{DMP} ne sont applicables.

6.4.3. Performance de *TIH* sur les instances de taille réelle

Dans les parties précédentes, nous avons traité les instances de petite taille pour lesquelles les méthodes exactes et l'heuristique étaient capables de trouver une solution dans une courte durée. Mais comme précisé dans les chapitres précédents, un hôpital de taille moyenne peut avoir 40 à 50 interventions chirurgicales par jour, ce qui équivaut à traiter 40 à 50 ensembles de DMR dans le service de stérilisation. Par conséquent, les méthodes utilisant les PLNE peuvent être inefficaces sur des grandes instances. Ainsi, nous appliquons *TIH* sur des grandes instances contenant un grand nombre de tâches.

Les instances contiennent 50 tâches et sont testées sur 4 machines ce qui est le cas dans les services de stérilisation que nous étudions. Les ensembles de DMR arrivent au service de stérilisation d'une façon irrégulière. Un cycle de lavage dure une heure. Pour chaque combinaison de nombre de tâches/nombre de machines, nous testons 30 instances. Les instances ont été créées comme expliqué au début de la partie 6.4.

Nous comparons les résultats donnés par *TIH* à la stratégie couramment utilisées dans les services de stérilisation, qui est le *FIFO* online. Nous montrons d'abord les valeurs minimales, maximales et moyennes des *DMP* trouvées par *TIH* et *FIFO online* (*cf.* tableau 6.5). Ensuite, nous définissons des intervalles de longueurs égales pour montrer le nombre d'instances ayant un *DMP* correspondant à ces intervalles (*cf.* figure 6.4). Pour afficher le nombre d'instances appartenant à différentes gammes de durée moyenne de pré-désinfection, nous définissons 5 intervalles consécutifs. Pour chaque intervalle de temps, nous vérifions les valeurs de *DMP* pour les instances testées.

Tableau 6.5. Comparaison entre *TIH* et *FIFO online* pour *DMP* en présence de 4 machines et d'une arrivée irrégulière des tâches

TIH			*FIFO online*		
Min.	Max.	moy.	Min.	Max.	moy.
0.41min.	2.16 min.	1.09 min.	22.9 min.	34.2 min.	30.7 min.

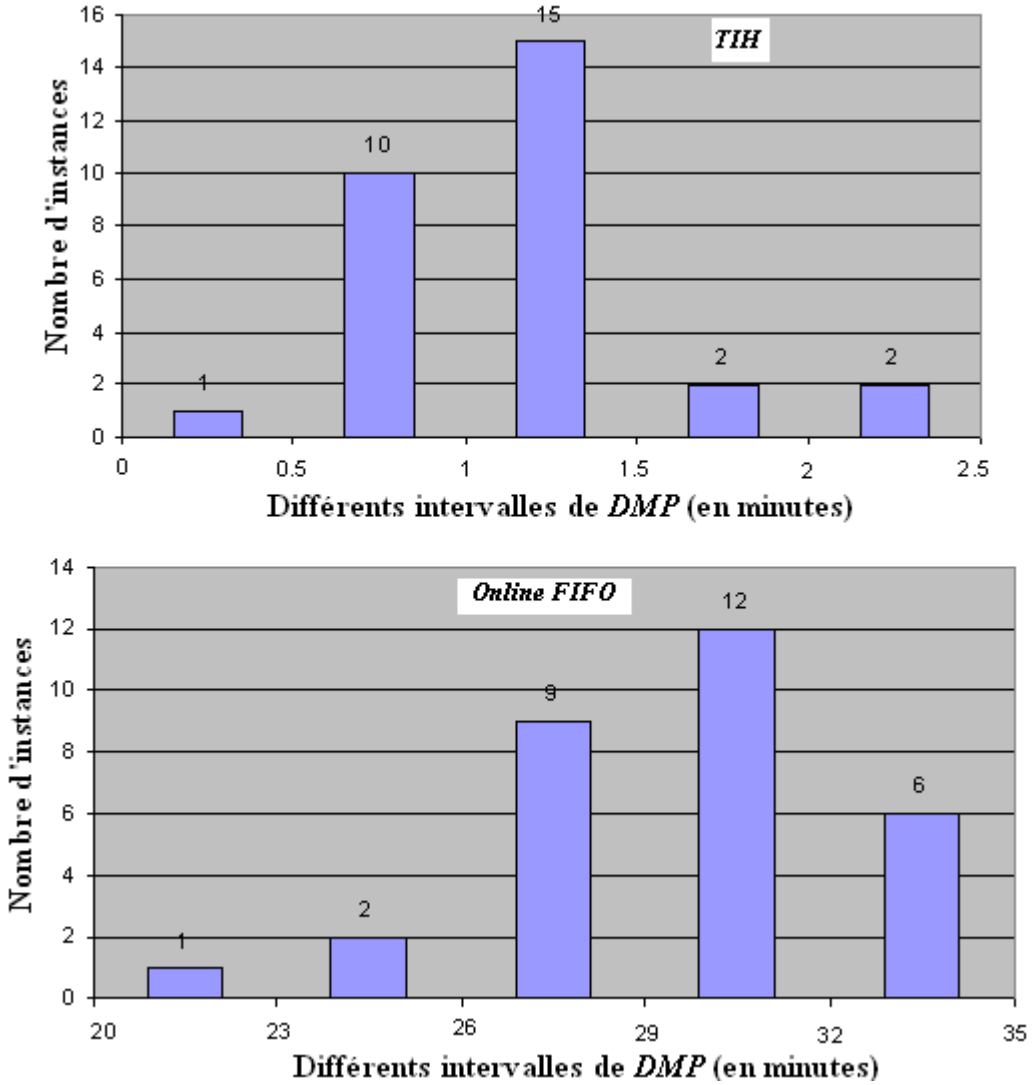

Figure 6.4. Répartition des instances de type 1 dans des intervalles, en fonction de la valeur trouvée pour le DMP, pour *TIH* et *FIFO online*

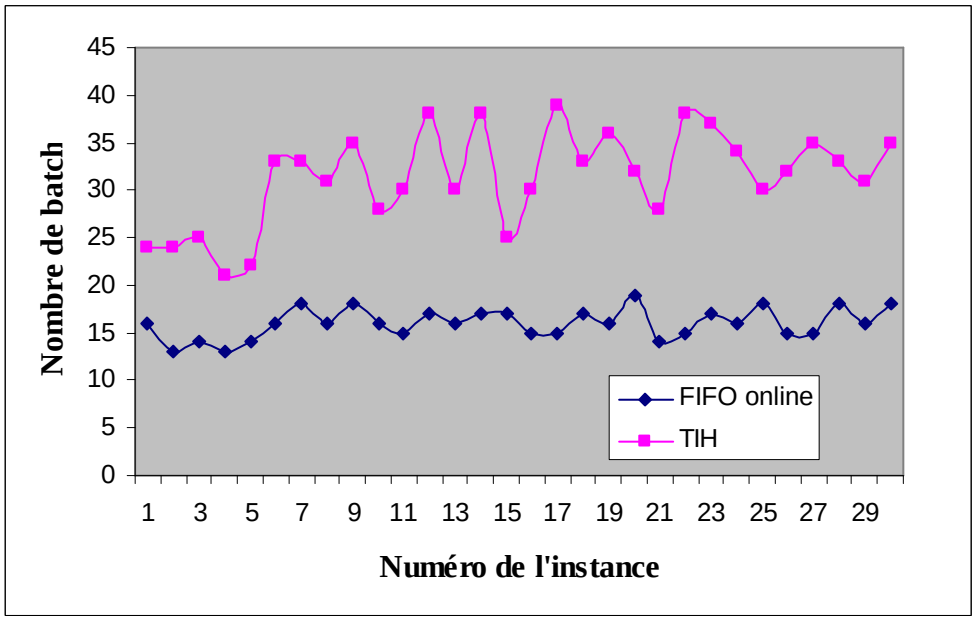

Figure 6.5. Nombre de batch par *TIH* et *FIFO online* pour l'instance type 1

Il est clair que *TIH* est plus performant que *FIFO online* pour le critère *DMP*. Nous pouvons noter qu'une borne inférieure naturelle pour *DMP* est de 0 minute. Ainsi, nous voyons que *TIH* donne des résultats assez proches de cette borne inférieure alors que *FIFO online* en est loin. Nous allons maintenant étudier les performances de ces méthodes pour le critère de la minimisation du nombre de batch. Pour chaque instance, de 1 à 30, nous montrons sur la figure 6.5 le nombre de batch formés par *TIH* et *FIFO online*.

Clairement, *FIFO online* est plus efficace que *TIH* pour le deuxième critère d'optimisation. Puisque *FIFO online* lance un cycle de lavage seulement quand il y a un ensemble de DMR qui n'entre pas dans le batch actuel, le but de *FIFO online* peut être vu comme la maximisation de la capacité utilisée des machines. Par contre, cette stratégie provoque de longues durées de pré désinfection.

Par curiosité, nous étendons notre étude et faisons la même comparaison lorsque le ramassage des DMR est régulier, *i.e.* les instances de type 2 et 3. Pour chaque type d'instances, 30 instances sont testées. Nous montrons sur le tableau 6.6 les valeurs minimales, maximales et moyennes des *DMP* trouvées par *TIH* et *FIFO online*, et le nombre d'instances appartenant aux différents intervalles de *DMP* sur la figure 6.6.

Tableau 6.6. Comparaison, entre *TIH* et *FIFO online*, du *DMP* en présence de 4 machines avec les 2ème et 3ème types d'instances

Type inst.	*TIH*			*FIFO online*		
	Min.	Max.	Moy.	Min.	Max.	Moy.
2	2.1 min.	5.4 min.	3 min.	29.1 min.	42 min.	36 min.
3	6.5 min.	8 min.	7 min.	39.5 min.	53,4 min.	46 min.

Pour le 2ème et le 3ème type d'instances, tous les dépassements de la durée idéale de pré désinfection sont inférieurs à 10 minutes, ce qui implique une bonne durée de pré désinfection pour les *DMR* d'un point de vue pratique. Par contre *FIFO online* ne peut assurer de courtes durées de pré désinfection. Continuons maintenant avec le 2ème critère d'optimisation.

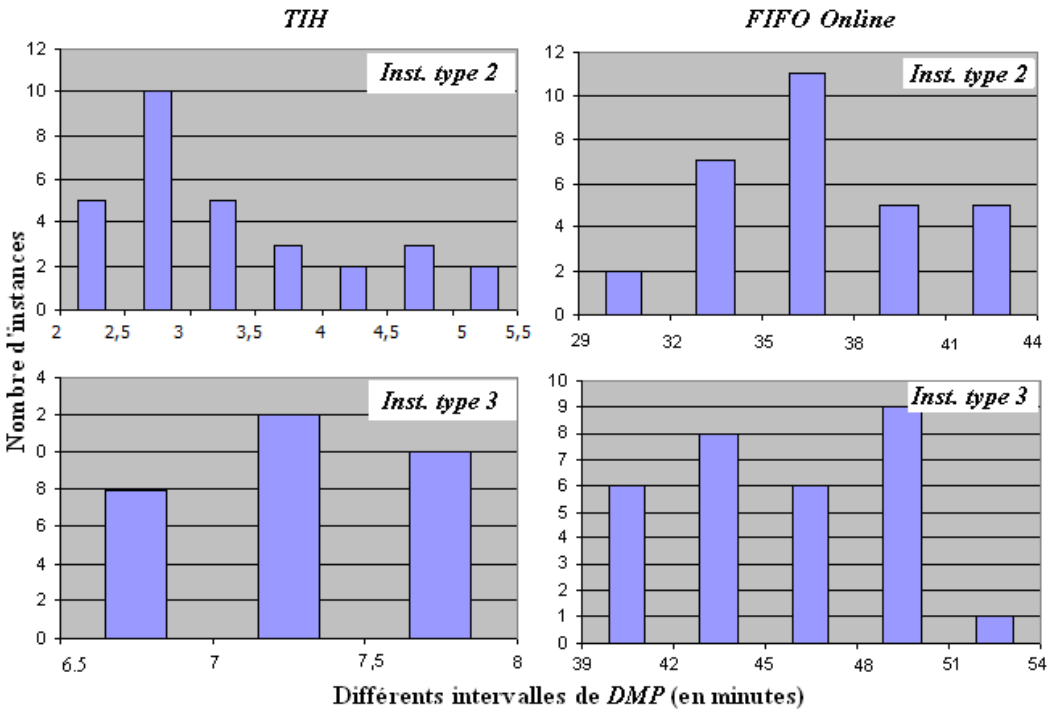

Figure 6.6. Répartition des instances de type 2 et 3 dans des intervalles, en fonction de la valeur trouvée pour le DMP, pour *TIH* et *FIFO online*

Sur la figure 6.7, nous montrons le nombre de batch formés pour toutes les instances résolues par *TIH* et *FIFO online*. Pour l'instance de type 2, *FIFO online* est toujours plus efficace que *TIH* alors que le gap entre eux diminue quand on passe aux instances de type 3.

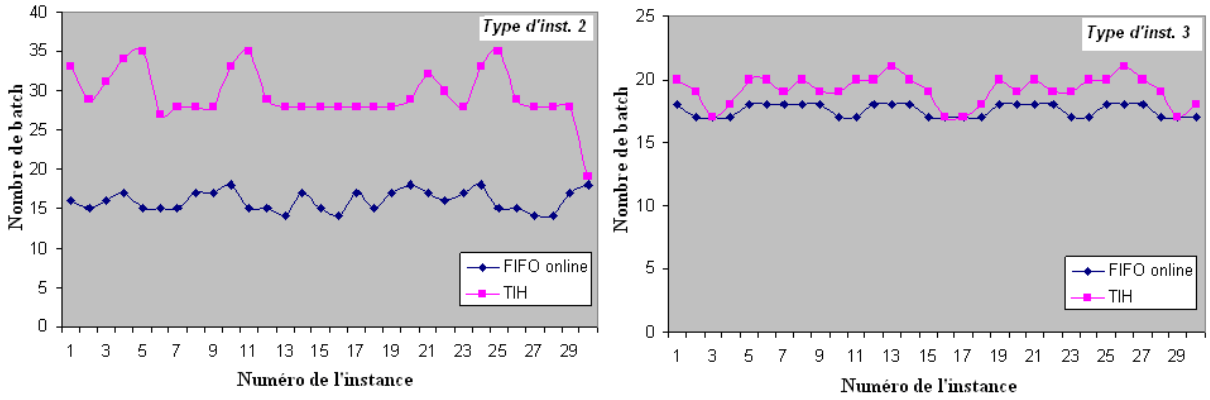

Figure 6.7. Nombre de batch par *TIH* et *FIFO online* pour les instances de type 2 et 3

Ces résultats fournissent des informations importantes concernant la minimisation du *DMP* et le nombre de batch. Nous observons qu'en commençant par les instances de type 1, les valeurs de *DMP* augmentent légèrement avec les types d'instances. L'inverse est observé pour le nombre de batch créés. Moins on forme de batch, plus la valeur du *DMP* augmente. *FIFO online* est un bon exemple pour cette observation. *FIFO online* a tendance à maximiser la capacité utilisée des batch. Pour ce faire, il fait attendre les tâches dans le stock de lavage qui provoque naturellement une augmentation dans les durées de pré desinfection. Alors qu'avec un lancement régulier des cycles de lavage, comme *TIH* essaie de faire, on obtient plus de batch tout en minimisant l'attente des tâches dans le stock de lavage, ce qui diminue les durée de pré desinfection.

6.5. Conclusion

Ce chapitre était plus orienté sur l'application que les 2 chapitres précédents. Notre but étant la suppression du rinçage manuel dans le système, nous avons besoin de garantir des bonnes durées de pré désinfection.

En fonction de nos investigations auprès de quelques services de stérilisation (service de stérilisation de CHU de Grenoble, service de stérilisation du Centre Hospitalier Privé St. Martin de Caen, ainsi que les services de stérilisation ayant participé à l'enquête EESS [36]), une bonne durée de pré désinfection pour les DMR peut être définie comme entre 20 et 30 minutes. 15 minutes étant obligatoires, 50 minutes étant la limite supérieure et 20 minutes étant l'idéale, nous avons défini une fonction objectif minimisant le dépassement moyen de la durée idéale de pré désinfection pour tous les ensembles de DMR traités dans une journée.

En l'absence de rinçage manuel, nous avons vu que la stratégie naturelle du chargement des laveurs, *i.e. FIFO* online, provoquait des durées excessives pour la pré désinfection des *DMR*. Nous avons donc proposé des méthodes exactes et une heuristique pour que les DMR aient une bonne durée de pré désinfection tout en supprimant le rinçage manuel du système. Parmi les méthodes exactes, nous avons utilisé le *PLNE* présenté dans le chapitre précédent en le modifiant légèrement et proposé un modèle *ε-contrainte* qui a pour but, en critère secondaire, de minimiser le nombre de cycles de lavage lancés. Nous avons aussi proposé un algorithme composé qui fonctionne à l'inverse du modèle *ε-contrainte*. Concernant la méthode heuristique, un algorithme (de complexité polynomiale) assez puissant pour les grandes instances est développé.

D'après les tests numériques, nos méthodes de résolution garantissent des bonnes durées de pré désinfection en présence de 3 et 4 machines. Pour des instances de taille réelle, nous avons comparé l'heuristique *TIH* à la stratégie *FIFO online* et vu que *TIH* est assez efficace pour avoir de bonnes durées de pré désinfection. En ce qui concerne la minimisation du nombre de cycles de lavage par contre, *FIFO online* est évidemment plus performant que *TIH*. Mais le coût d'un cycle de lavage est estimé être assez petit, autour d'un euro. Donc, le deuxième critère n'étant pas aussi important que le premier, les tests numériques montrent qu'avec *TIH*, le rinçage manuel peut être enlevé du système tout en assurant des bonnes durées de pré désinfection.

Conclusion de la partie 2

Nous avons consacré la partie 2 aux méthodes d'optimisation offline. Nous avons d'abord étudié la minimisation de la durée totale de lavage et la minimisation des encours du stock de lavage dont les fonctions objectifs correspondantes sont la minimisation du C_{max} et la minimisation de la $\sum C_j$, respectivement, dans la littérature d'ordonnancement. Puisque l'étape de lavage est généralement un goulet d'étranglement, l'étude de ces fonctions objectifs nous semble utile pour faire face à ce problème. Nous nous sommes ensuite dirigé vers une autre fonction objectif afin d'optimiser la durée de pré désinfection des DMR. Rappelons que de longues durées de pré désinfection peuvent corroder les DMR. Un autre but à l'étude de cette fonction objectif est d'évaluer l'opportunité de supprimer le poste de rinçage manuel. Puisque les laveurs appliquent un rinçage automatique avant de commencer le lavage et qu'un seul rinçage est suffisant si la durée de pré désinfection d'un DMR n'est pas élevée, il sera possible d'envisager la suppression du poste du rinçage manuel.

La partie 2 nous montre qu'il est possible de charger plus efficacement les laveurs si l'arrivée de DMR est connue à l'avance. Pour chacune des fonctions objectifs, nous avons comparé nos méthodes d'optimisation à une stratégie naturelle de chargement des laveurs, *FIFO online*. Le test sur plusieurs instances montre que les méthodes proposées sont capables d'augmenter la performance de l'étape de lavage d'un point de vue opérationnel. Notons que dans la plupart des services de stérilisation, il n'y a pas une connexion directe entre les blocs opératoires et le service de stérilisation. Par contre, l'emploi du temps des interventions chirurgicales peut être utilisé pour connaître l'arrivée des DMR à l'avance. Voire, un système d'information liant les blocs opératoires au service de stérilisation peut aider à gérer les incertitudes comme par exemple l'annulation d'une intervention chirurgicale, le retard d'arrivée d'un ensemble de DMR, etc.

~ PARTIE 3 ~

Prise en compte des incertitudes et impact de l'optimisation du chargement des laveurs sur le processus de stérilisation

Cette partie est consacrée à l'étude de l'impact de nos méthodes d'optimisation, développées dans la partie 2, sur le processus de stérilisation. Nous considérons pour cela que le processus de stérilisation est soumis à des incertitudes sur les dates d'arrivée des DMR, ce qui nous conduit à étudier des cas online et semi-online. Après avoir fait une revue de la littérature sur l'ordonnancement par batch online et semi-online, nous proposons une version semi-online de l'heuristique *TIH*. Cette nouvelle heuristique est en fait une méthode de ré-optimisation, qui lance l'heuristique *TIH* à chaque fois qu'une tâche est en retard ou en avance par rapport à ce qui a été prévu. Ensuite, une heuristique online est présentée. Nous comparons les performances de ces deux heuristiques, d'abord sur l'étape de lavage, puis sur le processus de stérilisation complet, grâce à un modèle de simulation construit en ARENA.

Chapitre 7 : Motivations et Etat de l'art en ordonnancement par batch online

7.1. Motivations

Dans la partie 2, nous nous sommes concentrés sur des méthodes d'optimisation offline pour le chargement des laveurs. Ces travaux nous ont permis de voir que l'optimisation offline aide à améliorer le flux de dispositifs médicaux au niveau de l'étape de lavage. Cependant, l'hypothèse faite de connaître à l'avance toute l'information concernant la date d'arrivée de DMR au service de stérilisation n'est pas toujours valide pour tous les services de stérilisation. De plus, comme dans tous les systèmes de production, l'organisation des hôpitaux n'empêche pas qu'il y ait des incertitudes concernant l'arrivée de DMR au service de stérilisation.

Les incertitudes concernant l'arrivée des DMR au service de stérilisation peuvent par exemple se traduire par un retard d'arrivée dans le service de stérilisation, des DMR qui peuvent manquer dans un ensemble, etc. En présence de telles incertitudes, la connaissance à l'avance de l'arrivée de DMR au service de stérilisation est imparfaite. Alors, même si les méthodes offlines sont puissantes pour l'optimisation de l'étape de lavage, elles ne sont pas complètement adaptées aux services de stérilisation. Dans cette partie, nous nous intéressons donc aux méthodes qui permettent de faire face à de telles incertitudes.

Nous nous concentrons ici surtout sur les méthodes semi-online et online. L'étape de lavage étant toujours notre centre principal d'intérêt, nous proposons dans les chapitres suivants une méthode semi-online, puis une méthode purement online pour l'optimisation du chargement des laveurs. Ces méthodes sont plus précisément des algorithmes déterministes de type semi-online et online. Nous commençons par tester l'efficacité de ces méthodes sur l'étape de lavage. Ensuite, nous étudions leur impact sur le processus global de stérilisation.

Cette partie est plus orientée application que la partie 2. Nous avons vu dans le chapitre 6 que la minimisation de la durée de pré désinfection, tout en enlevant le rinçage manuel, était un objectif relativement important. Dans cette partie aussi, notre objectif est de proposer des méthodes qui assurent de bonnes

durées de pré désinfection pour les DMR, en l'absence de rinçage manuel. Rappelons que garantir de bonnes durées de pré désinfection est important pour éviter l'usure des DMR par le liquide pré désinfectant.

Nous testons nos méthodes de résolution via des modèles de simulation. Le logiciel avec lequel nous avons construit nos modèles est ARENA (version 11.0). Pour le codage des algorithmes semi-online et online, nous utilisons le module VBA (*Visual Basic for Applications*) d'ARENA. Ces modèles de simulation seront utilisés aussi dans la dernière section pour tester l'impact de la minimisation du nombre de cycles de lavage dans le cas où les ensembles de DMR sont simultanément disponibles.

L'organisation de cette partie du mémoire est la suivante : nous présentons une revue de la littérature en ordonnancement par batch online/semi-online dans la section suivante. Ensuite, nous parlons de la stratégie naturelle, *i.e. FIFO online*, appliquée généralement dans l'étape de lavage et discutons son efficacité sur les durées de pré désinfection. Dans le chapitre suivant, nous proposons un algorithme semi-online, inspiré de l'heuristique *TIH* présentée dans le chapitre 6, et un algorithme online. Après avoir testé leurs performances sur les durées de pré désinfection, nous testons l'impact de cette optimisation sur le service de stérilisation via des modèles de simulation.

7.2. Etat de l'art sur l'ordonnancement par batch online

Alors que la revue de la littérature sur l'ordonnancement par batch offline est assez large, il n'existe pas autant d'articles sur l'ordonnancement par batch online dans la littérature. Presque tous les articles étudiant l'ordonnancement par batch online prennent en compte des tâches de taille unitaire. A notre connaissance, il n'y a qu'un seul article dans la littérature qui étudie l'ordonnancement online de tâches ayant des tailles différentes [140]. Comme cela a été fait pour les articles sur l'ordonnancement par batch offline, nous allons également montrer dans cette section différents travaux sur l'ordonnancement par batch online via des tableaux.

Commençons par le cas où les tâches ont des tailles unitaires. Dans ces problèmes, nous avons encore deux types de travaux qui dépendent de la capacité de la machine : machine de capacité infinie ou machine de capacité finie. B étant la capacité d'une machine et N le nombre de tâches dans le problème, si la capacité de la machine est finie (limitée), alors un batch peut contenir au plus B tâches. Si, par contre, sa capacité est infinie (illimitée), alors on a $B \geq N$. Nous regroupons les articles en fonction du nombre de machines.

Le tableau 7.1 montre les différentes hypothèses prises en compte, le ratio de compétitivité, et la fonction objectif étudiée pour les articles cités. Si l'inverse n'est pas précisé, les tâches ont des durées d'exécution différentes. Donnons maintenant la définition du ratio de compétitivité pour un problème de minimisation :

Définition 4. [54] Soit *A(I)* la solution trouvée par l'algorithme online *A* et soit *OPT(I)* la solution optimale trouvée par un algorithme offline pour une instance *I*. Alors pour un problème de minimisation, le ratio de compétitivité de l'algorithme *A* est défini par :

$$R_A = \max_{\forall I} \{A(I)/OPT(I)\}$$

La quatrième colonne du tableau 7.1 montre le ratio de compétitivité des travaux cités. La première colonne précise le nombre de machines ainsi que leur capacité, la deuxième colonne indique les hypothèses supplémentaires prises en compte (s'il y en a). La colonne 3 cite les articles, et enfin, la colonne 5 mentionne la fonction objectif étudiée. Pour les notations utilisées dans les tableaux 7.1 et 7.2, le lecteur peut se référer à l'annexe C.

De la même façon, le tableau 7.2 présente les travaux online en présence de plusieurs machines parallèles. Nous avons expliqué les hypothèses supplémentaires par des notes de bas de page. Mais pour une meilleure compréhension, vous trouverez des explications plus détaillées dans le glossaire des notations.

Tableau 7.1. Ordonnancement *p-batch online* des tâches sur une machine

Groupe	*Hypothèses supp.*	*Ref.*	*Ratio de compétitivité*	*Fonc. obj.*
Une machine, capacité infinie		[116]	$(\sqrt{5}+1)/2$	C_{max}
	r_0-r_1	[145]	$(\sqrt{5}+1)/2$	
	r_0-r_1, F	[97]	$(9-\sqrt{5})/4$	
	*préemption**	[44]	3/2	
	Nombre de préemption au plus une seule fois pour chaque tâche*	[45]	3/2	
	$p_j=p$	[120]	1.618	
	$r_j \nearrow p_j$	[120]	1.618	
	F2, p_j *comp.*	[46]	1.7808	
	$p_j \geq cp_{j+1}$	[88]	$(\sqrt{c^2+4}-c)/2+1$	
	$p_j \leq cp_{j+1}$	[88]	$(\sqrt{c^2+4}-c)/2+1$	
	r(t)	[142]	1.382	
	d_j	[130]	2	L_{max}
	$p_j \nearrow d_j$	[141]	$(\sqrt{5}+1)/2$	
	$p_j \in [p, (1+\varphi)p] \ \forall j, d_j$	[37]	$1+\varphi$	
	F, $setup_j$ *comp*, $p_j=p$.	[48]	2	$\sum C_j$
		[19]	10/3	$\sum w_j C_j$
Une machine, capacité finie		[115]	*2*	C_{max}
	$p_j \in [p, (1+\varphi)p] \ \forall j \ d_j$	[37]	$1+\varphi$	
	r_0-r_1	[145]		
	préemption limitée*	[20]	*3/2*	
	capacité machine=2	[115]	*7/4*	
	d_j	[130]	*3*	L_{max}
	d_j	[130]	$(\sqrt{5}+1)/2$	
		[19]	$4+\varepsilon$	$\sum w_j C_j$

***** Contrairement à la préemption classique, si un batch est préempté, tout le travail fait est perdu et les tâches du batch ne sont pas exécutées.

Tableau 7.2. Ordonnancement *p-batch online* des tâches sur plusieurs machines

Groupe	*Hypothèses supp.*	*Ref.*	*Ratio de compétitivité*	*Fonc. obj.*
Machines parallèles, capacité infinie		[89]	[1] $1+\gamma$	C_{max}
		[145]	$(\sqrt{5}+1)/2$	
	*préemption**	[143]	$(5-\sqrt{5})/2$	
	$p_j=p$	[146]	[2] $1+\beta_m$	
	2 machines	[103]	$\sqrt{2}$	
	2 machines	[131]	$\sqrt{2}$	
	Préemption limitée*	[47]	1.366	
	$p_j=p$, *prec.*	[13]	[3] ρ^m	$\sum w_j C_j$
Machines parallèles, capacité finie	$p_j=p$	[146]	$(\sqrt{5}+1)/2$	C_{max}
	$p_j=p$, *prec.*	[13]	2	$\sum C_j$
		[147]	[4] $4(2-1/B+\varepsilon)$	$\sum w_j C_j$

Nous voyons que la plupart des travaux concerne le cas d'une machine de capacité infinie. Le cas de machines parallèles avec une capacité finie est bien entendu plus intéressant dans notre étude.

En ce qui concerne l'ordonnancement par batch online des tâches avec des tailles différentes, Yongqiang et Enyu [140] étudient la minimisation du makespan sur une seule machine. Ils étudient d'abord un cas particulier où il n'y a que deux dates de disponibilité différentes. L'algorithme qu'ils proposent a un ratio de compétitivité égal à 119/44. Ensuite, pour le cas général, ils proposent un algorithme ayant un ratio de compétitivité égal à 7/2. A notre connaissance, il n'y pas d'article sur l'ordonnancement par batch online en présence de machines parallèles et de tâches de tailles différentes.

[1] $\gamma = (\frac{\sqrt{m^2+4}-m}{2})$ avec m le nombre de machines

[2] $(1+\beta_m)^{m+1}=\beta_m+2$ avec m le nombre de machines et β_m *un paramètre tel que* $0<\beta_m<1$

[3] m le nombre de machines, ρ^m un nombre positif tel que $\rho^{m+1} - \rho = 1$

[4] ε un nombre très petit et positif, B la capacité de la machine

7.3. Stratégie naturelle des services de stérilisation : *FIFO online*

Nous avons présenté la stratégie *FIFO online* dans les sections précédentes. Le principe de *FIFO online* est le même que celui de l'algorithme *next fit* (*NF*). Les batch sont remplis avec des tâches consécutives. Dès qu'une tache n'entre pas dans le batch existant, le batch est fermé et un autre batch est ouvert pour cette tâche. Evidemment *FIFO online* peut provoquer de longues durées d'attente. Lorsqu'il n'y a pas de rinçage manuel, si la durée d'attente d'une tâche augmente, sa durée de pré désinfection augmente également. Du coup, la performance de *FIFO online* peut donc être assez faible pour le critère de minimisation des durées de pré désinfection (cf. chapitre 6). La faiblesse de *FIFO online* est liée à la fermeture des batch. Un batch est fermé si et seulement s'il est plein ou s'il n'y a pas suffisamment d'espace pour une prochaine tâche. Puisque dans la configuration online, l'instant d'arrivée d'une tâche est inconnu, *FIFO online* peut causer des attentes excessives, provoquant l'augmentation de la durée d'attente et de pré désinfection des tâches.

Même si *FIFO online* est fréquemment appliqué dans les services de stérilisation pour le chargement des laveurs, le seuil de remplissage n'est pas toujours à 100%, c'est-à-dire que l'on n'attend pas forcément qu'un batch soit rempli à 100% pour le lancer. Au contraire, ce seuil est souvent autour de 80%. Donc, si un batch est rempli à 80% de la capacité d'un laveur, alors un cycle de lavage est lancé si un laveur est disponible. Si aucun laveur n'est disponible, le batch n'est pas fermé tout de suite, il est laissé ouvert jusqu'à la prochaine disponibilité d'un laveur. Si des tâches arrivent quand le batch attend la libération d'un laveur, elles sont mises dans le batch si leur taille le permet.

Dans le chapitre suivant, nous allons proposer d'autres algorithmes capables de fournir de bonnes durées de pré désinfection pour les ensembles de DMR. Nous comparons ces algorithmes à la stratégie *FIFO online* pour différents seuils de remplissage : 70%, 80%, 90% et 100%.

7.4. Conclusion

Les méthodes offlines que nous avons étudiées dans la partie 2 nous ont permis de voir qu'il est possible d'appliquer d'autres méthodes que *FIFO online* pour le chargement des laveurs à l'étape de lavage. Ces méthodes sont capables de diminuer l'attente des DMR et d'optimiser leurs durées de pré désinfection. L'inconvénient de ces méthodes est qu'elles nécessitent de connaître à l'avance toutes les données concernant les ensembles de DMR, ce qui n'est pas toujours possible. Un ensemble de DMR peut très bien arriver au service de stérilisation avec un peu de retard si l'intervention pour laquelle il est utilisé ne se déroule pas

comme prévu. Notre but est alors de proposer des algorithmes capables de réduire l'impact des incertitudes sur les dates d'arrivée au service de stérilisation des ensembles de DMR.

Nous nous intéressons dans cette partie d'abord aux méthodes d'optimisation semi-online et online. Nous commencerons par un cas semi-online où une partie de l'information sur les ensembles de DMR est connue. Ensuite, nous continuons avec un autre cas où le service de stérilisation de l'hôpital n'est pas du tout en communication avec les blocs opératoires. Alors, aucune information relative à l'avance sur l'arrivée des ensembles de DMR ne pourra être connue à l'avance. Pour ce type de service, nous proposons un algorithme online.

Le critère le plus important pour la stérilisation est bien sûr la qualité de stérilisation des DMR. Rappelons que le liquide de pré désinfection peut avoir un effet de corrosion sur les DMR si cette étape dure considérablement longtemps. La stratégie naturelle pour le remplissage des laveurs n'aide pas beaucoup pour optimiser la durée de pré désinfection de DMR. Nous allons donc proposer des algorithmes permettant d'optimiser les durées de pré désinfection.

Dans le chapitre suivant, nous présentons un algorithme semi-online et un algorithme online. Après avoir testé leurs performances pour la durée de pré désinfection, nous allons essayer d'étudier leur impact sur un service de stérilisation. Pour ce faire, nous allons construire un modèle de simulation en utilisant le logiciel ARENA.

Chapitre 8 : Approches semi-online et online et leurs impacts sur les durées de pré désinfection

8.1. Introduction

Notons qu'un algorithme *semi-online* est un algorithme qui a une information partielle sur les données d'un problème. Nous avons vu que *TIH* était efficace pour la minimisation du *DMP* et donc, nous nous sommes posé la question : est-ce que *TIH* peut être modifié afin de gérer les incertitudes concernant l'arrivée des DMR ? Dans ce but, nous avons intégré *TIH* dans un modèle de ré-optimisation. La démarche de cette approche est de faire exécuter *TIH* à chaque fois qu'il y a une anomalie concernant l'arrivée d'un ensemble de DMR. Une anomalie concernant les arrivées au service de stérilisation peut être de 2 types : un ensemble de DMR peut arriver après sa date d'arrivée prévue, ou, avant sa date d'arrivée prévue. La première étape est d'exécuter *TIH* comme si toutes les données étaient déterministes et connues à l'avance. De cette façon, on obtient un premier planning pour le lavage des ensembles de DMR. Un planning pour le lavage consiste à savoir comment construire des batch et quand lancer les cycles de lavage. Ensuite, dès qu'une tâche/un ensemble de DMR est en avance ou en retard par rapport à la date prévue pour son arrivée au service de stérilisation, *TIH* est ré-exécuté afin de trouver un nouveau planning pour le lavage.

Un algorithme online, par contre, est un algorithme qui n'a aucune connaissance sur les données d'un problème à l'avance. Les données sont connues au fur et à mesure. Dans notre problème d'ordonnancement par batch, cela correspond à ne pas connaître la taille, le début de pré désinfection et la date d'arrivée d'un ensemble de DMR avant son entrée dans le service de stérilisation. Dès qu'il entre au service de stérilisation, ces données deviennent connues. L'algorithme online que nous proposons a également pour objectif de minimiser le *DMP*.

8.2. Heuristiques semi-online et online

Nous développons dans ce chapitre deux algorithmes : un algorithme semi-online basé sur l'algorithme *TIH* présenté dans le chapitre 6, puis un

algorithme online. Ces deux algorithmes ont pour but de minimiser le dépassement moyen de la durée idéale de pré désinfection (*DMP*).

8.2.1. Heuristique de ré-optimisation : *TIH correctif*

Cette heuristique utilise un planning d'arrivée prévisionnel pour savoir quelle tâche entre à quel moment dans le système. Ici, le système désigne bien sûr le service de stérilisation et l'entrée des tâches dans le système se fait dans le stock de lavage. En fonction de ce premier planning d'arrivée, un premier ordonnancement est trouvé et les dates de début d'exécution des batch sont déterminées. Par contre, il suffit qu'une tâche n'arrive pas comme prévu pour bouleverser l'ordonnancement précédent. Alors en pratique, l'ordonnancement des tâches serait mis à jour à chaque arrivée. Soit L_{prevue} la liste des dates d'arrivée prévisionnelles des tâches. L_{prevue} contient les tâches en ordre croissant de leur date d'arrivée au service de stérilisation. L_{prevue} contient aussi des informations prévisionnelles sur la taille et le début de pré désinfection des tâches. Pour déterminer les dates de lancement des batch prévisionnnel, l'algorithme *TIH* est utilisé. Ainsi, des instants de lancement des batch sont trouvés. Mais, il suffit qu'il y ait une tâche en retard/avance pour que *TIH* soit réexécuté pour déterminer de nouveaux instants prévisionnels pour le lancement des batch.

Expliquons maintenant l'algorithme étape par étape :

- Etape 1 : La première étape consiste à copier la liste L_{prevue} dans une autre liste, L_1, sur laquelle *TIH* va faire des calculs.
- Etape 2 : A cette étape, *TIH* est exécuté avec les tâches de L_1 pour déterminer une pré affectation des tâches aux batch et des batch aux machines. De cette façon, nous obtenons une prévision pour la date de lancement des batch. Après avoir fait la pré affectation des tâches aux batch, les 2 prochaines étapes gèrent les anomalies concernant les dates d'arrivée des tâches.
- Etape 3 : Lorsqu'un batch se remplit prévisionnellement, si une tâche arrive avant sa date d'arrivée prévue, alors dans l'étape 3, la liste L_1 est mise à jour et *TIH* est ré exécuté.
- Etape 4 : Si une tâche n'est pas encore arrivée alors qu'elle aurait dû l'être, l'étape 4 considère que cette tâche arrivera en même temps que la tâche suivante. Donc, la date d'arrivée de la tâche en retard est mise à jour de telle manière que sa nouvelle date d'arrivée devient égale à celle de la tâche qui la suit, et *TIH* est ré exécuté.
- Etape 5 : Dès que le moment de lancement d'un batch est atteint, la procédure *PFF* de *TIH* est exécutée pour mettre les tâches ensemble dans un batch. Une fois que le batch est fermé, il est exécuté sur la première machine qui devient disponible parmi toutes les machines.

TIH correctif

1. Copier la liste L_{prevue} dans la liste L_1
2. Exécuter *TIH* sur les tâches de L_1 afin de faire une pré affectation des tâches aux batch. Pour chaque batch *b* créé, poser $début_b$ comme la date prévue pour le début d'exécution du batch *b*
3. Quand une tâche arrive en avance,
    3.1. Mettre à jour sa date d'arrivée et ré-exécuter *TIH* afin de déterminer les nouveaux instants d'exécution de batch.
4. Quand une tâche est supposée arriver,
    4.1. Si elle n'est pas encore arrivée, mettre à jour sa date d'arrivée de telle façon qu'elle arrive avec la tâche qui la suit. Ré-exécuter *TIH* afin de déterminer les nouveaux instants d'exécution des batch.
5. A chaque fois qu'un nouveau $début_b$ est atteint, relancer la procédure *PFF* de *TIH* afin de déterminer les tâches à mettre dans le batch *b* et affecter le batch à la première machine disponible. Mettre à jour la disponibilité de la machine. Relancer *TIH* pour déterminer les nouvelles dates prévues pour le début d'exécution des batch.

En résumé, l'heuristique *TIH correctif* est composée de plusieurs étapes différentes qui sont : mise à jour des données des tâches qui ont une anomalie d'arrivée, détermination des moments de lancement, création d'un batch et son affectation sur une machine.

Notons que les tâches sont effacées de la liste L_1 si et seulement si elles sont affectées à une machine. Imaginons que toutes les tâches d'un batch *b* soient disponibles et il n'y ait pas de machine disponible (comme déjà dit, quand *TIH* affecte les tâches aux batch, les instants de début d'exécution des batch sont aussi calculés). Si, avant le lancement de ce batch, une tâche *j* dont la prévision est faite pour un autre batch arrive en avance ou en retard, le lancement de *PFF* dans l'étape 5 peut mettre cette tâche dans le batch *b*. Alors, la tâche *j* est exécutée dans le batch *b* alors qu'elle devait être mise dans un autre batch. Dans ce cas, la détermination des nouveaux instants pour le lancement devient nécessaire. Ainsi, *TIH* est relancé à la fin de l'étape 5. Notons aussi que *TIH* est utilisé uniquement pour déterminer les instants de lancement de batch. La création des batch se fait avec *PFF*. Rappelons que *PFF* trie les tâches par ordre croissant des dates de disponibilité (ici, par ordre croissant des dates de débuts de pré désinfection), puis il parcourt la liste contenant ces tâches pour créer un seul batch.

Dans la partie suivante, nous expliquons la démarche que nous avons suivie pour intégrer *TIH correctif* dans un modèle de simulation.

8.2.2. Intégration de *TIH correctif* dans un modèle construit en ARENA

Notre but dans cette partie est non seulement de proposer des algorithmes semi-online et online, mais aussi de voir leur impact sur le système de stérilisation entier. Dans ce but, nous avons préféré intégrer *TIH correctif* dans un modèle de simulation au lieu de le coder dans un langage comme cela a été fait pour les algorithmes proposés dans la partie 2. Nous présentons ici un modèle construit en ARENA comprenant seulement l'étape de lavage, modèle qui sera complété plus tard.

Nous avons utilisé la version 11.0 d'ARENA pour construire notre modèle qui représente, pour l'instant, uniquement l'étape de lavage. Il permet de visualiser l'arrivée des tâches au service de stérilisation ainsi que leur exécution sur les machines.

Nous nous sommes servis du module VBA pour lire des données et exécuter *TIH correctif*. Le modèle est composé de 3 grands blocs appelés : *horloge*, *arrivée des tâches* et *machines*. La figure 8.1 représente le modèle. Expliquons les blocs du modèle un par un.

Bloc *horloge* : Rappelons que *TIH correctif* s'exécute avec 2 types de données : les données prévues et les données réelles. Les données prévues représentent l'arrivée prévue des tâches alors que les données réelles sont pour les arrivées réelles. Ces données sont gardées dans les fichiers « .txt ». Le bloc horloge est chargé de la lecture des données prévues et il fonctionne avec des entités fictives qui correspondent aux arrivées prévues des entités réelles. Les entités fictives ne correspondent à aucune tâche, alors que les entités réelles représenteront les tâches.

Le module VBA1 lit les données prévues. Quand une tâche est censée entrer au service de stérilisation en fonction de sa date d'arrivée prévue, une entité fictive passe par VBA2 et le module VBA2 vérifie si la tâche correspondant à cette entité se trouve dans le stock de lavage. Si ce n'est pas le cas, c'est à dire, si une tâche ne se trouve pas dans le stock de lavage alors qu'il est l'heure de son entrée prévue au stock de lavage, cela signifie que la tâche a du retard.

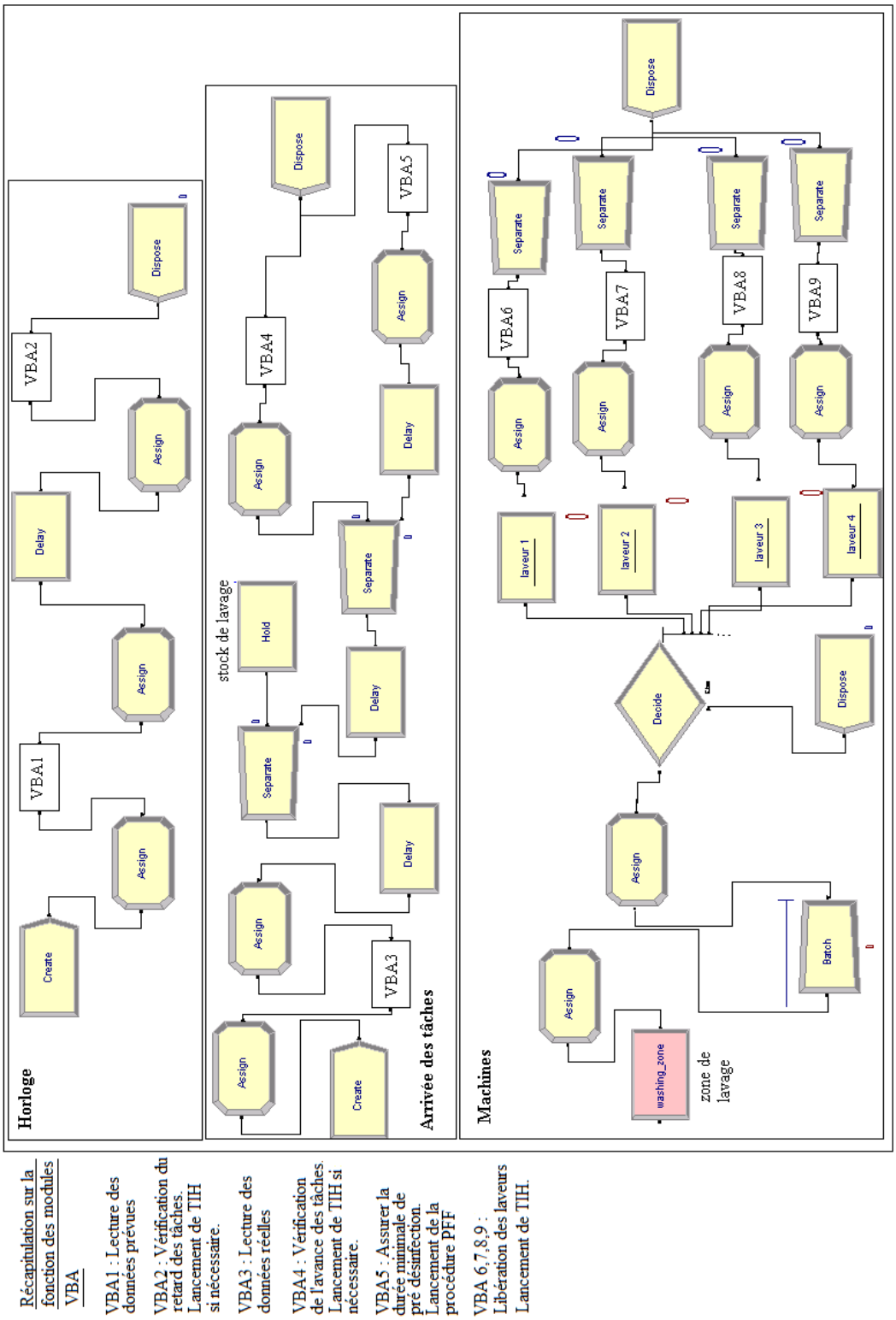

Figure 8.1. Modèle de simulation pour appliquer *TIH correctif*

Considérons le cas où la dernière tâche d'un problème contenant N tâches est retard. Puisqu'il n'y a plus de tâche suivant cette dernière, la mise à jour de sa date d'arrivée prévue peut provoquer une confusion. Afin de l'éviter, nous créons une dernière tâche, tâche N+1, dont la date d'arrivée prévue est égale à l'heure de fermeture du service de stérilisation. De cette façon, pour chaque tâche en retard, l'heure limite pour entrer dans le service est l'heure de fermeture du service. Nous supposons que toutes les tâches d'une instance arrivent au service de stérilisation soit avec un retard ou une avance, soit à l'heure et il n'y a jamais de cas où une tâche n'arrive pas au service de stérilisation.

En résumé, le bloc *horloge* déclenche l'étape 4 de *TIH correctif* à chaque passage d'une entité fictive par le bloc VBA2. Si nous avons un problème avec N tâches, le nombre maximal de fois que l'étape 4 est exécutée pour une tâche j ($j \leq N$) est égale à N+1-j. Ceci correspond au cas où, après la première mise à jour de la date d'arrivée prévisionnelle pour la tâche j, elle n'arrive toujours pas dans le service avec la tâche j+1, ni avec la tâche j+2, …, et arrive avec la tâche N+1 (fermeture du service). Dans le cas extrême, toutes les tâches sont en retard un nombre maximal de fois, et l'étape 4 de *TIH correctif* est exécutée $N(N+1)/2$ fois.

Bloc *arrivée des tâches* : Ce bloc représente l'entrée réelle des tâches dans le stock de lavage. Le module VBA3 est responsable de la lecture des données réelles. Quand une tâche arrive au service de stérilisation (*i.e.* sa date d'arrivée réelle est atteinte), une entité réelle, créé par le module « create », entre dans le stock de lavage. Lors de son entrée dans le stock de lavage, l'entité lui correspondant est dupliquée. L'entité copiée garde les mêmes données que celle qui est dupliquée (*i.e.* même valeur pour la taille, le début de pré désinfection et la date d'arrivée à l'étape de lavage). Une première copie de l'entité dupliquée passe par VBA4. Le module VBA4 détermine si la tâche est en avance permettant ainsi d'exécuter l'étape 3 de *TIH correctif*. De cette façon, nous mettons à jour la date d'arrivée de la tâche et re-déterminons les nouveaux instants de lancement des cycles de lavage. Quant à la deuxième copie de l'entité dupliquée, elle passe par le module VBA5 dès qu'elle est pré désinfectée au moins 15 minutes (*i.e.* la durée minimale pour la pré désinfection). Puisque le lancement d'un cycle de lavage demande au moins 15 minutes de pré désinfection pour toutes les tâches qu'il contient, le passage d'une entité par VBA5 nous permet de voir s'il est l'heure de lancer un cycle de lavage. Ainsi, en fonction de « $début_b$ » déterminé par *TIH correctif*, si l'heure actuelle est égale à un $début_b$, nous pouvons maintenant lancer un cycle de lavage. Pour celui-ci, nous exécutons la procédure *PFF* pour déterminer quelles tâches mettre dans le batch pour lequel un cycle de lavage va être lancé. Ainsi, l'étape 5 de *TIH correctif* est déclenchée. Une fois que toutes les tâches d'un batch sont déterminées, elles sont transférées à la *zone de lavage* qui se trouve dans le bloc *machines*.

Puisqu'une tâche ne peut arriver en avance qu'une seule fois (en avance par rapport à sa première date d'arrivée prévisionnelle ou bien après la mise à jour de sa date d'arrivée prévisionnelle due à un retard), le nombre de fois que l'étape 3 (également l'étape 5) est déclenchée au maximum une fois par tâche.

Bloc *machines* : Ce bloc représente les laveurs. L'entrée des tâches dans ce bloc se fait via un module de stationnement appelé « zone de lavage ». Quand toutes les tâches d'un batch sont présentes dans cette zone, un batch est formé et envoyé aux laveurs. Le batch est affecté au premier laveur disponible. Une fois que l'exécution d'un batch est terminée, il passe par un module VBA (VBA 6, 7, 8 ou 9). Le but de ces modules VBA est de libérer les machines auxquelles ils sont associées et de déclencher la fonction *PFF* s'il est l'heure de créer un batch. Donc, l'étape 5 de *TIH correctif* est aussi appelée par ces modules VBA. Si, dans un problème avec N tâches, chaque batch ne contient qu'une seule tâche, le nombre de fois que l'étape 5 est appelée N.

Le modèle que nous venons de décrire représente seulement l'étape de lavage. Dans la suşte, nous allons utiliser des aspects stochastiques pour les étapes suivant l'étape de lavage afin de tester l'impact de notre algorithme sur le processus global de stérilisation. Ainsi, le modèle de simulation présenté ici sera complété avec les autres étapes du processus de stérilisation dans la section suivante.

8.2.3. Heuristique online : *HO*

Contrairement à *TIH correctif*, l'algorithme online présenté ici ne nécessite aucune connaissance à l'avance sur les données. Toute information sur une tâche devient connue dès son arrivée au service de stérilisation.

Parallèlement à ce qui a été fait pour *TIH correctif*, nous intégrons aussi *HO* dans un modèle de simulation. Puisque le modèle construit pour *HO* ressemble à celui de *TIH correctif*, nous n'expliquons pas de nouveau le modèle de simulation. Nous présentons ci-après le pseudo code de *HO*.

Notations utilisées dans l'algorithme
t : un instant
$U(t)$: ensemble des tâches qui n'ont pas encore été mises dans un batch, mais qui sont dans le stock de lavage, à l'instant t.
att : durée d'attente dans le stock de lavage
dp : durée minimale de pré désinfection
pre_j : date de début de pré désinfection de la tâche j

Algorithme online
1. Poser *t* égal à l'instant d'arrivée de la tâche qui entre dans le stock de lavage.
2. Trouver la tâche *j* telle que $pre_j = \min\{pre_{j'} \mid j' \in U(t)\}$.
3. Si $pre_j + att \leq t$, et s'il y a une machine disponible,
   3.1. Appliquer *PFF* sur les tâches j'' ($j'' \in U(t)$) telles que $pre_{j''} + dp \leq t$
   3.2. Lancer un cycle de lavage
   3.3. Si le stock de lavage est vide, aller à l'étape 1. Sinon, aller à l'étape 2.
Fin si
4. Si $pre_j + att \geq t$ ou s'il n'y a pas de machine disponible
   4.1. Soit $t' = \max$ ($pre_j + att$, première machine disponible)
   4.2. Si une tâche j'' arrive entre $[t, t']$,
      4.2.1. Poser *t* égal à l'instant d'arrivée de la tâche j'', aller à l'étape 2.
      Sinon
      4.2.2. Poser *t* égal à t', aller à l'étape 2.
   Fin si
Fin si

Nous avons deux paramètres dans cette heuristique : *att* et *dp*. Le paramètre *att* représente la durée d'attente dans le stock de lavage pour une première tâche dont le début de pré désinfection est le plus petit parmi les autres tâches. L'idée de l'heuristique est de faire attendre cette première tâche pendant une durée *att* après son début de pré désinfection. Dès que la tâche a attendu suffisamment longtemps et s'il y a une machine disponible, nous appliquons *PFF* sur les tâches dans le stock de lavage sachant que toutes les tâches doivent être pré désinfectées au moins *dp* minutes. Nous avons fixé *dp* à 15 minutes car 15 minutes est la durée minimale de pré désinfection pour les DMR. Puisque la durée idéale de pré désinfection est de 20 minutes, nous avons décidé que le paramètre *att* sera égal à 20 minutes. Bien sûr, ces valeurs peuvent être changées si nécessaire.

Nous pouvons maintenant faire des expérimentations numériques pour tester l'efficacité de nos algorithmes sur les durées de pré désinfection.

8.3. Expérimentations numériques

Notre étude dans cette section représente la simulation d'un service de stérilisation. Pour bien comprendre de quel service il s'agit et quelles données sont utilisées, donnons d'abord quelques explications avant de passer à l'analyse des résultats.

8.3.1. Données utilisées

Nous testons l'efficacité de nos algorithmes sur un cas réel. Dans ce but, nous avons créé des instances cela a été fait dans le chapitre 6 (*cf.* ci-dessous). Les modèles de simulation sont aussi inspirés du cas réel. Le service de stérilisation du Centre Hospitalier Privé St. Martin de Caen (CHPSM) est l'établissement qui a servi de référence pour nos modèles de simulation. D'après l'enquête *EESS*, l'organisation de la plupart des services de stérilisation se ressemble au niveau du nombre de postes dans le service, des capacités des laveurs ainsi que de la taille des ensembles de DMR. Le service de stérilisation du CHPSM est un exemple représentatif sur lequel nous pouvons tester nos heuristiques.

Rappelons que nous avons construit nos modèles de simulation avec ARENA 11.0. Pour construire notre modèle de simulation, nous avons considéré le modèle générique proposé par Ngo Cong [102] pour les services de stérilisation. Notre modèle contient 4 laveurs. Tous les laveurs ont la même capacité, égale à 6 DIN. La durée d'un cycle de lavage est de 60 minutes. Le service de stérilisation du CHPSM effectue un rinçage manuel. Mais, nous voulons voir si nos heuristiques sont capables de fournir de bonnes durées de pré désinfection sans rinçage manuel. Rappelons que le but du rinçage manuel est de faire attendre les DMR pour le lavage sans risque d'usure due à un séjour prolongé dans le liquide pré désinfectant. Nous excluons donc le rinçage manuel de nos modèles de simulation.

Le nombre d'ensembles de DMR qui arrivent au service est à peu près 50 par jour. Nous créons donc des instances contenant 50 tâches. En ce qui concerne l'arrivée et la taille des ensembles de DMR, nous créons les instances suivant les indications données pour la création des instances dans la partie 2. Rappelons brièvement cette création. D'après nos observations, nous pouvons avoir au maximum 36 tailles différentes pour les ensembles de DMR. La taille maximale peut être égale à la capacité d'un laveur. Ainsi, nous avons estimé la taille des tâches par la formule suivante : capacité laveur*U[1, 36]/36. Les ensembles de DMR arrivent au service en général un par un. Mais de temps de temps, il peut y en avoir 2 ou plus qui arrivent en même temps. D'après les observations faites au service de stérilisation du CHPSM, la différence entre 2 arrivées consécutives peut être de 40 minutes au maximum. Donc, nous avons échantillonné les dates de disponibilité d'une distribution uniforme telle que la différence entre les dates d'arrivée de 2 tâches consécutives est égale à X avec X~U [0; 40]. Pour un service de stérilisation qui ouvre à 8^h par exemple, le premier ensemble de DMR arrivera X minutes après 8^h. Ensuite, le deuxième ensemble de DMR arrivera encore X minutes suivant le premier ensemble de DMR, et ainsi de suite. Outre ce type d'arrivée, nous supposons qu'il peut y avoir un ramassage régulier des DMR

auprès des blocs opératoires. Dans ce cas, les DMR arrivent régulièrement au service de stérilisation. Nous considérons deux valeurs pour les inter-arrivées des tâches : 20 minutes et 40 minutes. Conformément au service de stérilisation étudié, afin d'avoir en moyenne 50 ensembles de DMR arrivant au service de stérilisation par jour, nous supposons que le nombre de tâches arrivant au service en même temps est échantillonné par distribution uniforme qui est U[0;2] pour le cas d'inter-arrivée de 20 minutes, et U[1;3] pour les inter-arrivées de 40 minutes (plus la différence entre deux ramassage est long, plus la chance d'avoir des ensemble de DMR arrivant en même temps au service de stérilisation augmente). Nous avons ainsi défini 3 types de dates d'arrivée. Nous allons encore une fois regrouper les instances en fonction de ces types d'arrivée. Nous nommons « 1er type » les instances dont les tâches arrivent individuellement, « 2ème type » les instances dont les tâches peuvent avoir 20 minutes d'inter-arrivées et enfin « 3ème type » l'inter-arrivée de 40 minutes. Pour les types d'arrivée 1 et 2, nous avons généré la date de début de pré désinfection des tâches suivant la formule suivante : $t_j = r'_j - U[5 ; 25]$ avec t_j la date de début de pré désinfection et r'_j la date d'arrivée dans le service de stérilisation pour la tâche *j*, et U une distribution uniforme. Pour le 3ème type, on a $t_j = r'_j - U[5 ; 40]$.

Supposons que les instances ainsi créées représentent les données prévues, *i.e.* les arrivées et les tailles prévues des tâches. Elles peuvent donc être utilisées par *TIH correctif* comme étant une première estimation de l'arrivée des ensembles de DMR. Par contre, nous avons besoin d'autres instances pour représenter l'arrivée réelle des ensembles de DMR. Rappelons que si une tâche est en retard/avance, nous parlerons de son anomalie d'arrivée. Alors qu'une tâche est en retard de 5 minutes, une autre tâche peut être en avance de 40 minutes. Donc en fait, il s'agit de différentes grandeurs/valeurs d'anomalie d'arrivée. Afin de créer les instances représentant l'arrivée réelle des ensembles de DMR et tester la performance de *TIH correctif*, nous considérons des « cas » différents en fonction de la durée de retard/avance des tâches. Ces cas représentent la valeur du retard/avance des tâches en fonction du type d'instance. Pour les instances du type 1, le premier cas concerne les petits retards/avances, inférieurs à 15 minutes. Si jamais une tâche est en retard/avance de plus de 15 minutes mais moins de 30 minutes, elle sera représentée dans le cas deux pour une anomalie d'arrivée de valeur moyenne. Le troisième cas correspond au cas où les tâches peuvent être en retard/avance de 60 minutes. Enfin, dans le quatrième cas, nous ne considérons que des retards qui peuvent aller jusqu'à 30 minutes. En résumé, nous avons considéré 4 cas pour les instances de type 1:

- cas 1 : une tâche peut être en avance/retard de 0 à 15 minutes,
- cas 2 : une tâche peut être en avance/retard de 0 à 30 minutes,
- cas 3 : une tâche peut être en avance/retard de 0 à 60 minutes,
- cas 4 : une tâche ne peut être qu'en retard de 0 à 30 minutes.

Pour le ramassage régulier, i.e. les instances de types 2 et 3, nous avons généré 3 cas pour le type 2 et 1 cas pour le type 3. Les 3 cas considérés pour les instances de type 2 sont :

- cas 1 : une tâche peut être en avance/retard de 0 ou 20 minutes,
- cas 2 : une tâche peut être en avance/retard de 0, 20 ou 40 minutes,
- cas 3 : une tâche peut être en avance/retard de 0, 20, 40 ou 60 minutes,

Le seul cas considéré pour les instances de type 3 est :

- cas 1 : une tâche peut être en avance/retard de 0 ou 40 minutes,

Donc, pour chaque instance prévue, nous avons créé 4 instances réelles pour une instance prévue de type 1, 3 instances réelles pour une instance prévue de type 2, 1 instance réelle pour une instance prévue de type 3 telles que l'arrivée réelle d'une tâche se situe dans un intervalle $[r'_j-Z ; r'_j+Z]$ avec r'_j la date d'arrivée prévue de la tâche j et Z la valeur d'avance/de retard égale à une valeur entre 0 et 15 pour le cas 1, 0 et 30 minutes pour le cas 2, 0 et 60 minutes pour le cas 3 quand il s'agit d'une instance de type 1. Encore pour le type 1 et pour le cas 4, l'arrivée réelle d'une tâche est situé dans un intervalle $[r'_j ; r'_j+Z]$ avec Z le retard entre 0 et 30 minutes. Quand il s'agit d'une instance de type 2, la valeur de Z peut être 0 ou 20 pour le cas 1, 0, 20 ou 40 pour le cas 2, 0, 20, 40 ou 60 pour le cas 3. Enfin, quand nous avons une instance prévue de type 3, la valeur de Z peut être 0 ou 40.

Les seules données modifiées dans les instances prévues sont les dates d'arrivées des tâches et les dates de début de pré désinfection. En ce qui concerne l'information sur la taille d'une tâche, nous n'avons fait aucune modification dans un premier temps. Donc, ces données sont communes entre les instances réelles et instances prévues (après avoir fait les premiers tests avec *TIH correctif* pour la durée de pré désinfection, nous testerons d'autres instances où la taille des tâches dans les instances prévues sera différente de celle des données dans les instances réelles).

Pour déterminer le nombre de tâches modifiées pour une instance prévue, nous avons posé une hypothèse qui impose qu'une instance prévue ait T tâches modifiées avec T égal à 5, 25, ou 50. Rappelons que les instances créées contiennent 50 tâches. Donc, pour une instance prévue, nous avons 3 configurations possibles :

- configuration 1 : 10% des tâches sont en avance/retard (T = 5)
- configuration 2 : 50% des tâches sont en avance/retard (T = 25)

- configuration 3 : 100% des tâches sont en avance/retard (T = 50).

Pour être plus clair, rappelons les spécificités des instances réelles. Nous avons 3 types d'instance dans les instances prévues, 1er, 2ème et 3ème type en fonction de l'arrivée des tâches. Une instance peut appartenir à la configuration 1, 2 ou 3. Ensuite, chaque tâche modifiée peut avoir un cas d'anomalie.

Pour chaque combinaison du type d'instance, cas d'anomalie et configuration, nous avons créé 30 instances. Le tableau 8.1 montre le nombre d'instances créées en fonction de la configuration et du cas d'anomalie. La même répartition est valide pour les instances de 2ème et 3ème type.

Tableau 8.1. Répartition réelle des instances de type 1

Type d'instance	1							
Configuration	1				2			3
Cas	1	2	3	4	1	2	3	1
Nombre d'instances créées	30	30	30	30	30	30	30	30

Nous pouvons maintenant aborder la performance de *TIH correctif* et de *HO* (heuristique online) sur ces instances pour le critère de minimisation du dépassement moyen de la durée idéale de pré désinfection (*DMP*).

8.3.2. Performance de *TIH correctif* et de *HO* pour le dépassement moyen de la durée idéale de pré désinfection (*DMP*)

Nous avons vu que l'heuristique *TIH* était relativement performante pour l'optimisation de la durée de pré désinfection. La question qui se pose est « est-ce que *TIH correctif* est capable de gérer les incertitudes (*i.e.* « anomalies ») concernant les dates d'arrivées des DMR au service de stérilisation. Nous allons donc regarder la qualité de *TIH correctif* et de *HO* pour les durées de pré désinfection.

Nous comparons les résultats de *TIH correctif* et de *HO* à la stratégie *FIFO online* avec différents seuils de remplissage. Lorsque nous avons expliqué *FIFO online*, nous avons précisé qu'un batch se fermait si et seulement si un ensemble de DMR n'entrait pas dans ce batch à cause d'un manque de place. Ce cas considère que le seuil de remplissage est égal à 100%. Cette fois-ci, nous avons considéré *FIFO online* avec différents seuils de remplissage égaux à 70%, 80%, 90% et 100%. C'est-à-dire que, par exemple, pour un seuil de 70%, un batch est fermé s'il est à 70% plein. Ensuite, nous choisissons la valeur du seuil de remplissage qui permet d'obtenir la meilleure durée de moyenne de pré désinfection (c'est-à-dire, celle la plus proche de 20 minutes). Notons que *FIFO*

online est aussi intégré dans un modèle de simulation et donc tous les tests numériques ont été faits en ARENA.

Dans les tableaux suivants, nous montrons les résultats de *TIH correctif*, de *HO* et de *FIFO online* pour le critère de *DMP*. La durée idéale de pré désinfection est fixée à 20 minutes dans toutes les expérimentations numériques. Les instances sont caractérisées en fonction de leurs « type d'instance + cas + configuration ». Nous avons regroupé les tableaux d'après les configurations. Chaque tableau montre la valeur minimale, maximale et moyenne du *DMP* pour les instances testées dans chaque combinaison de type d'instance + cas.

La première colonne représente le type d'instance, la deuxième indique le cas d'anomalie. Puis, pour *TIH correctif*, *HO* et *FIFO online*, la durée minimale et la durée maximale du dépassement de la durée idéale de pré désinfection sont indiquées. Ensuite dans la colonne suivante, la durée moyenne du *DMP* pour toutes les instances testées est présentée. La dernière colonne représente une borne inférieure naturelle. Pour l'obtention de la borne inférieure, nous avons supposé que les tâches sont traitées tout de suite, *i.e.* sans attente dans le stock de lavage. Pour ce but, la durée de pré désinfection d'une tâche, disons *j*, se calcule comme suit : $r_j - t_j$ avec r_j sa date d'arrivée et t_j la date de début de sa pré désinfection. Ainsi, le dépassement de la durée idéale de pré désinfection pour la tâche *j* s'obtient de la façon suivante : $max(r_j - t_j - 20 ; 0)$.

Les caractéristiques des résultats montrés dans le tableau 8.1 se ressemblent. *TIH correctif* et *HO* sont efficaces pour la minimisation du *DMP* d'un point de vue opérationnel, alors que *FIFO online* est peu performant (même s'il est testé avec différents seuils de remplissage). Rappelons qu'on pénalise les durées de pré désinfection supérieure à 20 minutes. Si un ensemble de DMR a une durée de pré désinfection comprise entre 15 et 20 minutes, alors la pénalisation est nulle. Il y a toujours des tâches dans chaque instance qui sont affectées à un cycle de lavage sans attente dans le stock de lavage. De plus, quand ces tâches n'ont pas beaucoup de différence entre leur début de pré désinfection et leur arrivée au service de stérilisation, elles obtiennent une durée de pré désinfection assez petite. Donc évidemment, nous obtenons dans chaque instance 0 minute de pénalisation minimale sur la durée de pré désinfection. Cette observation est valable pour *TIH correctif*, *HO* et *FIFO online*. L'unité de toutes les valeurs présentées dans les tableaux suivants est la minute.

Tableau 8.2. Comparaison du *DMP* obtenu avec *TIH correctif*, *HO* et *FIFO online* dans la configuration 1

		TIH correctif			*HO*			*FIFO online*			*Borne Inf.*
Type Inst.	Cas	min	max	moy.	min	max	moy.	min	max	moy.	
1	1	0	10	4.7	0	14	2.5	0	99	23.1	0.9
	2	0	23	5.6	0	25	3.4	0	119	19.2	1.5
	3	0	22	3.9	0	25	3	0	82	12	0.7
	4	0	35	5.9	0	44	2.7	0	97	11.4	0.9
2	1	0	32	5.4	0	31	3.5	0	66	18	1.6
	2	0	31	5.4	0	16	2.3	0	73	17	0.5
	3	0	47	6.2	0	20	3.1	0	73	20	1.3
3	1	0	33	7.3	0	30	5.3	0	69	14.2	3.1

Tableau 8.3. Comparaison du *DMP* obtenu avec *TIH correctif*, *HO* et *FIFO online* dans la configuration 2

		TIH correctif			*HO*			*FIFO online*			*Borne Inf.*
Type Inst.	Cas	min	max	moy.	min	max	moy.	min	max	moy.	
1	1	0	36	6.5	0	37	3.4	0	59	10.9	1.7
	2	0	35	5.2	0	29	2	0	69	10.9	0.8
	3	0	30	5.1	0	29	2.1	0	91	15.2	1
	4	0	32	5.2	0	29	3.1	0	91	14.5	1.8
2	1	0	43	5.2	0	28	4.4	0	77	18	1.8
	2	0	34	6.1	0	36	3.8	0	88	24	0.4
	3	0	44	4.2	0	28	4	0	66	19	1
3	1	0	47	9	0	29	8.1	0	78	13	3.8

Tableau 8.4. Comparaison du *DMP* obtenu avec *TIH correctif*, *HO* et *FIFO online* dans la configuration 3

		TIH correctif			*HO*			*FIFO online*			*Borne Inf.*
Type Inst.	Cas	min	max	moy.	min	max	moy.	min	max	moy.	
1	1	0	27	3.2	0	25	3.6	0	91	11.8	0.8
	2	0	31	4.8	0	25	3.1	0	96	18.8	1.4
	3	0	29	5.4	0	22	3.1	0	59	13.6	0.5
	4	0	30	5.2	0	29	4.4	0	61	11.2	0.7
2	1	0	32	7.1	0	20	3.8	0	65	11	1.4
	2	0	25	6.5	0	22	4.3	0	61	10	1.7
	3	0	42	7.4	0	26	3.6	0	76	11	1.5
3	1	0	44	9.2	0	31	8.9	0	74	14	4.1

Nous observons que *HO* est légèrement plus performant que *TIH correctif*. Cette observation a deux raisons principales. *HO* essaie de respecter la durée

idéale de pré désinfection pour que toutes les tâches aient une pénalisation nulle. Alors qu'avec *TIH correctif*, il suffit de recalculer un nouveau planning à chaque fois qu'il y a une tâche ayant une anomalie, *i.e.* retard ou avance. De cette façon, il est inévitable de s'éloigner du planning initial qui peut très bien avoir une bonne valeur pour *DMP*. Malgré tout, *TIH correctif* n'est pas trop influencé par des différentes valeurs d'anomalie et donc sa performance est à peu près la même pour toutes les instances.

Ces résultats donnent lieu à une question : si la différence entre les données prévues et les données réelles n'était pas que sur la date d'arrivée et le début de pré désinfection des tâches mais aussi sur la taille des tâches, comment se comporterait *TIH correctif* ? Par curiosité, nous avons donc créé 2 ensembles d'instances complètement différents au niveau du début de pré désinfection, de la taille, de la date d'arrivée dans le service. Le premier ensemble représente les données prévues, et le deuxième ensemble représente les données réelles. Ces instances contiennent 50 tâches chacune. Il n'y a aucune donnée commune entre ces deux ensembles d'instances. Mais à chaque arrivée d'une tâche dans le service de stérilisation, nous constatons que la tâche attendue n'est pas là et que nous avons une tâche complètement différente. Appelons ce type d'anomalie la *configuration 4*. Nous avons créé les instances pour les 3 types d'instance en fonction de l'organisation de l'arrivée des tâches dans les instances prévues. L'arrivée des tâches dans les instances réelles sera complètement différente des instances prévue tout en respectant la règle d'arrivée des tâches dans le instances de type 1, 2 et 3. Nous avons créé 30 instances prévues, et également 30 instances réelles. Les résultats pour le *DMP* sont donnés dans le tableau 8.5.

Tableau 8.5. Performance de *TIH correctif* pour la configuration 4

Type Inst.	min	max	moy.	Type Inst.	min	max	moy.	Type Inst.	min	max	moy.
1	0	55	7.6	2	0	54	6.5	3	0	60	10.2

Nous voyons grâce au tableau 8.5 que la performance de *TIH correctif* est légèrement moins bonne que dans les tests précédents. Mais, malgré le fait que les données prévues et réelles ne contiennent aucune information commune, nous pouvons dire que *TIH correctif* se comporte assez bien. Nous allons maintenant continuer avec le second critère d'optimisation qui est la minimisation du nombre de cycles de lavage.

8.3.3. Comparaison du nombre de batch créés

Nous avons vu que la minimisation du *DMP* était en rapport inverse avec la minimisation du nombre de cycles de lavage (*cf.* chapitre 6). Comme déjà précisé, le lancement d'un cycle de lavage est estimé coûter autour de 1 euro.

Même si ce deuxième critère n'est pas aussi important que la minimisation du *DMP*, voyons si nos méthodes de résolution sont, ou non, très loin de la stratégie *FIFO online*.

Nous allons examiner le nombre de batch créés en fonction du type d'instance. Pour chaque type d'instance, nous allons montrer sur un graphique les nombres minimaux et maximaux de batch, ainsi que le nombre moyen de batch donné par les différentes méthodes de résolution. Cette étude est faite avec *TIH correctif*, *HO* et *FIFO online*. Les graphiques suivants nous renseignent sur le nombre de batch créés par chacune de ces méthodes de résolution.

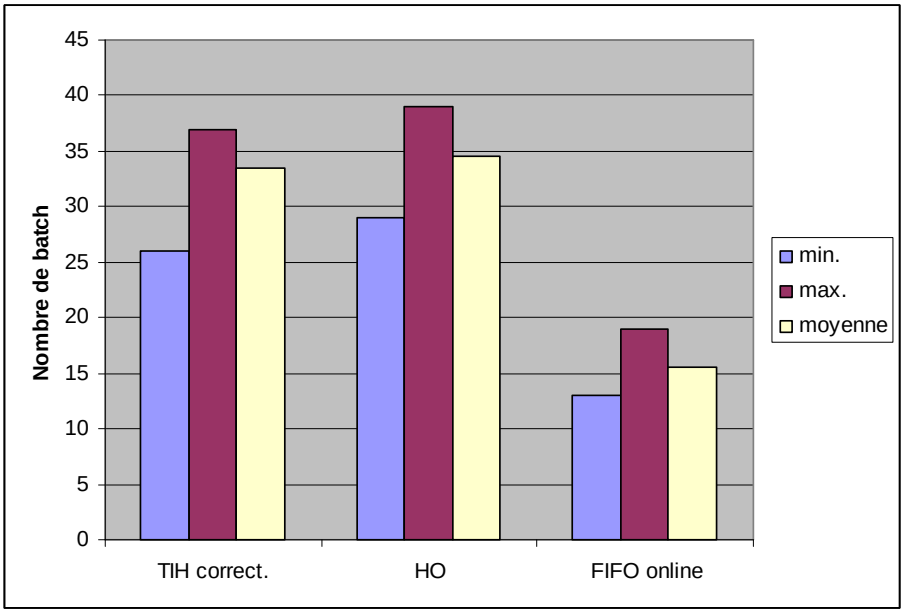

Figure 8.2. Nombre de batch donnés par *TIH correctif*, *HO* et *FIFO online* pour les instances du 1er type

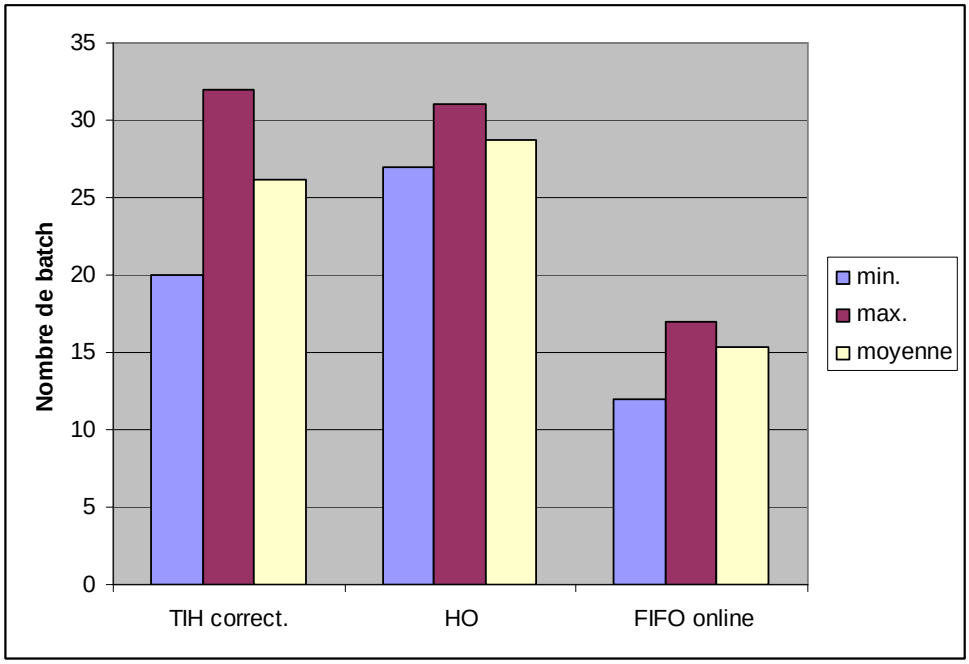

Figure 8.3. Nombre de batch donnés par *TIH correctif*, *HO* et *FIFO online* pour les instances du 2ème type

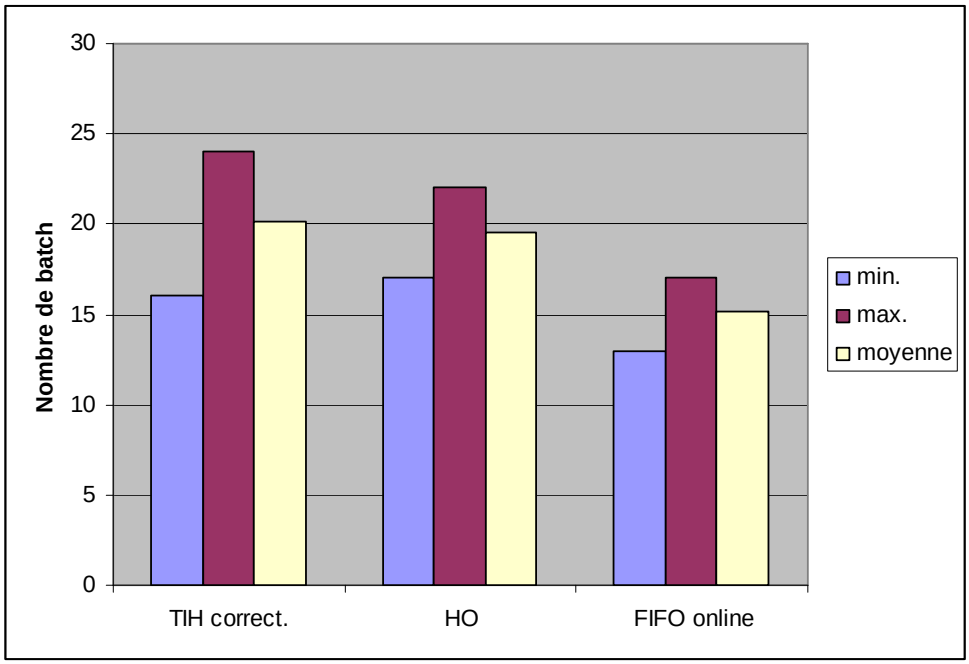

Figure 8.4. Nombre de batch donnés par *TIH correctif*, *HO* et *FIFO online* pour les instances du 3ème type

Similairement à la performance de *TIH*, *TIH correctif* se montre moins efficace que *FIFO online* quel que soit le seuil de remplissage. La même observation est valide pour *HO*. Mais avec l'accroissement du type d'instance, le nombre de batch créés par *TIH correctif* et *HO* s'approche de celui de *FIFO online*. La raison de cette observation est due au fait que quand les tâches arrivent en grande quantité, comme dans les instances de type 2 et 3, il est possible de mettre plus de tâches dans un cycle de lavage. Le nombre de batch créé est ainsi minimisé.

Dans la section suivante, nous allons compléter les modèles de simulation afin de voir l'impact de *TIH correctif* et *HO* sur l'ensemble du service de stérilisation.

8.4. Performance de *TIH correctif* et de *HO* sur l'ensemble du processus de stérilisation

Nous venons de voir que *TIH correctif* et *HO* sont bien performants pour l'optimisation de la durée idéale de pré désinfection. 20 minutes étant la durée idéale de pré désinfection, la valeur moyenne pour le dépassement de la durée idéale de pré désinfection pour les instances de type 1, 2 et 3 est de l'ordre de 5 à 10 minutes. Cela veut dire que la moyenne pour la durée de pré désinfection est autour de 25 à 30 minutes.

La seconde étape est de voir l'impact de nos méthodes d'optimisation sur l'ensemble du système où le système correspond au service de stérilisation. Pour ce faire, nous allons compléter le modèle de simulation que nous avons conçu dans la partie 8.1.2.

Rappelons que le service de stérilisation étudié contient les étapes « lavage, vérification + conditionnement, stérilisation ». Donc, nous ne nous occupons pas du transfert des ensembles de DMR vers les blocs opératoires. Les étapes qui restent à compléter sont « vérification + conditionnement, stérilisation ».

Expliquons brièvement le routage des ensembles de DMR après le lavage. Après avoir été lavés, les ensembles de DMR arrivent dans la zone de conditionnement. Cette zone est physiquement séparée de la zone de lavage. En effet, les laveurs possèdent deux portes. Les portes de chargement s'ouvrent dans la zone de lavage. Les portes de déchargement s'ouvrent à la zone de conditionnement. Les conditions d'hygiène nécessitent une telle séparation.

Rappelons que les ensembles de DMR sont composés de boîtes et de sachets (*cf* partie 1). Les DMR contenus dans les boîtes et les sachets appartenant à un ensemble de DMR sont lavés tous ensemble dans le même cycle de lavage. Les DMR qui sortent d'un laveur sont classés en deux groupes pour le conditionnement : les DMR qui seront mis dans des boîtes et les DMR qui seront mis dans des sachets. Les activités de conditionnement sont effectuées dans deux lieux, un premier lieu pour les DMR en boîtes et un deuxième lieu pour les DMR en sachets. Les DMR sont donc acheminés vers ces lieux de conditionnement après le lavage. Dans le service de stérilisation étudié, nous avons 4 postes de conditionnement pour les boîtes et 1 poste de conditionnement pour les sachets. La durée de conditionnement d'une boite varie entre 15 et 30 minutes. Le conditionnement d'un sachet dure entre 0.5 et 1 minute.

8.4.1. Finalisation du modèle de simulation

Dans la partie 8.1.2, nous avons présenté le modèle de simulation que nous avons construit pour *TIH correctif*. Puisque les modèles de simulation sur lesquels nous avons testé *HO* et *FIFO online* ressemblent à ce modèle, nous avons expliqué seulement le modèle de simulation de *TIH*. De plus, les parties des modèles situées après l'étape de lavage sont communes à tous les modèles.

Voyons maintenant la suite de l'acheminement des DMR après l'étape de lavage. Après sa sortie d'un laveur, les ensembles des DMR sont acheminés vers un stock où les DMR sont mis en attente pour le conditionnement. Quand un ensemble de DMR sort d'un laveur, il passe d'abord par le module VBA10 qui sert à lire la donnée qui indique le nombre de boîtes contenues dans cet ensemble de DMR. Ensuite, l'entité originale, représentant l'ensemble de DMR, est multipliée en fonction de ce nombre, et, les entités multipliées sont acheminées vers le module VBA 11. Les entités ainsi créées représentent les DMR qui seront conditionnés en boîtes. Par exemple, si un ensemble de DMR contient 3 boîtes, cet ensemble de DMR est détruit par un module « separate » et 3 entités sont créées et acheminées vers le module VBA 11. Chaque nouvelle entité est en fait un groupe de DMR qui sera mis dans une même boîte à l'étape de conditionnement. Dans le module VBA 11, nous affectons une taille à ces entités qui représente la taille des boîtes (en panier DIN) après leur conditionnement. Puis, ces entités sont mises dans un stock et les DMR attendent le conditionnement dans le stock appelé « stock DMR 1 ». Le module « VBA 14 » sert à envoyer les entités aux postes de conditionnement à chaque fois qu'un poste se libère.

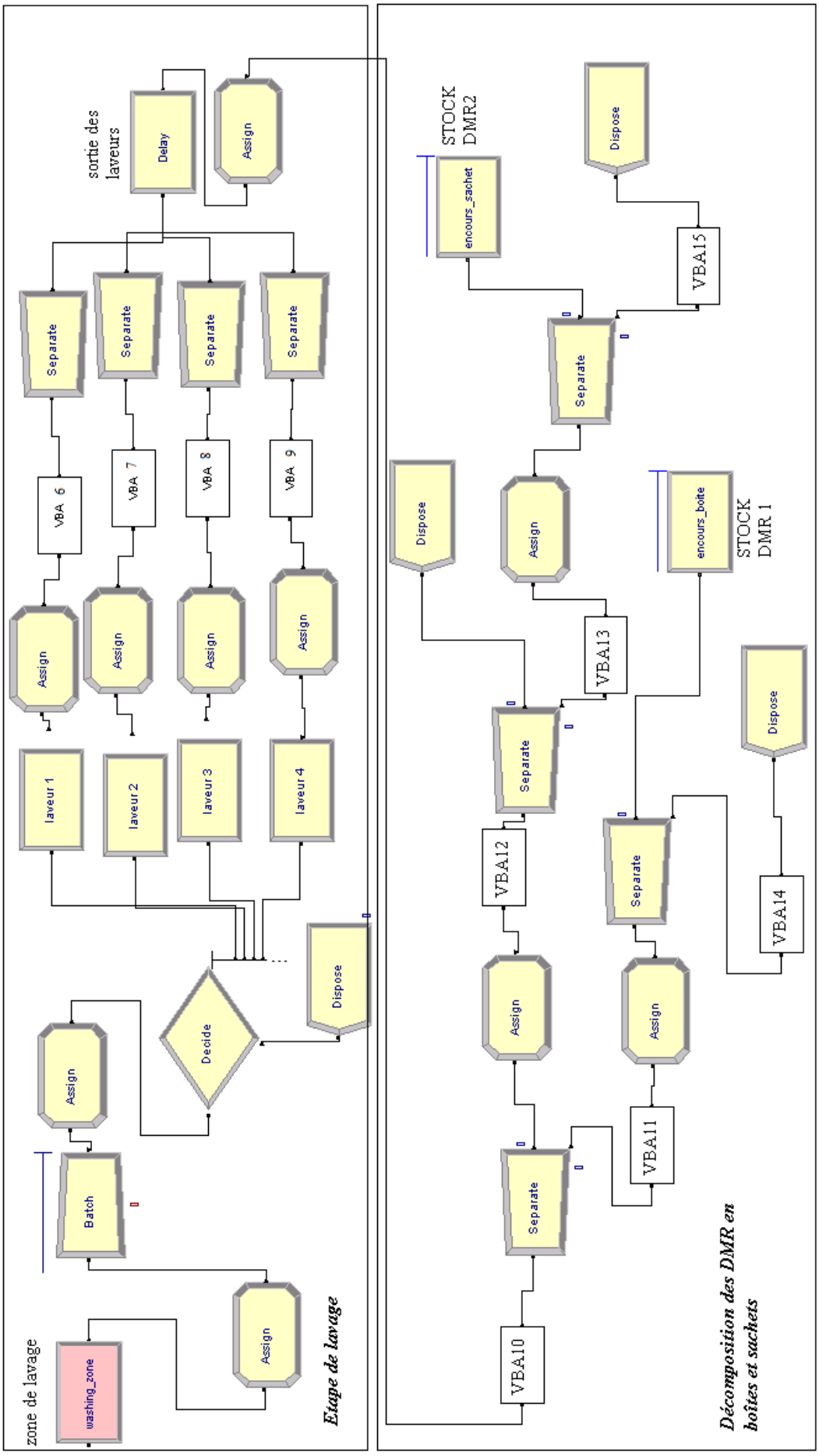

Figure 8.5. Etape de lavage et décomposition des ensembles de DMR après le lavage

Parallèlement à l'acheminement des DMR à travers les modules VBA 10, 11 et 14, les modules VBA 12, 13 et 15 font la même chose pour les DMR qui seront mis dans des sachets. Après sa décomposition suite au lavage, une entité représentant un ensemble de DMR passe par le module VBA12 qui est responsable de la lecture du nombre de sachets contenu par cet ensemble. Ensuite, de nouvelles entités sont créées via un module « separate » et les nouvelles entités représentant les futurs sachets passent par le module VBA13 pour définir leur taille. Enfin, ces entités sont mises dans le « stock DMR2 » représentant le stock de conditionnement des sachets.

L'étape suivant cette décomposition est le conditionnement et la stérilisation. La figure 8.6 montre ces étapes.

La suite du modèle de simulation est composée de 4 blocs (*cf.* figure 8.6). Comme nous l'avons déjà dit, le modèle contient 4 postes de conditionnement pour les boîtes et 1 poste de conditionnement pour les sachets. Les DMR qui attendent dans le « stock DMR 1 » sont transférés aux postes de conditionnement pour les boîtes dès qu'un poste est disponible. De la même manière, si le poste de conditionnement des sachets est disponible, une entité représentant les DMR qui vont constituer un sachet est envoyée à ce poste. Ainsi, le conditionnement dans des boîtes et sachets est fait. Quand une boîte, ou un sachet, est préparé, il est transféré devant les autoclaves, représentés par le bloc 3. Puis, en fonction de la stratégie de stérilisation, les boîtes et sachets sont chargés dans les autoclaves pour la stérilisation. L'étape de stérilisation est la dernière étape du processus de stérilisation. Après cette étape, les boîtes et sachets quittent le service de stérilisation et sont envoyés vers les zones de stockage avant leur réutilisation.

Nous devons donner une petite information concernant la composition des ensembles de DMR. Après avoir décomposé un ensemble de DMR suite au lavage, les boîtes et sachets de cet ensemble sont reconstitués grâce à l'étape de conditionnement comme nous l'avons mentionné ci-dessus. Après le processus de stérilisation, ces boîtes et sachets sont stockés avant leur réutilisation. Ensuite, en fonction du besoin d'un chirurgien pour une intervention, un nouvel ensemble de DMR est alors constitué au niveau des blocs opératoires avec les boîtes et sachets stockés. La composition des ensembles de DMR ne se fait donc pas au niveau du service de stérilisation.

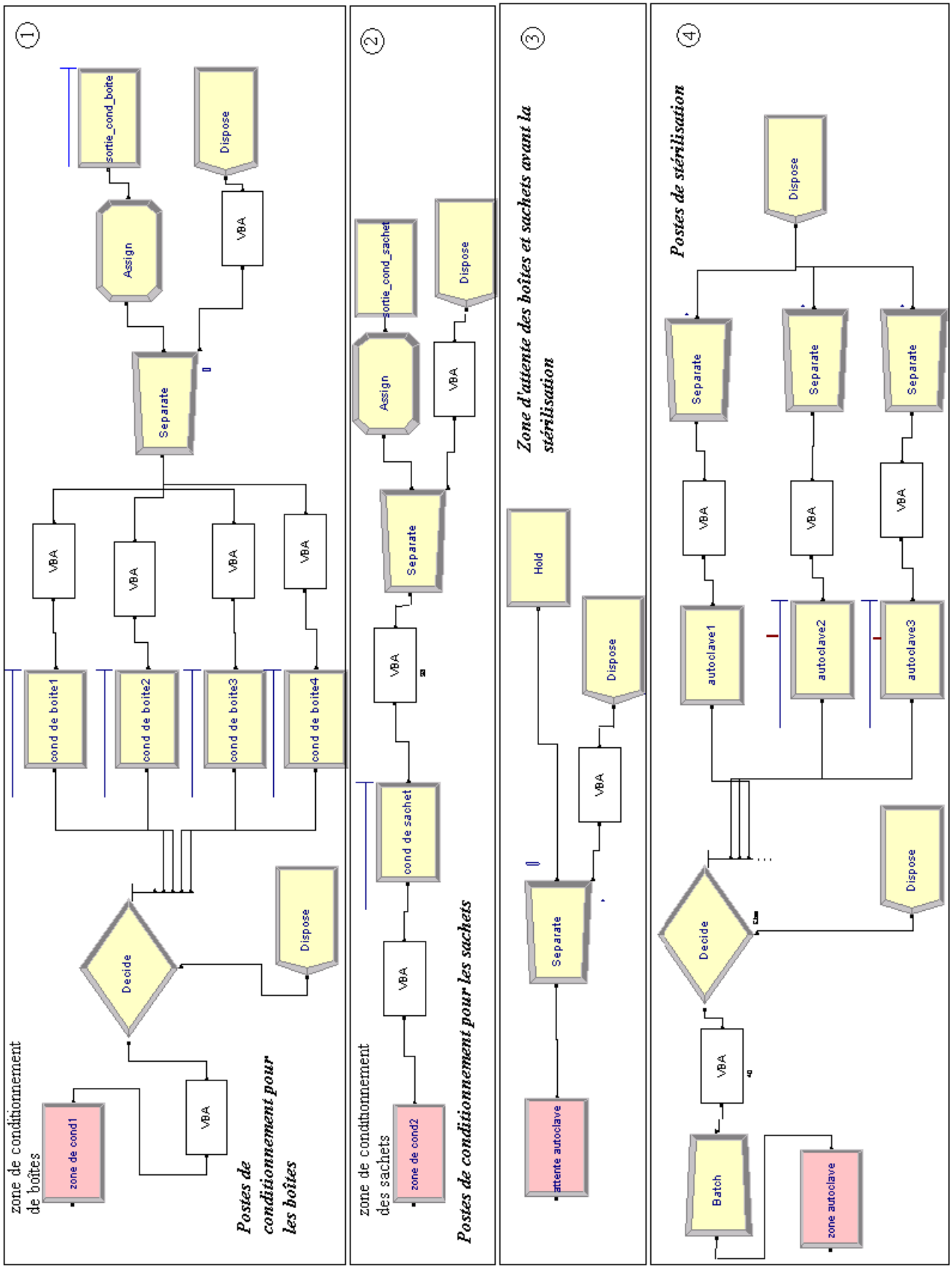

Figure 8.6. Etape de conditionnement et stérilisation

8.4.2. Données utilisées pour les expérimentations numériques

Nous avons vu dans la partie précédente que la performance de *TIH correctif* était satisfaisante indépendamment de la configuration et des cas que nous avons déterminés (*cf.* section 8.2.2.). Ce qui a le plus d'impact sur la durée de pré désinfection est le type d'arrivée des ensembles de DMR, *i.e.* les instances de 1er, 2ème et 3ème type. Donc, nous regroupons nos analyses en fonction du type des instances dans cette partie.

Même si nous avons utilisé les mêmes instances, il a fallu rajouter des données supplémentaires à ces instances. Puisque les ensembles de DMR sont composés de boîtes et de sachets, il faudrait déterminer le nombre de boîtes et de sachets contenus dans chaque ensemble de DMR ainsi que la taille de ces boîtes et de ces sachets. D'après nos observations, les ensembles de DMR peuvent contenir de 0 à 5 boîtes indépendamment de la grandeur (taille) de l'ensemble. Nous pouvons très bien avoir un ensemble de DMR dont la taille est égale à 1.2 DIN, (sachant que la capacité d'un laveur est 6 DIN) et qui contient 5 boîtes. Dans ce cas les boîtes sont très petites et contiennent de petits instruments. En ce qui concerne le nombre de sachets dans un ensemble de DMR, il varie entre 0 et 3. Les sachets sont généralement plus petits que les boîtes. Conformément à la réalité, nous supposons pour notre modèle de simulation que la taille des sachets est standard (*i.e*. ils ont tous la même taille).

Nous avons utilisé la distribution uniforme pour déterminer le nombre de boîtes et de sachets dans un ensemble de DMR. Pour chaque tâche d'une instance, sans considérer sa taille, nous avons déterminé le nombre de boîtes avec U[0,5] et le nombre de sachets avec U[0,3]. Puisque la taille des sachets est beaucoup plus petite que la taille des boîtes et qu'ils ont une taille unique, nous avons déterminé une taille fixe pour les sachets qui est égale à la plus petite taille d'un ensemble de DMR, *i.e.* à 1/6 panier DIN. Rappelons que la somme des tailles de boîtes et des sachets donne la taille d'un ensemble de DMR. Alors, concernant l'affection des tailles aux boîtes d'un ensemble de DMR, nous déterminons d'abord le nombre de sachets et la somme de leurs tailles, ensuite, nous soustrayons cette somme à la taille originale de l'ensemble de DMR. Le nombre qui reste, disons *stb* (pour *somme taille boîtes*), nous donne la somme de tailles des boîtes de cet ensemble. Enfin, nous affectons des tailles aléatoires à chaque boite pour que la somme de leur taille fasse exactement *stb*.

Le service de stérilisation travaille de 8 heures du matin jusqu'à 21 heures. D'après nos observations, les premiers ensembles de DMR arrivent vers 9 heures. Donc, nous faisons commencer la simulation à 9 heures du matin jusqu'à la fermeture, ce qui fait un horizon de 12 heures.

8.4.3. Expérimentations numériques

Nous allons maintenant lancer une série de simulations sur les instances déjà créées. Nous avons vu dans la section précédente que *TIH correctif* et *HO* sont des heuristiques assez bonnes pour optimiser la durée de pré désinfection. Cette fois-ci, notre but est de voir si nos méthodes de résolution sont capables d'améliorer le processus de stérilisation. Cette amélioration peut être examinée sous deux angles : 1) minimiser les attentes dans les stocks d'encours, 2) maximiser le nombre de boîtes et de sachets stérilisés dans une journée.

Commençons par les durées d'attente dans les stocks d'encours. Ces stocks d'encours sont plus précisément « le stock de lavage, le stock de conditionnement, et le stock de stérilisation ». Dans la figure 8.7 , nous montrons les valeurs maximales et moyennes de la durée d'attente des DMR pour *TIH correctif*, *HO* et *FIFO online* testés sur les instances de type 1. Les stocks d'encours représentés sur les figures suivantes sont notés comme suivant :

- stock lavage : pour les ensembles de DMR attendant le lavage,
- stock boîte : pour les DMR attendant le conditionnement des boîtes,
- stock sachet : pour les DMR attendant le conditionnement des sachets,
- stock stérilisation : pour les boîtes et les sachets attendant la stérilisation.

Nous voyons que l'attente des ensembles de DMR dans le stock de lavage est bien réduite par les heuristiques *TIH correctif* et *HO*. Bien que moindre, cette optimisation a aussi un impact sur le stock des DMR attendant le conditionnement des boîtes. *HO* est un peu plus performant que *TIH correctif* pour les valeurs moyennes de ce stock. En effet, nous comprenons aussi par la figure 8.7 que l'étape de conditionnement peut être vue comme un goulet, car, l'attente des DMR est plus élevée dans ce stock que dans les autres. Pour les deux autres stocks d'encours, *i.e.* encours de DMR attendant le conditionnement en sachet et la stérilisation, il n'y a pas de grande différence entre nos méthodes et *FIFO online*. Voyons maintenant le nombre de DMR maximal et moyen attendant dans ces stocks.

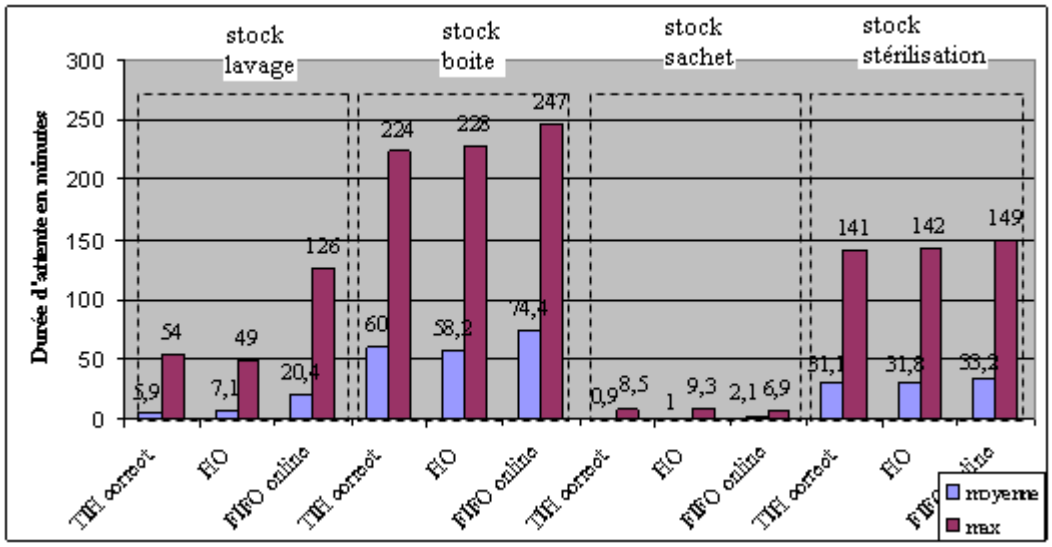

Figure 8.7. Durées d'attente dans les différents stocks pour les instances de 1er type

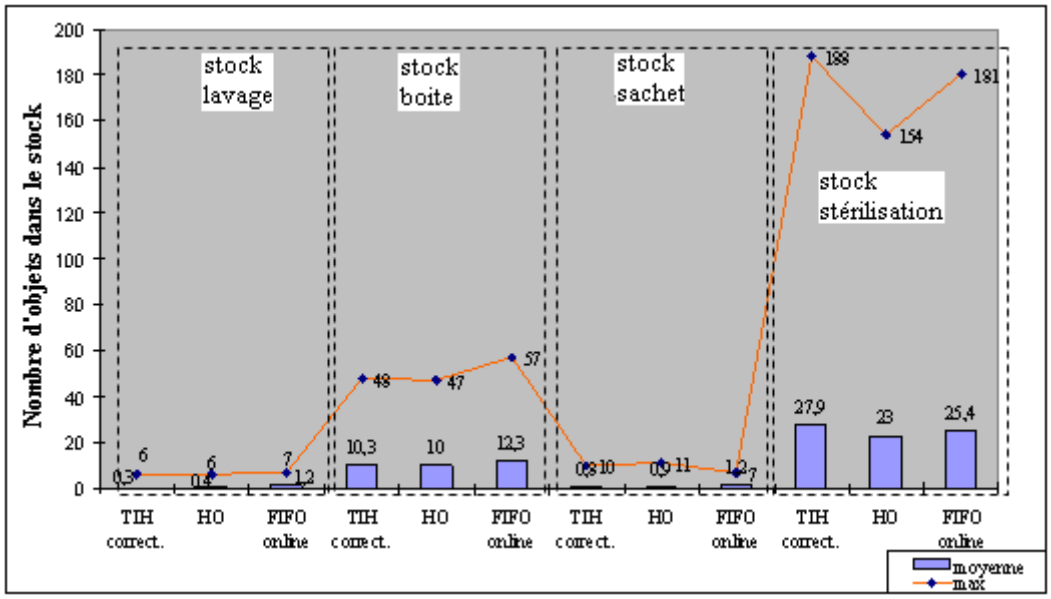

Figure 8.8. Nombre d'entités en attente dans les différents stocks pour les instances de type 1

La figure 8.8 montre qu'il n'y a pas beaucoup de différence entre nos méthodes d'optimisation et *FIFO online* pour le nombre d'entités attendant dans les différents stocks.

Les figures suivantes nous donnent des informations sur l'attente de DMR pour les instances de type 2 et 3 comme cela a été fait ci-dessus. Nous observons le même comportement sur ces instances aussi. C'est-à-dire que, comparé à *FIFO online*, *TIH correctif* et *HO* diminue considérablement l'attente de DMR dans les stocks d'encours de lavage et conditionnement des boîtes. Pour les deux autres

stocks, *i.e.* stock d'encours de conditionnement en sachet et stérilisation, l'attente de DMR est indépendante de la méthode appliquée.

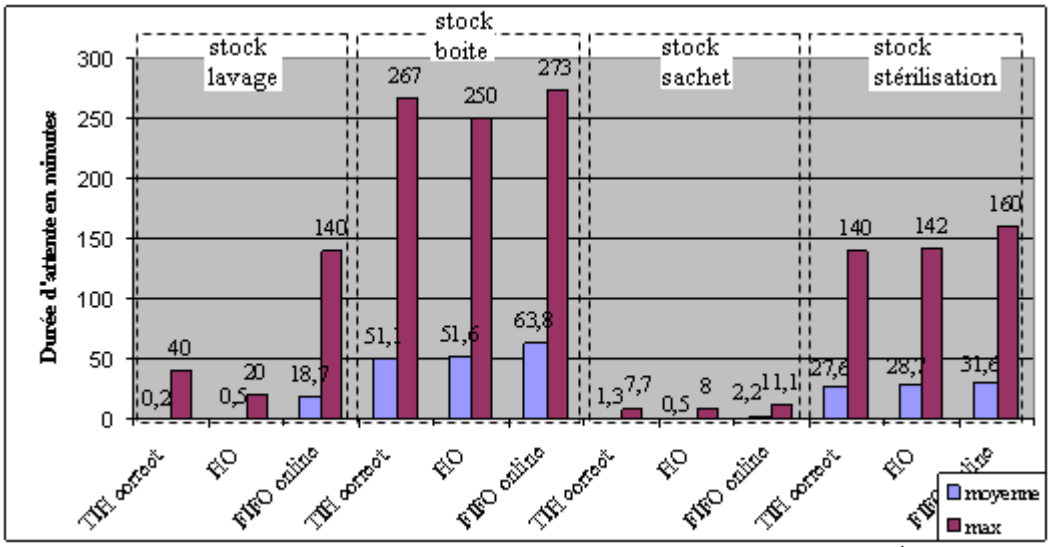

Figure 8.9. Durées d'attente dans les différents stocks pour les instances de 2ème type

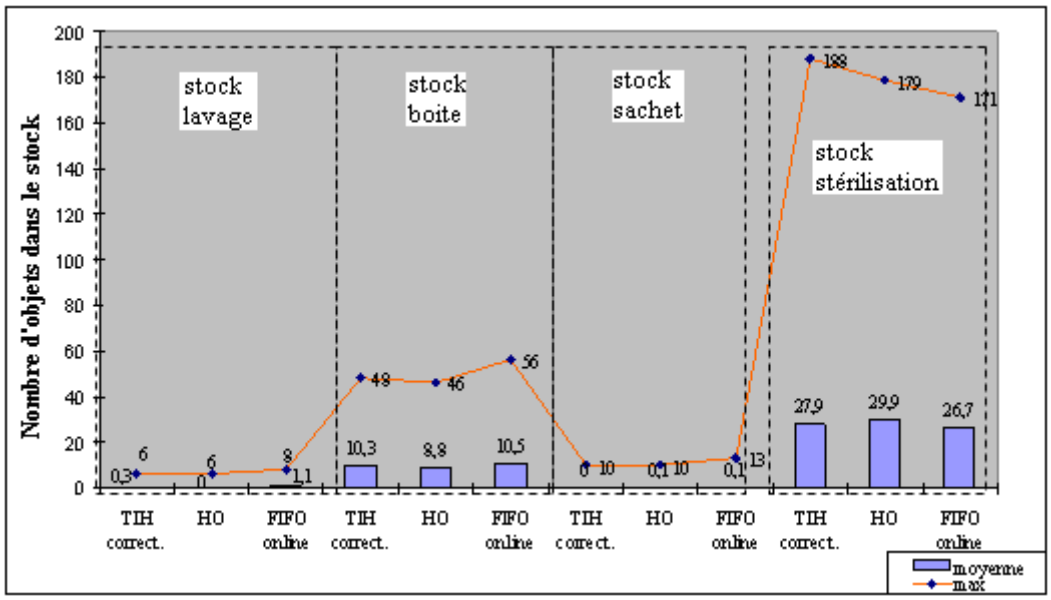

Figure 8.10. Nombre d'entités en attente dans les différents stocks pour les instances de type 2

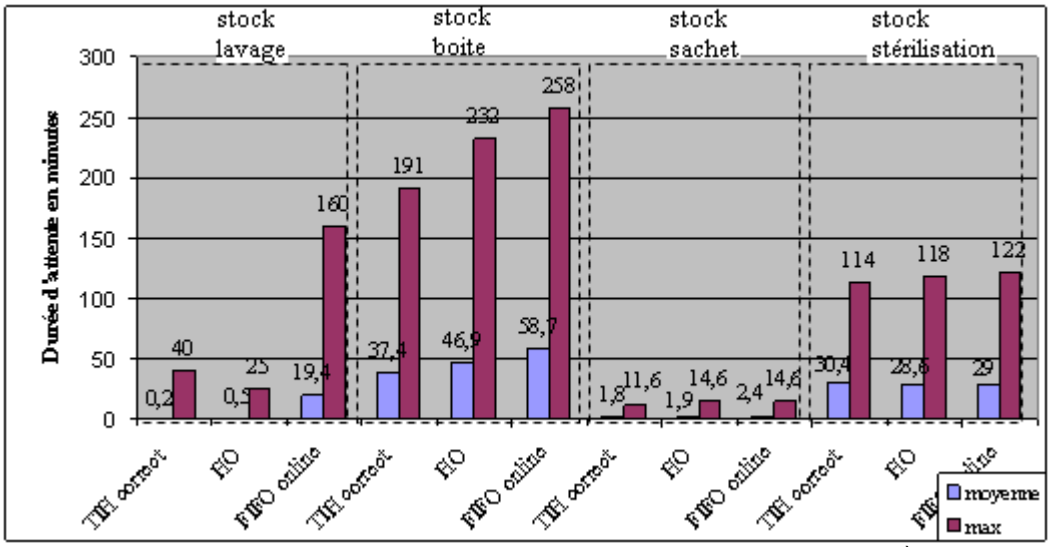

Figure 8.11. Durées d'attente dans les différents stocks pour les instances de 3ème type

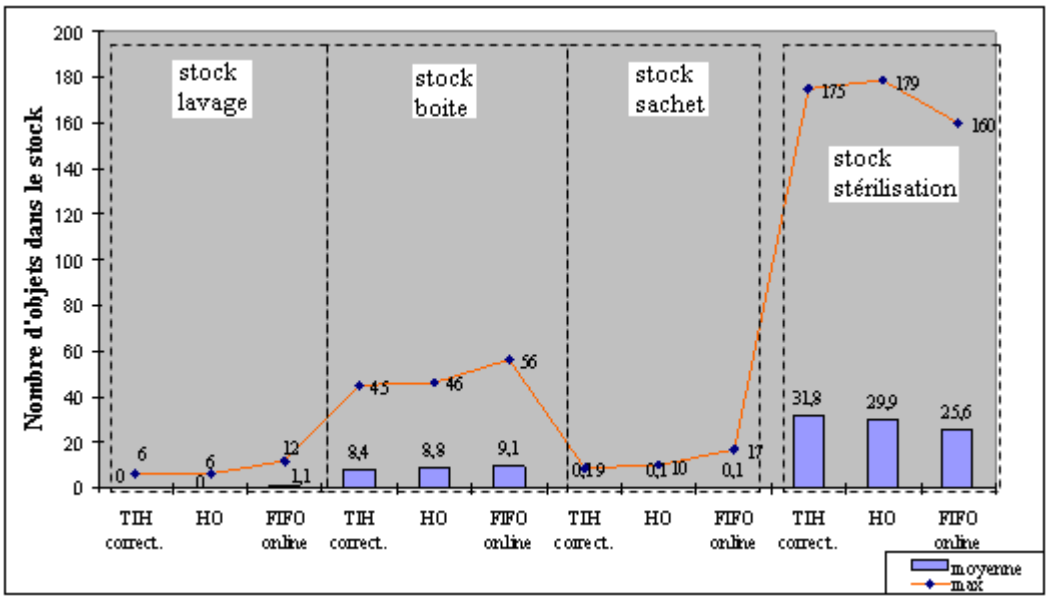

Figure 8.12. Nombre d'objets en attente dans les différents stocks pour les instances de type 3

Pour les instances de type 2 et 3, l'attente des tâches dans le stock de lavage est petite due à l'arrivée simultanée de plus d'une tâche. Alors, quand il y a 2 ou 3 tâches qui arrivent en même temps, nos heuristiques commencent leur exécution sans les faire attendre dans le stock de lavage. Le comportement de *FIFO online* est légèrement meilleur que nos méthodes pour le stock d'encours de stérilisation. La raison principale est due à l'arrivée en grande quantité des tâches depuis l'étape de lavage en passant par l'étape de conditionnement (plus l'attente augmente dans le stock de lavage, plus augmente le nombre de tâches arrivant en

même temps aux étapes suivantes). Dans ce cas, le seuil de remplissage des autoclaves est atteint plus rapidement et donc les tâches attendent moins longtemps.

En ce qui concerne le nombre de DMR stérilisés par jour, les performances de *TIH correctif*, *HO* et *FIFO online* sont à peu près égales. Puisque la seule chose qui change entre ces méthodes est l'utilisation des ressources de lavage, nous pouvons d'abord regarder le mouvement des ensembles de DMR dans cette étape. Nous avons donc observé le nombre d'ensembles de DMR lavés. *TIH correctif* et *HO* sont légèrement meilleurs que *FIFO online*. *FIFO online* est capable de traiter en moyenne 45 et 46 ensembles de DMR alors que les deux autres stratégies peuvent traiter entre 47 et 50 ensembles de DMR par jour. Puisqu'il ne s'agit pas d'une grande différence pour l'étape de lavage, les nombres de boîtes et de sachets stérilisés par ces trois méthodes sont aussi proches les uns des autres. Notons que le nombre moyen de boîtes et de sachets entrant dans le processus de stérilisation par jour est à peu près de 160 pour les boîtes et de 100 pour les sachets. Le nombre de boîtes et sachets stérilisés par jour via nos méthodes est entre 95 et 100 pour les boites, entre 35 et 40 pour les sachets, soit à peu près 130-140 boîtes et sachets. Les boîtes et sachets qui ne sont pas stérilisés attendent le lendemain. Avant que les nouveaux DMR arrivent à l'étape de stérilisation, les opérateurs stérilisent les DMR restant du jour précédent.

Il est possible d'augmenter le nombre de boîtes et de sachets stérilisés par jour. Nous avons observé, via les modèles de simulation, qu'il y avait deux points influant sur le nombre de boîtes et sachets stérilisés : 1) nombre de postes de conditionnement, 2) le seuil de remplissage des autoclaves. Dans un premier temps, supposons qu'il ne soit pas possible d'augmenter le nombre de postes de conditionnement. Intéressons nous au 2ème point. Le seuil de remplissage des autoclaves est de 80%. Nous pouvons lancer un cycle de stérilisation si un autoclave est au moins à 80% plein, ce qui correspond en moyenne à 15 et 20 boites et sachets en total dans notre cas. Ce seuil de remplissage est en effet très élevé et provoque des attentes longues.

Nous avons vu que l'obtention des bonnes durées de pré désinfection nous permettrait d'enlever le rinçage manuel et de transférer les opérateurs de ce poste de travail vers d'autres postes. En suivant cette idée, nous avons rajouté un 5ème poste de conditionnement des boîtes aux modèles de simulation. Notre but est de voir si le rajout de ce 5ème poste améliore les résultats de nos méthodes de résolution au niveau de la durée d'attente dans les stocks d'encours et au niveau du nombre de boîtes et de sachets stérilisés. Mais d'abord, parlons du pourcentage de remplissage des ressources de lavage, de conditionnement et de stérilisation.

Cela nous permettra de mieux comprendre le besoin d'un 5ème poste de conditionnement des boîtes.

Dans le cadre de notre analyse, nous parlons aussi du taux d'utilisation des ressources du processus de stérilisation. Nous avons remarqué sur les figures précédentes que l'attente des DMR était plus élevée dans le stock de conditionnement des boîtes que dans les autres stocks. Donc, nous pouvons estimer que le taux d'utilisation des postes de conditionnement est plus élevé que celui des laveurs ou des autoclaves. Dans les figures suivantes, nous montrons le taux d'utilisation des postes de lavage, de conditionnement et de stérilisation par les trois stratégies, *i.e. TIH correctif*, *HO* et *FIFO online*. Le taux d'utilisation d'un poste est le rapport entre sa durée d'utilisation et la durée d'ouverture du service.

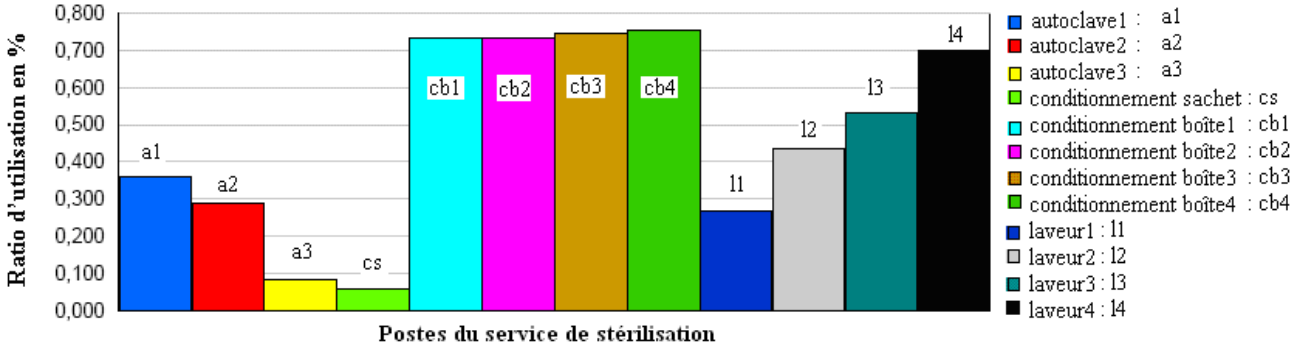

Figure 8.13. Ratio d'utilisation des postes par *TIH correctif*

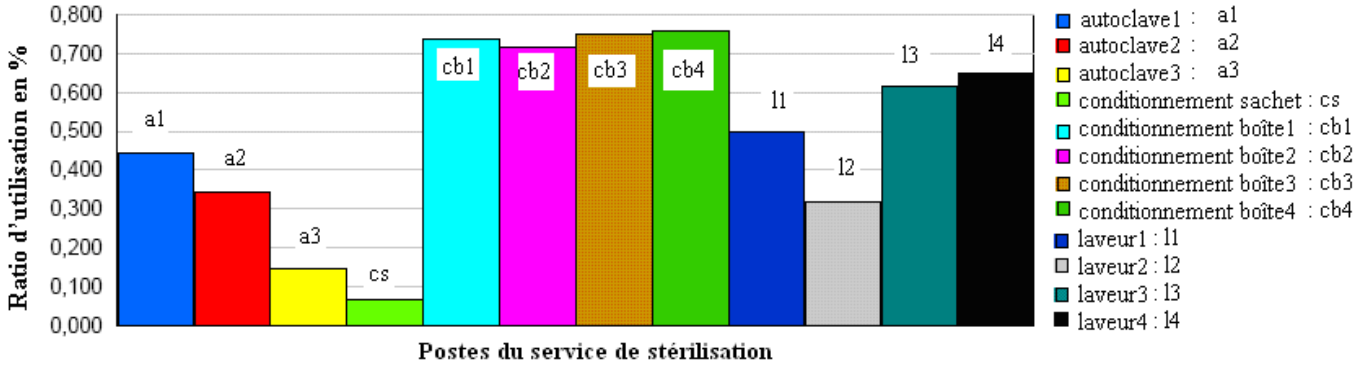

Figure 8.14. Ratio d'utilisation des postes par *HO*

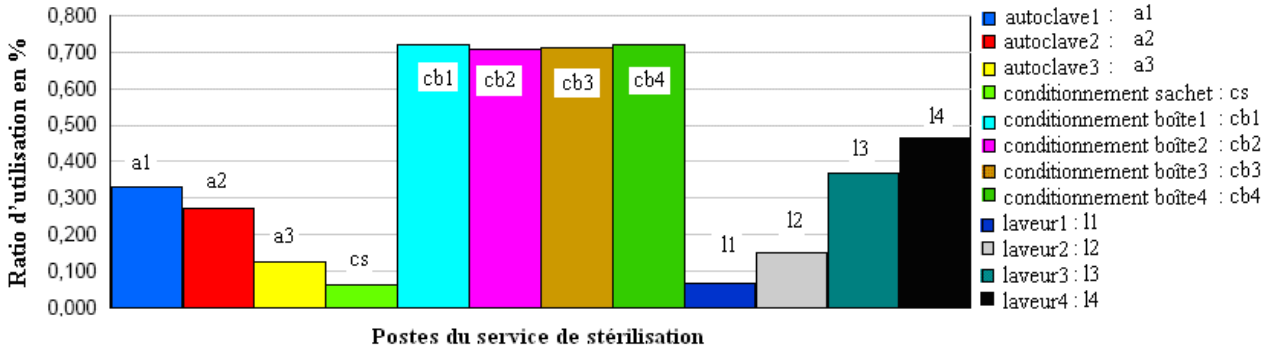

Figure 8.15. Ratio d'utilisation des postes par *FIFO online*

Le taux d'utilisation des autoclaves est conforme à nos observations. En effet, nous avons vu au cours des simulations que le 3ème autoclave était rarement utilisé. Nous voyons par ces figures que son taux d'utilisation est inférieur à 10%. Alors, nous pouvons en déduire que le troisième poste n'a pas beaucoup d'utilité.

En ce qui concerne l'utilisation des laveurs, nos méthodes sont capables de diminuer l'attente des tâches dans le stock de lavage en augmentant le taux d'utilisation des laveurs. Nous voyons grâce à ces simulations que le goulet d'étranglement de l'étape de lavage est du au fait que l'on fait attendre trop longtemps les tâches dans le stock de lavage. Ainsi, le taux d'utilisation des laveurs avec *FIFO online* est assez faible. La durée de conditionnement des sachets (1 minute maximum) rend le taux d'utilisation de ce poste assez petit. Par contre, les postes de conditionnement des boîtes nécessitent une intervention due au fait qu'ils sont trop chargés, *i.e.* taux d'utilisation élevé.

D'après les tests numériques, nos méthodes de résolution online et semi-online sont capables de garantir des bonnes durées de pré désinfection pour les DMR. Cela nous a permis de supprimer le rinçage manuel et ainsi nous n'avons pas représenté ce poste dans les modèles de simulation. Alors, pourquoi ne pas transférer le poste de rinçage manuel à un poste de conditionnement des boîtes en tant que 5ème poste ? Cela pourrait nous aider à accélérer le flux de DMR tout en allégant la charge des postes de conditionnement des boîtes. Nous reprenons donc les mêmes instances sur les mêmes modèles de simulation mais cette fois-ci, nous avons 5 postes de conditionnement des boîtes au lieu de 4.

Les simulations ont montré qu'il n'y a pas un changement considérable du nombre d'entités dans le stock de stérilisation mais la diminution de la durée d'attente dans le stock de conditionnement des boîtes est remarquable. Nous montrons sur les figures 8.16 et 8.17 la durée d'attente et le nombre d'entités dans le stock d'encours du conditionnement des boîtes pour les trois types d'instance.

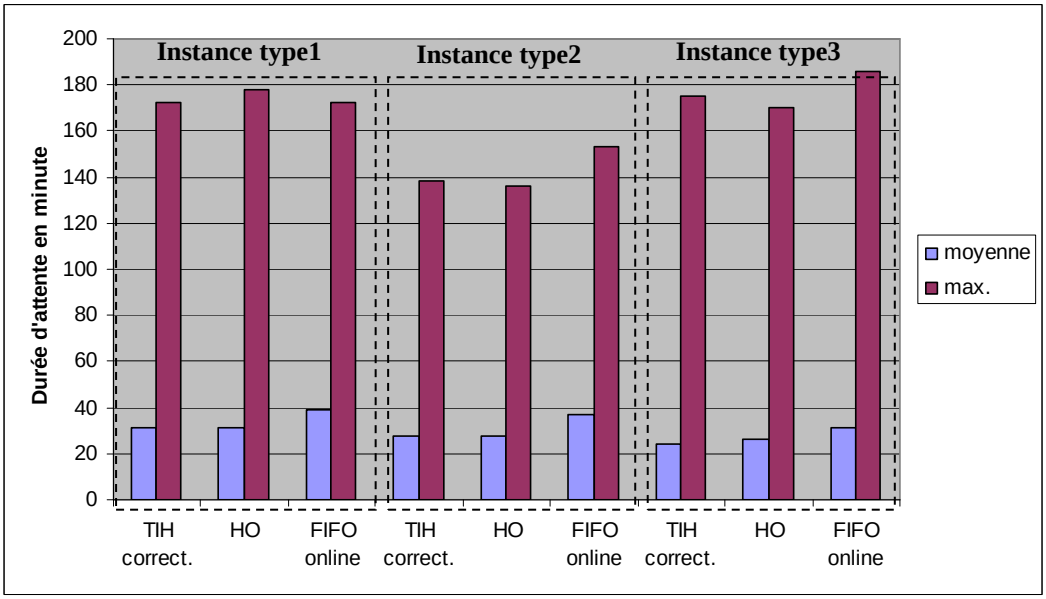

Figure 8.16. Durées d'attente dans le stock de conditionnement des boîtes pour les instances de type 1, 2 et 3

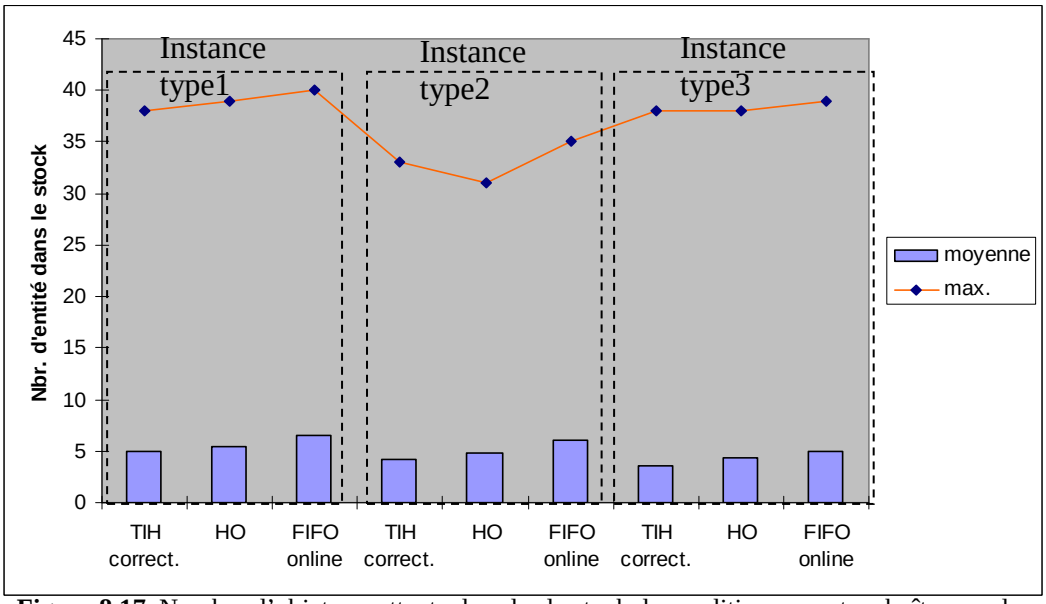

Figure 8.17. Nombre d'objets en attente dans les le stock de conditionnement en boîte pour les instances de type 1, 2 et 3

Notons qu'il y a une amélioration par rapport au cas avec 4 postes de conditionnement des boîtes pour toutes les stratégies et pour tous les types d'instance. L'attente de DMR dans le stock de conditionnement des boîtes

s'améliore de 17 minutes en moyenne. L'impact de l'ajout du 5ème poste de conditionnement en boîte est donc extremement fort.

Alors, est-que ce changement a un impact sur la quantité de DMR stérilisé aussi ? Pour ce but, nous avons compté le nombre de boîtes et de sachets stérilisés par jour. Nous montrons dans le tableau 8.6 le nombre moyen de boîtes et sachets stérilisés par les trois stratégies en fonction des différents types d'instance.

Tableau 8.6. Nombre moyen de boîtes et de sachets stérilisé par jour

	Instance type 1	Instance type 2	Instance type 3
TIH correct.	176	178	162
HO	192	186	174
FIFO online	173	169	153

Nous avions dit qu'en présence de 4 postes de conditionnement des boîtes, le nombre de boîtes et de sachets stérilisés par jour variait entre 130-140. Nous voyons grâce au tableau 8.6 que l'ajout du 5ème poste de conditionnement des boîtes augmente énormément le nombre de boîtes et sachets stérilisés dans une journée. Cette augmentation est valable pour toutes les stratégies. Mais la performance de *HO* est beaucoup plus remarquable par rapport aux autres. De plus, nous comprenons grâce à ce tableau que l'impact de nos méthodes d'optimisation sur le processus de stérilisation n'est pas seulement efficace pour l'optimisation de la durée de pré désinfection, mais aussi, en permettant l'ajout d'un 5ème poste de conditionnement de boîtes, pour l'attente des DMR dans des stocks d'encours et pour le nombre de DMR stérilisés.

8.5. Conclusion

Notre but dans ce chapitre était de proposer des algorithmes semi-online et online pour optimiser la durée de pré désinfection des DMR. Pour ce but, nous avons présenté *TIH correctif* comme algorithme semi-online et *HO* comme algorithme online. La spécificité de *TIH correctif* est qu'il suppose de connaître à l'avance toutes les données d'une instance, *i.e.* la taille, la date d'arrivée et le début de pré désinfection des ensembles de DMR. Si un ensemble de DMR est en retard/avance, les données concernant cet ensemble sont mises à jour et l'heuristique *TIH* est relancée pour déterminer les nouveaux instants de lancement des batch ainsi que les ensembles mis dans ces batch. Les tests numériques ont montré que *TIH correctif* et *HO* sont capables de fournir de bonnes durées de pré désinfection, *i.e.* autour de 25 minutes, alors que *FIFO online* est peu performant pour ce critère même avec différents seuils de remplissage.

Nous avons aussi testé l'impact de ces méthodes via des modèles de simulation construits en ARENA 11.0. *TIH correctif* et *HO* peuvent diminuer l'attente des DMR dans les stocks de lavage et de conditionnement des boîtes, mais surtout dans le stock de lavage. De cette façon, plus de DMR sont envoyés aux postes de conditionnement. Mais nous avons vu par les simulations que l'étape de conditionnement des boîtes était aussi un goulet d'étranglement et donc, la minimisation de l'attente avant le lavage ne pouvait pas avoir beaucoup d'impact sur le nombre de boîtes et de sachets stérilisés par jour. La deuxième raison qui limite le nombre de boîtes et de sachets stérilisés par jour est la stratégie de chargement des autoclaves. Le seuil de remplissage pour ces machines est de 80%. Nous avons observé que ce niveau de seuil provoque de longues attentes dans le stock de stérilisation et empêche de stériliser plus de DMR. Pour des raisons sanitaires, il n'est pas possible de diminuer le seuil de 80%. Moins chargé, les autoclaves ne fonctionnent pas avec la pression adéquate et la vapeur d'eau ne circule pas bien entre les matériels et donc, les microorganismes peuvent rester vivants même après la stérilisation.

En ce qui concerne le goulet d'étranglement de l'étape de conditionnement des boîtes, la seule façon pour y remédier est d'ajouter un nouveau poste de travail. Par contre, cela représente une charge financière supplémentaire pour le service de stérilisation. Nous avons des heuristiques efficaces pour optimiser les durées de pré désinfection. Dans ce cas, le rinçage manuel avant le lavage devient inutile. En plus, chaque cycle de lavage peut commencer par un rinçage automatique. Donc, il serait possible d'enlever le rinçage manuel du système. Et, peut être, l'opérateur de ce poste pourrait être transféré vers un autre poste, par exemple, vers un $5^{ème}$ poste de conditionnement. Nous avons donc rajouté un $5^{ème}$ poste de conditionnement des boîtes aux modèles de simulation. Les tests ont montré que l'attente des DMR dans le stock de conditionnement des boîtes est maintenant autour de 30 minutes alors qu'elle était de 40-45 minutes. Mais le résultat le plus remarquable est le nombre de DMR stérilisés dans une journée. Dans la nouvelle configuration, nous sommes capables de stériliser entre 180 et 190 boîtes et sachets par jour avec *TIH correctif et HO*, et entre 160 et 170 boîtes et sachets avec *FIFO online*. Ces nombres étaient autour de 130-140 avant le rajout du $5^{ème}$ poste de conditionnement.

Conclusion de la partie 3

Nous avons étudié dans cette partie des configurations semi-online et online pour notre problème de chargement des laveurs. Pour ces études, nous avons repris la fonction objectif du chapitre 6 qui a pour but d'optimiser la durée de pré désinfection et de supprimer le rinçage manuel. Nous avons vu, dans le chapitre 6, que l'heuristique *TIH* est très puissante pour obtenir des bonnes durées de pré désinfection. Pour le cas semi-online de notre problème, nous avons modifié cette heuristique et l'avons insérée dans un modèle de simulation conçu en ARENA. Cette nouvelle heuristique que l'on appelle *TIH correctif* consiste à reformer un plan d'ordonnancement à chaque fois qu'une tâche est en retard ou en avance. Pour le cas purement online, nous avons proposé une autre heuristique. Les expérimentations numériques montrent que ces deux méthodes sont capables d'obtenir des bonnes durées de pré désinfection.

La deuxième étape de l'étude online (également semi-online) consistait à tester l'impact de nos méthodes d'optimisation online et semi-online sur l'ensemble du processus de stérilisation. Grâce aux premières simulations, nous avons découvert que l'étape de conditionnement des boîtes est aussi un goulet d'étranglement. Il n'était donc pas possible d'augmenter la performance du service de stérilisation. Mais l'optimisation de la durée de pré désinfection nous a permis de transférer l'opérateur du poste de rinçage manuel vers les postes de conditionnement. En rajoutant, donc, un poste supplémentaire pour le conditionnement des boîtes, nous avons testé encore une fois les mêmes instances. Les résultats numériques montrent que, avec nos méthodes, il est possible de diminuer l'attente des DMR dans différents stock en-cours et d'augmenter le nombre de DMR stérilisés par jour.

~ PARTIE 4 ~

Modèle de couplage entre l'optimisation combinatoire et la simulation : un cas d'étude

Nous effectuons dans cette partie une étude de cas dont le but est de minimiser le nombre de cycles de lavage lancés dans le cas où les ensembles de DMR sont simultanément disponibles. Pour ce faire, nous modélisons l'étape de lavage comme un problème de bin packing et nous utilisons une méthode existant dans la littérature. Afin de tester l'impact de cette optimisation sur l'ensemble du service de stérilisation, comme cela a été fait dans la partie 3, nous construisons un modèle de simulation en ARENA pour lequel les cycles optimaux pour le lavage sont fournis comme une entrée du modèle.

Chapitre 9 : Minimisation du nombre de cycles de lavage pour le cas où tous les DMR sont simultanément disponibles

9.1. Motivations

Nous avons parlé jusqu'à maintenant des services de stérilisation où les ensembles de DMR arrivent dans le service à différents instants de la journée. Ce type d'organisation est valide pour la plupart des hôpitaux ayant un service de stérilisation interne, *i.e.* un service de stérilisation qui appartient à cet hôpital. Néanmoins, il y a des hôpitaux qui ne possèdent pas de service de stérilisation. Ces hôpitaux envoient leurs DMR à un service de stérilisation externe. Là, les DMR sont stérilisés avant d'être renvoyés à l'hôpital.

Un service de stérilisation externe possède les étapes de lavage, vérification, conditionnement et stérilisation. La pré désinfection et le rinçage des DMR se font au niveau des blocs opératoires avant leur arrivée dans le service de stérilisation. La fonction objectif (*i.e.* l'optimisation des durées de pré désinfection) étudiée auparavant n'est donc plus valide ici. Mais, nous pouvons très bien nous intéresser à la minimisation de la durée d'attente des DMR dans l'étape de lavage.

Quand un hôpital envoie ses DMR à un service externe, les DMR partent de l'hôpital tous ensembles et arrivent donc en même temps dans le service. Ils sont simultanément présents devant les laveurs. Ils sont donc prêts pour le lavage en même temps. Notre but dans ce chapitre est de minimiser le nombre de cycles de lavage et voir l'impact de cette optimisation via un modèle de simulation sur l'ensemble du service de stérilisation.

9.2. Couplage entre l'optimisation et la simulation

Quand nous parlons du couplage entre l'optimisation et la simulation pour un problème d'optimisation, il peut s'agir de deux types d'application [43]: 1- un problème d'optimisation coupé en plusieurs morceaux pour lesquels une partie est prise en charge par les méthodes d'optimisation et l'autre partie par les méthodes de simulation, 2- application d'une méthode d'optimisation à une étape spécifique

d'un problème plus grand. Le premier cas est considéré pour des problèmes pour lesquels il est possible de proposer des modèles mathématiques ou bien des modèles de simulation. L'inconvénient des modèles mathématiques est lié au fait qu'il ne peut pas être tout à fait possible de représenter le problème avec toutes les contraintes, incertitudes, etc. [43]. Par contre, l'utilisation des modèles de simulation est un bon moyen pour représenter le comportement physique d'un système de production, transport, etc. où le problème d'optimisation trouve son origine. Mais, l'utilisation des modèles de simulation peut être insuffisante pour l'optimisation du problème [105]. Pour de tels cas, une façon de coupler la simulation avec l'optimisation est de proposer des modèles analytiques pour des sous problèmes du problème initial en les liant avec d'autres parties du problème de départ où les entrées, sorties, évaluations, etc. sont gérées par des outils de modèles à événements discrets, plus précisément par un outil de simulation [96]. Ainsi, un couplage entre l'optimisation et la simulation est obtenu.

Nous nous intéressons plutôt au point deux cité ci-dessus. L'étude effectuée dans cette partie est similaire à ce qui a été fait dans la partie 3. Le couplage entre la simulation et l'optimisation que nous proposons consiste donc en un seul, aspect qui est de tester l'impact de la méthode d'optimisation proposée sur l'ensemble du service de stérilisation.

Nous continuons par la suite la méthode de résolution. Après avoir présenté cette méthode, nous allons la comparer à l'heuristique *FFD* pour le nombre de batch créés et pour son impact sur le processus de stérilisation via un modèle de simulation.

9.3. Modélisation de l'étape de lavage comme un problème de bin packing

Nous modélisons cette fois-ci l'étape de lavage comme un problème de bin packing. Nous avons fait la transformation suivante pour l'obtenir :

- les cycles de lavage sont des « bins » à remplir avec des ensembles de DMR,
- les ensembles de DMR sont des « objets » à mettre dans les bins,
- tous les objets sont disponibles en même temps.
- les objets ont des tailles différentes.

Expliquons brièvement le problème de bin packing. Dans un problème de bin packing, nous avons des boîtes appelées « bin » qui ont toutes la même taille et des objets de tailles différentes. Classiquement, dans la littérature, les tailles sont normalisées de la façon suivante : la taille des bins est unitaire et les objets

sont de tailles inférieures ou égales à 1. Les objets sont rangés dans les bins. Le but est de trouver le rangement le plus économique possible, *i.e.* de minimiser le nombre de bins utilisés.

Le bin packing est l'un des problèmes le plus connus dans le domaine de l'optimisation combinatoire. Etant un problème NP-dur au sens fort, le bin packing est largement étudié par la communauté de la recherche opérationnelle. Il y a un grand nombre de travaux dans la littérature. Parmi les plus importants, nous pouvons citer Johnson *et al.* [57] qui proposent les algorithmes classiques de bin packing dont nous avons largement discuté dans la partie 2. Deux autres résultats remarquables sont les travaux de Fernandez de la Vega et Lueker [41] et de Karmarkar et Karp [60]. Fernandez de la Vega et Lueker développent un APTAS[1] tel que la sortie de cet algorithme ne dépasse pas $(1+\varepsilon)nb+g(\varepsilon)$ bins avec ε un nombre positif, $g(\varepsilon)$ une fonction de ε et nb la solution optimale. La durée d'exécution est une fonction linéaire de nb mais elle dépend exponentiellement de ε. Karmarkar et Karp [60] donnent un meilleur résultat qui permet d'avoir un AFPTAS[2]. Ils donnent un algorithme qui permet la garantie asymptotique suivante : $A \leq nb+O(log^2 nb)$ avec A la valeur trouvée par l'algorithme et nb la solution optimale. Outre les algorithmes approchés, Gilmore et Gomory [50] proposent un modèle linéaire pour le problème « cutting stock » et développent la méthode de génération de colonnes pour la résolution exacte de ce problème. En effet, la différence entre les problèmes de cutting stock et de bin packing se révèle dans les instances traitées. Dans les deux problèmes, nous avons des objets de tailles différentes. Dans les instances du problème de bin packing par contre, le nombre de tailles d'objet distinctes est beaucoup plus élevé que dans les instances du problème de cutting stock. Donc, quand des objets de même taille apparaissent de nombreuses fois dans le problème, le terme de cutting stock est préféré.

Comme expliqué dans les chapitres précédents, le nombre de tailles différentes dans notre problème n'est pas très élevé. Les tailles des ensembles de DMR peuvent être des multiples de $1/36^{ème}$ de la capacité des laveurs. Alors, nous pouvons avoir 36 tailles différentes au maximum. Nous pouvons donc nous servir du modèle linéaire de Gilmore et Gomory pour la résolution exacte de notre problème de bin packing.

[1] Asymptotic polynomial time approximation scheme
[2] Asymptotic fully polynomial time approximation scheme

9.4. Modèle linéaire en nombres entiers

Notons par $I = \{s_1, s_2,\ldots, s_n\}$ un ensemble d'objets pour lesquels la taille, s_j, de l'objet j, $j \in [1, n]$, est inférieure ou égale à 1, Le but du problème de bin packing est de répartirr ces objets dans un nombre minimum de sous-ensemble I_1, $I_2, \ldots, I_K$, tel que $\sum_{s_j \in I_k} s_j \leq 1$ pour tous les sous ensembles I_k avec $k \in [1, K]$. Le modèle linéaire de Gilmore et Gomory nécessite de présenter l'ensemble I sous une forme de multiplicité $a \in [0, 1]^d$ et $b \in \mathbb{Z}_+^d$, où d est le nombre de tailles différentes dans I et b_i est la multiplicité de la taille a_i ($i \in \{1, 2, \ldots, d\}$). Plus clairement, b_i est le nombre de fois où la taille a_i est présente dans l'ensemble I. Dans ce cas, I peut être représenté par $I = <d, a, b>$.

Considérons maintenant un ensemble (ou encore une instance) $I = <d, a, b>$. Un placement faisable des objets de I dans un bin peut être décrit par un vecteur $v \in \mathbb{Z}_+^d$ tel que $a^T v_i \leq 1$. Ici, v_i représente le nombre de fois que l'objet de la taille a_i est mis dans le bin. Un tel vecteur v est appelé un « pattern ». Si nous pouvons énumérer tous les patterns $v_1, v_2,\ldots, v_P$, alors le problème peut être modélisé par le modèle linéaire en nombres entiers suivant :

Indices
p : $1,\ldots,P$ pour les patterns
i : $1,\ldots, d$ pour les tailles

Paramètres
b_i : d étant le nombre de tailles distinctes dans l'instance et a_i une taille, b_i est le nombre d'objets dont la taille est égale à a_i

Variables
q_p : nombre de fois que le pattern p est utilisé dans la solution
v_{ip} : nombre de fois qu'un objet de taille a_i se trouve dans le pattern p

Modèle linéaire de Gilmore et Gomory

$$\textit{Minimiser} \sum_{p=1}^{P} q_p$$

tq.

$$\sum_{p=1}^{P} q_p v_{ip} = b_i \qquad \text{pour } i = 1,2,\ldots,d \tag{1}$$

$$q_p \text{ entier pour } p = 1,2,\ldots,P$$

Ici, nous avons un nombre de pattern P après les avoir tous énumérés.

Expliquons le modèle sur un petit exemple. Soit 10 objets ayant les tailles : ¼, ¼, ¼, ½, ½, ½, ¾, ¾, ¾, ¾, respectivement. Pour faciliter le calcul, nous pouvons convertir ces tailles en entiers en les multipliant par 4. Ainsi, nous avons des objets de tailles 1, 2, 3 et des bins de capacité 4. Nous voyons que le nombre de tailles différentes est égal à 3 dans cet exemple, donc d est égal à 3. Alors, nous avons 3 multiplicités : b_1, b_2 et b_3 où b_1 est la multiplicité de la taille 1, b_2 est la multiplicité de la taille 2, b_3 est la multiplicité de la taille 3. En comptant les objets appartenant aux tailles 1, 2 et 3, nous trouvons $b_1 = 3$, $b_2 = 3$ et $b_3 = 4$. Jusqu'à maintenant, nous avons seulement transformé l'« input » de l'exemple sous la forme de multiplicité.

Pour ce petit exemple, il est facile de créer les patterns. On peut mettre tous les objets de taille 1 dans un pattern, 2 objets de taille 1 et 1 objet de taille 2 dans un pattern, ou encore 1 objet de taille 1 et 1 objet de taille 3 dans un pattern pour avoir des patterns dont la capacité est maximisée. Il peut également y avoir des patterns suivant d'autres configurations en prenant des combinaisons différentes des objets. La matrice suivante montre tous les patterns possibles pour notre exemple.

$$M = \begin{pmatrix} 0 & 1 & 2 & 3 & 0 & 0 & 0 & 1 & 2 & 1 \\ 0 & 0 & 0 & 0 & 1 & 2 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \end{pmatrix}$$

Chaque colonne de la matrice M est un vecteur/pattern qui représente une combinaison différente des objets de taille 1, 2 et 3. A chaque colonne, une variable entière q_p (p=1,…,10) est associée pour déterminer le nombre de fois que le pattern p est utilisé. Soit I l'instance montrant cet exemple. Evidemment, $Lin(I) \leq Opt(I)$ avec $Lin(I)$ la solution de la relaxation linéaire et $Opt(I)$ la solution entière du modèle linéaire de Gilmore et Gomory. Soit q'_p (p=1,…,10) les variables associées à chaque colonne dans la solution relaxée.

Notons que l'étude de la relaxation linéaire du modèle de Gilmore et Gomory peut être intéressante. En effet, la différence entre la solution relaxée et la solution entière vient des instances appelées « instances résiduelles » [123]. Expliquons une instance résiduelle sur l'exemple de l'instance précédente : I. Modifions un peu l'instance I et supposons qu'elle soit composée de deux instances I_1 et I_2 avec I_2 l'instance contenant un objet de taille 2 et un objet de taille 3, et I_1 l'instance contenant le reste des objets. Si on résout ces instances séparément, nous obtenons $Lin(I_1) = 4$ avec $q'_6 = 1$, et $q'_{10} = 3$. Donc, nous avons la même solution pour la solution entière $Opt(I_1)$. Quant à la solution linéaire pour

I_2, on a $Lin(I_2)$=1,25. Puisque $Lin(I) = Lin(I_1) + Lin(I_2)$, la solution optimale pour la solution entière est forcément plus grande que 5,25. Notons que I_2 est appelé « instance résiduelle » de *I*. Nous reviendrons par la suite sur les instances résiduelles.

L'inconvénient du modèle linéaire de Gilmore et Gomory est qu'il contient un grand nombre de variables. De ce fait, même la relaxation linéaire du modèle en nombres entiers peut être difficile à résoudre. Il y a, malgré tout, une forte relation entre la solution relaxée et la solution entière de ce modèle. Le modèle de Gilmore et Gomory a une solution relaxée très proche de la solution entière. La différence entre la solution optimale entière et optimale fractionnaire est très petite, et, dans la plupart des cas, elles sont égales. Dans la littérature sur le bin packing, nous avons une conjecture appelée « *integer round up property* » [123].

Conjecture 1. (Scheithauer and Terno, [123]). Soit $I = <d, a, b>$ une instance de notre problème, *Opt(I)* la solution optimale entière du problème et *Lin(I)* la solution optimale du modèle relaxé. Pout toute instance *I* dans un problème de bin packing, nous avons l'inégalité suivante : $Opt(I) - \lceil Lin(I) \rceil \leq 1$

Scheithauer and Terno [123] ont démontré que la conjecture 1 marche pour les instances contenant 6 (ou moins) tailles d'objet différentes, *i.e.* $d \leq 6$. Shmonin et Sebo [124] ont amélioré ce résultat en démontrant que la conjecture 1 marche pour $d \leq 7$.

Dans notre problème, nous n'avons pas beaucoup de tailles d'objets différentes et, a priori, après avoir créé tous les patterns, il ne doit pas être difficile de résoudre le modèle linéaire en nombres entiers. Si ce n'est pas le cas, nous pouvons nous servir de la solution relaxée pour avoir une solution proche de l'optimale.

Soit, de nouveau, une instance $I = <d, a, b>$ d'un problème de bin packing pour laquelle nous avons créé les patterns $v_1, v_2, \ldots v_d, \ldots, v_P$. Nous savons, grâce à la théorie de la programmation linéaire, qu'une solution entière ou relaxée pour le modèle de Gilmore et Gomory contient au plus *d* patterns (Gilmore et Gomory, [50]). Sans perte de généralité, supposons que les *d* premiers patterns, *i.e.* $v_1, v_2, \ldots v_d$, sont dans la base, et donc, $q_p = 0$ pour $p = d + 1, \ldots, P$. La valeur de la solution relaxée est alors :

$$Lin(I) = \sum_{p=1}^{d} q_p \text{ sachant que } \sum_{p=1}^{d} q_p * v_{ip} = b_i$$

où q_p n'est pas forcément un nombre entier. Alors, chaque q_p peut être décomposé de façon suivante :

$$q_p = \lfloor q_p \rfloor + q'_p$$

Nous avons simplement décomposé q_p en deux parties : partie entière et partie fractionnaire. Notons

$$b'_i = \sum_{p=1}^{d} q'_p * v_{ip}$$

Alors, l'instance I_r, avec $I_r = < d, a, b' >$, est une instance résiduelle pour I. Notons I_e l'instance restant après avoir enlevé I_r de I. Il est alors clair que

$$Lin(I_e) = \sum_{p=1}^{d} \lfloor q_p \rfloor = Opt(I_e)$$

De plus,

$$Lin(I) = Lin(I_e) + Lin(I_r).$$

Il est facile de voir que « $Opt(I) \leq Opt(I_e) + Opt(I_r)$ ». Ainsi, nous pouvons donner le corollaire suivant :

Corollaire 2. (Shmonin et Sebo [124]) Pour toute instance $I = <d, a, b>$ d'un problème de bin packing, la différence maximale entre $Opt(I)$ et $Lin(I)$ correspond aux instances résiduelles.

Le corollaire est assez important et peut être utile dans le cas où le modèle linéaire de Gilmore et Gomory ne peut pas être résolu à l'optimal pour trouver une solution entière. En effet, pour une instance résiduelle I_r de I, on a $Opt(I_r) \leq d$, car, nous pouvons mettre les objets de I_r dans des patterns identiques aux patterns utilisés $v_1, v_2, \ldots v_d$. Puisque chaque vecteur v_j peut être utilisé au plus 1 fois dans $Opt(I_r)$, on a $Opt(I_r) \leq d$.

Dans le cas où le modèle linéaire de Gilmore et Gomory ne peut pas être résolu à l'optimal, nous pouvons d'abord chercher la solution relaxée. Puisque cette solution peut contenir des vecteurs v utilisés fractionnairement, nous pouvons les arrondir au nombre entier inférieur pour obtenir les premiers bins. De cette façon, nous obtenons la solution optimale pour l'instance I_e. Maintenant, il reste à mettre les objets de l'instance résiduelle I_r dans les bins. Pour le faire, un simple algorithme, comme *FFD*, peut être appliqué.

Les tests numériques montreront si nous sommes capables de résoudre nos instances à l'optimal. Nous donnons, par la suite, un algorithme qui est capable de créer tous les patterns d'une instance donnée.

9.5. Algorithme pour la génération des patterns

Dans notre méthode de résolution, nous allons créer explicitement les patterns du modèle linéaire donné dans la section précédente. Même si la génération de tous les patterns nécessite une durée exponentielle, il est possible d'appliquer un algorithme à notre problème. Nous donnons ci-dessous le pseudo-code de l'algorithme de génération explicite des patterns:

Notations utilisées

- d : nombre de tailles différentes dans l'instance (nombre total de multiplicités),
- *Cap* : capacité d'un bin
- $taille_k$: taille des objets appartenant à la multiplicité k
- k : indice représentant les objets de la multiplicité k avec $0 \leq k \leq d$-1
- *Wrestant* : nombre réel pour calculer l'espace disponible dans un bin
- *aMax* : nombre entier pour calculer le nombre maximal d'objets qui peut être mis dans un bin pour la multiplicité k donnée après avoir placé un certain nombre d'objets de multiplicité k-1
- *ndp* : nombre correspondant au numéro de pattern
- a[k] : tableau d'entiers à une dimension pour retenir le nombre d'objets pour un numéro de pattern et pour une multiplicité, k, donnée
- M[*ndp*][k] : tableau d'entiers à deux dimensions pour retenir le nombre d'objets dans le pattern *ndp* pour la mulriplicité k

Algorithme *Créer pattern* (*CP*)

```
0 ndp = 0
1 Creerpattern (k, Wrestant, a)
2       si k = d
3               ndp = ndp + 1 ;
4               Pour entier i de 0 à d-1
5                       M[ndp][i] = a[i] ;
6               Fin pour
7       sinon
8               aMax = min(⌊W_restant / taille_k⌋, nombre d'objets de
                                                        multiplicité k)
9               Pour  entier i de 0 à aMax
10                      a[k] = i ;
11                      creerpattern(k + 1, Wrestant -( a[k] * taille_k), a) ;
12              Fin pour
13      Fin si
13 Fin Creerpattern
```

L'algorithme *CP* est un algorithme récursif qui se lance par l'appel suivant : Créerpattern (0, *Cap*, *a*). La valeur initiale de *k* est égale à 0, car, l'algorithme commence par placer les objets appartenant à la multiplicité 0. Puisqu'à l'état initial, aucun objet n'a été placé dans un bin, la valeur de l'espace disponible est égale à *Cap*. Le nombre maximum de fois qu'un objet peut se retrouver dans un bin dépend de sa taille. Donc, avec $aMax = Wrestant / multip_k$, on calcule le maximum de fois, *aMax*, que l'objet de multiplicité *k* est mis dans un batch. Si le nombre d'objets de multiplicité *k* est plus petit que la division de *Wrestant* par $multip_k$, nous n'avons pas besoin de créer des patterns dont l'objet de multiplicité de *k* apparaît plus que nécessaire. Puis, de 0 à *aMax*, on essaie de voir combien d'autres objets, de multiplicité supérieure à *k*, peuvent être mis dans le batch.

Exemple illustratif : Soit 2 tâches de taille 2 et 3 tâches de taille 3. Soient des bins de capacité 5.

Résolution de l'exemple illustratif avec *CP*

L'algorithme *CP* commence d'abord à placer les objets appartenant à la première multiplicité, *i.e.* il essaie d'abord de placer les objets de taille 2. Ici, l'indice *k* représente les objets de taille 2 s'il est égal à 0. Si *k* est égal à 1, les objets de taille 3 sont représentés.

Pour *k* égal à 0, on entre dans le cas inductif de l'algorithme, *i.e.* les lignes de codes entre « sinon » et « fin si ». Dans le cas inductif, on calcule le nombre maximal d'objets de taille 2 que l'on peut insérer dans un bin. Puisque jusqu'à maintenant aucun objet n'a été placé dans un bin, le nombre maximal d'objet de taille 2 que l'on peut mettre dans un bin est égal à $\lfloor 5/2 \rfloor = 2$ ($Cap / multip_0$). Evidemment le nombre minimal d'objets de taille quelconque que l'on peut placer dans un bin est 0. Donc, on entre dans une boucle « pour » afin de mettre *i* objet de taille 2 dans un bin où *i* varie entre 0 et 2. Pour chaque valeur différente de *i*, l'algorithme est appelé de nouveau, mais maintenant pour l'indice 1.

Prenons le cas où 0 objet de taille 2 soit placé dans un bin. Alors, l'algorithme sera appelé de la façon suivante : creerpattern(1, *5* -(0 * *2*), *a*). Cet appel signifie que l'on essaie maintenant de placer les objets de taille 3 dans un pattern où aucun objet de taille 2 a été placé précédemment. Puisque la valeur du *k* est inférieure à 2 (*d* = 2 dans l'exemple), on entre encore une fois dans le cas inductif de l'algorithme. Ainsi, nous calculons le nombre maximal par $\lfloor 5/3 \rfloor = 1$. Cela veut dire que quand il y a 0 objet de taille 2 placé initialement dans un bin, on peut mettre au maximum 1 objet de taille 3 dans ce bin. De cette façon, nous

essayons tous les placements possibles, de 0 à 1, pour les objets de taille 3 quand il y a 0 objet de taille 2 dans un bin.

Quand la valeur de *k* est incrémentée de 1 lors de l'appel récursif dans la boucle « pour », nous obtenons *k* égal à 2, et donc, à la prochaine exécution de l'algorithme, nous entrons dans le cas de base, *i.e.* les lignes de codes entre « si *k=d* » et « sinon ». Ce cas veut dire qu'un pattern a complètement été créé .

Enfin, les pattern créés sont contenus dans un tableau à deux dimensions, *i.e.* dans le tableau M. Les mêmes appels résursifs sont effectués pour trouver d'autres nombres d'objets de taille 3 dans un pattern quand il y a 1 et 2 objets de taille 2 initialement placés dans un pattern. La matrice des patterns finalement obtenue est la suivante :

$$\begin{pmatrix} 0 & 0 & 1 & 1 & 2 \\ 0 & 1 & 0 & 1 & 0 \end{pmatrix}$$

Une fois que la matrice des patterns est générée, elle est transmise au modèle linéaire de Gilmore et Gomory. Notons que l'algorithme *CP* considère des tailles entières. Donc avant d'exécuter l'algorithme, nous multiplions les tailles d'une instance et la capacité des laveurs de notre problème par un coefficient permettant de n'avoir que des tailles entières.

9.6. Application sur un cas inspiré du réel

Dans cette section, nous allons d'abord nous intéresser à la durée de résolution pour la génération explicite des patterns avec l'algorithme *CP*. La solution du modèle linéaire correspondra à nos cycles de lavage, chacun contenant un certain nombre d'ensembles de DMR. Nous allons ensuite tester l'impact de la minimisation du nombre de cycles de lavage via un modèle de simulation.

9.6.1. Tests numériques avec le modèle linéaire en nombres entiers

La génération explicite des patterns n'est, peut être, pas très pratique puisqu'elle nécessiterait une durée exponentielle. La limite de résolution avec l'algorithme que nous avons proposé dépend largement du nombre de tailles différentes dans le problème. Puisque nous en avons 36 au maximum dans notre problème de stérilisation, l'algorithme *CP* devrait être capable de s'exécuter sans problème de mémoire et en un temps assez court.

Pour les expérimentations numériques, nous avons créé 150 instances contenant 30, 50, et 70 objets/ensembles de DMR (50 instances pour chacun). Rappelons que les tailles des ensembles de DMR sont générées de la façon suivante : U[1, 36]*Capacité/36 avec U une distribution uniforme. Nous avons utilisé cette distribution pour affecter une taille à chaque objet des instances. La capacité des laveurs est de 6 DIN. Nous avons mesuré la durée de résolution pour la création des patterns et la durée de résolution avec CPLEX pour le modèle linéaire de Gilmore et Gomory.

Sur la figure 9.1, nous montrons, pour chaque instance testée, la durée de génération des patterns.

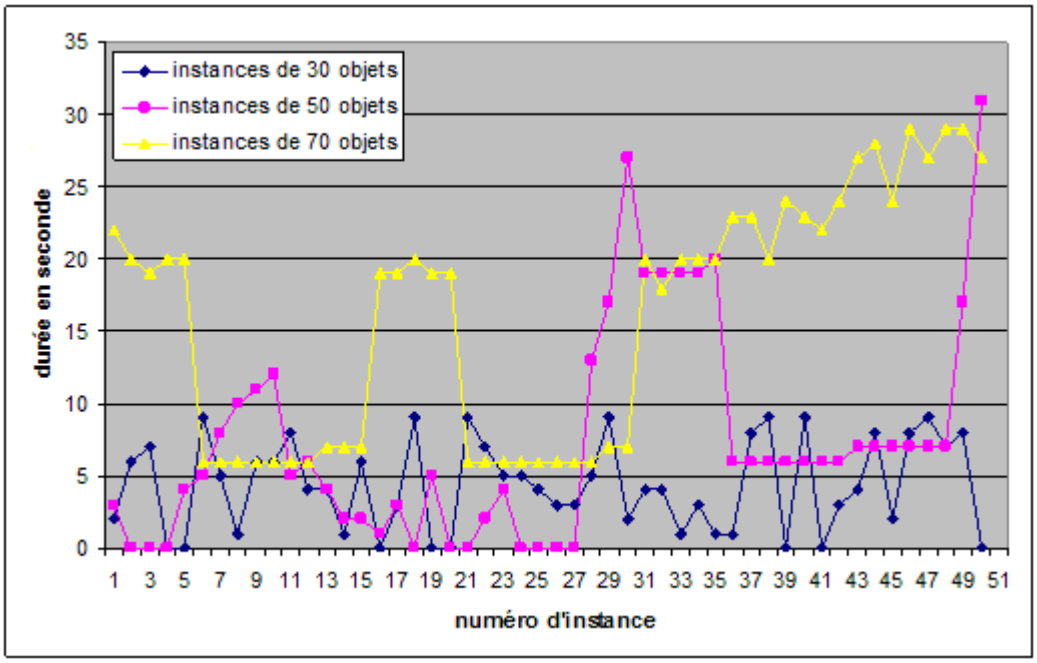

Figure 9.1. Durée de génération des patterns avec *CP* pour les instances de 30, 50 et 70 objets

Nous voyons que, quand il y a 30 objets dans les instances, la durée de création des patterns est plus courte que pour les autres instances. La raison de ce fait dépend du nombre de tailles différentes présentes dans une instance car la résolution avec l'algorithme *CP* dépend de ce nombre de tailles différentes. Alors évidemment, la probabilité d'avoir des tailles différentes en présence de 30 objets est moindre que quand il y a 50 ou 70 objets dans une instance. Nous voyons que la durée de génération des patterns a tendance à augmenter lorsque le nombre d'objets dans les instances augmente. Malgré tout, il y a un point qui attire notre attention. Les durées minimales, ainsi que maximales, d'exécution pour les instances de 50 et 70 objets sont à peu près les mêmes. Cette observation est due à la génération des tailles dans les instances. L'utilisation de la distribution

uniforme fournit, de temps en temps, des tailles similaires dans une instance, ce qui permet d'avoir des durées d'exécution assez petites avec *CP*. Une autre raison est lorsque les tailles sont assez grandes. Si une instance contient majoritairement des objets de grande taille (*i.e.* plus grandes que la moitié de la capacité d'un bin), alors, la génération des patterns s'accélère puisque les objets de grande taille ne peuvent pas être mis ensemble dans un même pattern. En ce qui concerne la durée de résolution du modèle linéaire après la création des patterns, même avec la génération explicite des patterns, la taille du modèle linéaire est assez petite pour qu'il soit résolu jusqu'à l'optimalité. La plus grande durée d'exécution avec CPLEX est de 5 secondes.

Les tests numériques ci-dessus montrent que l'algorithme de génération explicite de toutes les colonnes permet de trouver la solution optimale dans une durée assez courte (moins de 30 secondes). Par curiosité, nous avons testé les mêmes instances, en utilisant l'approche de génération de colonnes développée par Gilmore et Gomory [50]. Son principe est de générer uniquement les colonnes de coûts réduits les plus faibles. Elles sont obtenues à partir de la résolution d'un problème de sac à dos. Pour implémenter cette méthode, nous avons utilisé l'exemple de cutting stock du logiciel CPLEX[3]. Nous avons observé que cette méthode de génération de colonnes est beaucoup plus rapide puisque les instances sont maintenant résolues en moins de 5 secondes. Il est donc plus avantageux d'utiliser cette approche pour des plus grandes instances.

9.6.2. Impact de la minimisation du nombre de cycles de lavage sur le processus de stérilisation

L'heuristique *FFD* est l'une des méthodes les plus efficaces pour la résolution du problème de bin packing. Rappelons qu'elle a une garantie de performance égale à (11/9)**nb* +1 avec *nb* la solution optimale. Nous avons donc comparé la solution optimale, trouvée par la méthode expliquée dans la section précédente, aux solutions trouvées par *FFD*.

Nous avons déjà développé des modèles de simulation dans le chapitre précédent. Nous testons donc le chargement optimal et *FDD* sur les mêmes modèles de simulation. L'analyse que nous faisons consiste à tester 2 critères :

- le nombre de cycles de lavage lancés,
- le nombre de boîtes et de sachets stérilisés

[3] Après l'installation de CPLEX, plusieurs exemples sont à disposition de l'utilisateur pour l'apprentissage du logiciel. L'exemple de cutting stock applique la méthode de génération de colonnes développée par Gilmore et Gomory [50].

Dans le modèle de simulation, nous avons encore une fois quatre laveurs automatiques, cinq postes de conditionnement des boîtes et un poste pour les sachets, trois autoclaves. Nous utilisons les mêmes instances que celles utilisées dans la section 9.5.1. Puisque les ensembles de DMR arrivent au service de stérilisation déjà pré désinfecté et rincés, l'optimisation de la durée de pré désinfection n'est plus un critère d'optimisation. De plus, tous les DMR sont disponibles en même temps devant les laveurs. Donc, sans perte de généralité, nous fixons leur date de disponibilité égale à 0. Le nombre de boîtes et de sachets contenus par un ensemble de DMR est déterminé par les distributions uniformes U[1, 5] et U[0, 3], respectivement. Nous simulons chaque instance pour un horizon de 12 heures dans une journée de travail. Le tableau 9.1 présente le nombre moyen de cycles de lavage lancés par le chargement optimal des laveurs et par *FFD*.

Tableau 9.1. Comparaison du nombre de cycles de lavage lancés par *FFD* et par le chargement optimal

	70 tâches	50 tâches	30 tâches
FFD	28,4	18,9	10,9
Optimal	25,3	17,1	10,1

La différence entre ces deux méthodes a tendance à augmenter lorsque le nombre d'ensembles de DMR augmente. Mais, *FFD* est une heuristique puissante et donc la solution qu'elle fournit n'est pas très éloignée de la solution optimale.

A priori, nous n'attendions pas beaucoup de différence entre la solution optimale et *FFD* pour le nombre de boîtes et de sachets stérilisés dans une journée. Nous avons un horizon de temps de 12 heures entre l'ouverture et la fermeture du service de stérilisation. Le tableau 9.2 montre le pourcentage de DMR stérilisés par *FFD* et par le chargement optimal des laveurs. Nous avons simplement divisé la quantité stérilisée à la fin de la journée par la quantité totale de DMR qui est entrée dans le service au début de la journée.

Tableau 9.2. Pourcentage du DMR stérilisés par rapport à la quantité entrée dans le service

	70 tâches	50 tâches	30 tâches
FFD	75%	88%	100%
Opt.	79%	92%	100%

Le nombre moyen de boîtes et de sachets entrant dans le processus de stérilisation est 290 pour les instances de 70 tâches, et 225 pour les instances de 50 tâches. Alors, la stérilisation de 92% des DMR pour les instances de 50 tâches est équivalente à, environ, 210 boîtes et sachets. La différence de 15 boîtes et sachets est assez petite. Pour les instances de 70 tâches, la quantité maximale

stérilisée est autour de 80%, ce qui est équivalent à 225-230 boîtes et sachets. La raison principale pour cette observation est le nombre de postes de conditionnement. Même si le chargement des laveurs est optimal, la seule façon pour minimiser l'attente des DMR dans le stock de conditionnement est d'augmenter le nombre de ces postes. Nous avons observé qu'en rajoutant 2 postes de conditionnement des boîtes (*i.e.* 7 postes au lieu de 5), il est possible stériliser tous les DMR pour les instances de 70 tâches.

9.7. Conclusion

Nous avons développé dans cette section une méthode de résolution pour minimiser le nombre de cycles de lavage lancés. Notre hypothèse est la disponibilité simultanée des ensembles de DMR devant les laveurs. Un exemple pour cette hypothèse est simplement le cas des services de stérilisation externe. Un service de stérilisation est chargé de stériliser les DMR d'un hôpital qui sont envoyés à ce service en même temps. Un autre exemple serait un service de stérilisation appartenant à un hôpital qui n'effectuerait pas le lavage au fur et à mesure, mais attendrait l'accumulation d'une grande quantité de DMR pour commencer le lavage, alors un grand nombre de DMR deviendraient disponibles pour le lavage en même temps.

Un critère d'optimisation est la minimisation du nombre de cycles de lavage lancés. Nous avons ainsi modélisé l'étape de lavage comme un problème de bin packing. D'après nos investigations, les ensembles de DMR n'ont pas beaucoup de différence entre eux au niveau de leurs tailles. Nous avons au plus 36 tailles différentes pour les ensembles de DMR. Ce nombre étant assez petit, il nous a permis d'utiliser une méthode existant dans la littérature. Le modèle linéaire de Kantorovich n'étant pas suffisamment rapide pour la résolution du problème de bin-packing (*cf.* chapitre 6), nous avons utilisé le modèle linéaire proposé par Gilmore et Gomory. En effet, Gilmore et Gomory ont proposé une méthode appelée « génération de colonnes » pour la résolution de leur modèle. Le nombre de tailles différentes dans notre problème étant assez petit, nous avons donc décidé de créer explicitement les patterns qui sont les « input » de Gilmore et Gomory. Pour ce faire, nous avons proposé un algorithme récursif dont la durée d'exécution varie entre 0 et 30 secondes pour les instances que nous avons créées. En considérant que l'achat de CPLEX peut être couteux pour les services de stérilisation, nous avons voulu comparé les résultats optimaux aux solutions données par une heuristique performante. Ainsi, nous avons comparé les solutions optimales aux solutions données par l'algorithme *FFD*. Les résultats numériques montrent que la performance de *FFD* n'est pas loin de la solution optimale à la fois pour le nombre de cycles de lavage lancés et pour le nombre de boîtes et sachets stérilisés.

Conclusion de la partie 4

Nous avons effectué dans cette partie une étude de cas où une méthode de couplage entre l'optimisation combinatoire et la simulation est développée. Ce couplage consiste à tester l'impact de la minimisation du nombre de cycles de lavage lancés sur le processus de stérilisation via un modèle de simulation, comme cela a été fait dans la partie 3.

L'orientation de l'étude est plutôt pour les services de stérilisation externes, ou bien, pour les services qui préfèrent commencer le lavage des ensembles de DMR quand il y en a un grand nombre dans le stock de lavage. Ainsi, les ensembles de DMR sont simultanément disponibles. Cette étude nous a montré que l'algorithme *FFD* est bien puissant comparé à la méthode optimale que nous avons proposée pour la minimisation du nombre de cycles de lavage. Ainsi, cette heuristique peut être utilisée par les services de stérilisation sans l'achat du CPLEX, qui causerait un coût supplémentaire. Un autre résultat que nous avons retenu grâce aux simulations est le nombre de postes qui doivent se trouver dans un service de stérilisation. D'après les résultats des tests, un service ayant 4 postes de lavage, 5 postes de conditionnement en boîtes et 3 autoclaves est capable de stériliser 50 ensemble de DMR par jour. D'après quelques tests supplémentaires, en augmentant uniquement le nombre de postes de conditionnement en boîtes, de 5 à 7, il devient possible de stériliser l'ensemble des 70 ensembles de DMR.

~ CONCLUSION GENERALE et PERSPECTIVES ~

Nous avons proposé dans cette thèse des méthodes d'optimisation pour l'étape de lavage d'un service de stérilisation afin d'améliorer la performance du processus de stérilisation. Un service de stérilisation est composé de plusieurs étapes dont le lavage est généralement un goulet d'étranglement. Cette étape est composée des laveurs chargés du lavage des dispositifs médicaux réutilisables (DMR) et d'un stock où les DMR sont mis en attente avant d'être lavés. Dans un hôpital possédant un service de stérilisation, les DMR arrivent dans le service de stérilisation à différents moments tout au long d'une journée. Si un hôpital choisit d'externaliser la stérilisation de ces DMR, un service de stérilisation externe est chargé de stériliser les DMR de cet hôpital. Dans les deux cas, il s'agit d'ensembles de DMR ayant des tailles différentes et de laveurs à l'étape de lavage qui sont capables de laver plusieurs ensembles de DMR en même temps. Ces particularités du problème nous ont permis de modéliser l'étape de lavage comme un problème d'ordonnancement par batch.

Pour notre problème d'ordonnancement, nous avons proposé des méthodes de résolution autour de deux axes : 1-méthodes de résolution offline, 2-méthodes de résolution semi-online et online. Les méthodes offline que nous avons développées nous ont permit de travailler sur des problèmes théoriques. Nous avons d'abord choisi deux fonctions objectifs classiques de la littérature d'ordonnancement qui sont la minimisation du C_{max} et de la $\sum C_j$. D'après la recherche bibliographique que nous avons faite, la minimisation du C_{max} sur des machines parallèles est le seul critère étudié dans la littérature d'ordonnancement par batch. Pour ce problème, nous avons d'abord étudié des cas particuliers pour proposer des algorithmes exacts. Ces cas particuliers considèrent des tailles unitaires et aussi des tailles particulières appelées *fortement divisibles*. Pour le cas général avec des tailles arbitraires, nous avons développé un modèle PLNE et nous l'avons comparé à un autre PLNE de la littérature. Les tests numériques nous ont montré que le fait d'avoir des durées d'exécution égales nous a permis de développer un modèle PLNE plus efficace au niveau du temps de résolution. Nous avons aussi proposé un algorithme d'approximation avec un ratio d'approximation de 2. A notre connaissance, il n'y a pas d'algorithme

d'approximation, ni d'heuristique de complexité polynomiale, pour notre problème dans la littérature d'ordonnancement. Pour la minimisation de la $\sum C_j$, un critère qui n'a pas encore été étudié dans la littérature, nous avons considéré des cas particuliers et nous avons proposé des algorithmes exacts et approchés. Pour le cas général, nous avons proposé un modèle PLNE et des heuristiques. Suivant les critères classiques des problèmes d'ordonnancement, nous avons continué avec une nouvelle fonction objectif qui a pour but d'optimiser les durées de pré désinfection des DMR. Pour l'optimisation bi-critère de cette nouvelle fonction objectif avec la minimisation du nombre de batch, deux méthodes de résolution fonctionnant avec la résolution successive des modèles PLNE ont été proposées. Enfin, une heuristique puissante, de complexité polynomiale, a été proposée. Outre le point de vue scientifique, les approches offline nous ont permis de voir combien il serait possible d'améliorer le fonctionnement de l'étape de lavage dans une configuration purement offline.

En fonction du fonctionnement de la plupart des services de stérilisation, nous avons continué nos études avec des approches semi-online et online. Ces études prennent en compte des incertitudes, comme retard, avance, etc., concernant l'arrivée des ensembles de DMR au service de stérilisation. Nous avons développé deux algorithmes semi-online et online afin d'optimiser les durées de pré désinfection des DMR. L'un des intérêts pratiques de ces résultats est que nous avons montré la possibilité de supprimer le poste de rinçage manuel tout en obtenant de bonnes durées de pré désinfection des DMR avec nos méthodes semi-online et online. Nous avons ensuite intégré ces algorithmes dans des modèles de simulation pour tester leur impact sur l'ensemble du service de stérilisation. Nous avons ainsi mis en exergue que l'étape de conditionnement des boîtes ralentissait le processus de stérilisation en diminuant le nombre de boîtes et sachets stérilisées par jour. Ainsi, nous avons pu proposer que le poste de rinçage manuel soit remplacé par un poste supplémentaire de conditionnement (poste également manuel). Les résultats de simulation ont montré qu'en l'absence de rinçage manuel et en présence d'un poste de conditionnement en boîtes supplémentaire, nos méthodes de résolution sont capables d'augmenter le nombre de DMR stérilisés par jour tout en garantissant des bonnes durées de pré désinfection.

Enfin, nous avons effectué une étude de cas qui a pour but de minimiser le nombre de cycles de lavage lancés en présence de l'arrivée simultanée de tous les ensembles de DMR. Pour ce faire, nous avons utilisé une méthode existante dans la littérature de bin packing. Nous avons aussi couplé cette méthode avec un modèle de simulation pour tester son impact sur l'ensemble du service de stérilisation, comme cela a été fait pour les algorithmes semi-online et online dans la partie 3.

Il est possible d'étendre les travaux que nous avons faits dans cette thèse. En ce qui concerne les méthodes d'optimisation offline, il serait possible de développer certains résultats et/ou de travailler sur d'autres méthodes d'optimisation. Parmi les résultats à développer, nous pouvons citer l'algorithme d'approximation proposé dans la section 5.3.2. La garantie de performance de cet algorithme peut être améliorée. Parmi les nouvelles méthodes de résolution qui peuvent être proposées, nous pouvons compter, par exemple, des méta-heuristiques ou des méthodes exactes comme branch and cut. Ou encore, il serait possible de travailler sur d'autres cas particuliers (par exemple, comme cela a été fait pour la minimisation du C_{max}, un cas avec des tâches ayant des tailles fortement divisibles peut être étudié pour la minimisation de la $\sum C_j$).

Une autre perspective concerne l'amélioration des méthodes/résultats présentés dans cette thèse. Il pourrait être possible de modifier l'algorithme *TIH correctif* pour qu'il ait un aspect encore plus *correctif*. Dans son cas actuel, il est un algorithme de ré-optimisation qui utilise certaines procédures à chaque fois qu'une tâche est en retard ou en avance. Ce type d'application peut être difficile dans un service de stérilisation. Ainsi, au lieu de lancer ces procédures de correction (*i.e.* l'algorithme *TIH*) lors de chaque anomalie d'arrivée, il pourrait être possible de fixer des marges de retard/avance pour les tâches, et de lancer *TIH* quand une tâche arrive en dehors de sa marge de retard/avance. De cette façon, le pré planning n'aurait pas besoin d'être mis à jour souvent.

En ce qui concerne l'extension de nos travaux d'un point de vue intérêt pratique pour la stérilisation, il pourrait être possible de proposer d'autres méthodes plus puissantes, offline ou online, pour l'optimisation bi-critère de la durée de pré désinfection et du nombre de cycles de lavage lancés. Ou encore, il serait possible de considérer d'autres fonctions objectifs multicritères. Outre les méthodes de recherche opérationnelle, les outils du management peuvent s'appliquer à l'organisation du processus de stérilisation. Parmi ces outils, nous pouvons compter le management de risque, application des méthodes comme 5S, 6σ pour augmenter la qualité de l'organisation de stérilisation.

Comme précisé dans le manuscrit, nous avons construit nos modèles de simulation en considérant un modèle générique de simulation proposé pour les services de stérilisation. Mais, nous pouvons encore aller plus loin pour que ces modèles soient plus représentatifs d'un service de stérilisation réel. Pour ce faire, il pourrait être possible d'inclure dans le modèle de simulation la main-d'œuvre, *i.e.* les opérateurs de stérilisation. De cette façon, les incertitudes concernant l'effet humain peuvent être prises en compte aussi.

Une autre piste d'extension pourrait être la création d'un lien avec les blocs opératoires. Ce lien nécessiterait d'abord un système de communication

entre le service de stérilisation et les blocs opératoires. Ensuite, le planning des opérations de stérilisation pourrait être fait en fonction du besoin des chirurgiens dans les interventions. De cette façon, nous aurons affecté une priorité de stérilisation pour les ensembles de DMR en prenant en compte le nombre d'exemplaire de DMR dans le stock, la date d'intervention chirurgicale, etc. Ou encore dans l'autre sens, on pourrait établir le planning des chirurgiens en fonction des règles permettant d'avoir une stérilisation plus efficace.

Références

[1] Albert, F., Di Mascolo, M. and Marcon, E. (2008). Analyse de différentes stratégies de remplissage des laveurs dans un service de stérilisation de dispositifs médicaux. *7è Conférence Internationale de Modélisation et SIMulation – MOSIM'08, 31Mars au 2 Avril 2008, Paris, France.*

[2] Azizoglu, M. and Webster, S. (2000). Scheduling a batch processing machine with non-identical job sizes. *International Journal of Production Research*, 38 (10) : 2173-2184.

[3] Azizoglu, M. and Webster, S. (2001). Scheduling a batch processing machine with incompatible job families. *Computers & Industrial Engineering*, 39 (3-4) : 325-335.

[4] Bac, C. and Cornilleau, G. (2002). Comparaison internationale des dépenses de santé : une analyse des évolutions dans sept pays depuis 1970. *DREES Direction de la recherche des études de l'évaluation et des statistiques, no. 175, Juin 2002*

[5] Bai, S., Zhang, F., Li. S., and Liu, Q. (2007). Scheduling an unbounded batch machine to minimize maximum lateness. *Lecture Notes in Computer Science,* 4613 : 172-177.

[6] Balasubramanian, H., Mônch, L., Fowler, J. W. and Pfund. M. E. (2004). Genetic algorithm based scheduling of parallel batch machines with incompatible job families to minimize total weighted tardiness. *International Journal of Production Research,* 42(8) : 1621-1638.

[7] Baptiste, P. (2000). Batching identical jobs. *Mathematical methods of operations research,* 52(3) : 355-367

[8] Bardet, E. (2003). Externalisation de la stérilisation : conditions de mise en oeuvre, avantages, inconvénients, *Mémoire de l'école nationale de la santé publique, Rennes, France*

[9] Bernard, V. and Lacroix, P. (2001). Restructuration d'un service de stérilisation dans le cadre d'une démarche qualité. *ITBM-RBM,* 22(2) : 116-124

[10] Brucker, P., Gladky, A., Hoogeveen, H., Kovalyov, M.Y., Potts, C.N., Tautenhahn, T., and De Velde. S.V. (1998). Scheduling a batching machine. *Journal of Scheduling,* 1(1) : 31-54.

[11] Burke, E. K., De Causmaecker, P., Berghe, G. V. and Landeghem, H. V. (2004). The state of the art of nurse rostering, *Journal of Scheduling*, 7(6) : 441-499

[12] Cal, M-C., Deng, X., Feng, H., Li, G. and Liu., G. (2006). A PTAS for minimizing total completion time of bounded batch scheduling. *Lecture Notes in Computer Science,* 2337: 304 314.

[13] Cao, J., Yuan, J., Li, W., Bu, H. (2011). Online scheduling on batching machines to minimise the total weighted completion time of jobs with

precedence constraints and identical processing times. *International Journal of Systems Science*, 42(1) : 51.
[14] Cardoen B, Demeulemeester E, Beliën J, (2010). Operating room planning and scheduling: A literature review, *European Journal of Operational Research*, 201 : 921 - 932.
[15] Chahed, S., Marcon, E., Sahin, E., Feillet, D. and Dallery Y. (2009) Exploring new operational research opportunities within the home care context : the chemotherapy at home. *Health Care Management*, 12 :179-191
[16] Chandru, V., Lee, C.-Y., and Uzsoy. R. (1993). Minimizing total completion time on batch processing machines. *International Journal of Production Research*, 31(9) : 2097-2121.
[17] Chandru, V., C.-Y Lee, and R. Uzsoy. (1993). Minimizing total completion time on a batch processing machine with job families. *Operations Research Letters*, 13(2) : 01-05.
[18] Chang, Y., Damodaran, P. and Melouk, S. (2004) Minimizing makespan on parallel batch processing machines. *International Journal of Production Research*, 42 (19) : 4211-4220.
[19] Chen B, Deng X, Zang W. (2004). On-Line Scheduling a Batch Processing System to Minimize Total Weighted Job Completion Time. *Journal of Combinatorial Optimization*, 8(1) : 85-95.
[20] Chen H, Zhang Y, Yong X. (2009). On-Line Scheduling on a Single Bounded Batch Processing Machine with Restarts. *In: Proceedings of the First International Workshop on Education Technology and Computer Science - Washington, DC, USA*, p. 868–871.
[21] Chen, H., Du, B. and Huang, G.Q., (2010). Metaheuristics to minimise makespan on parallel batch processing machines with dynamic job arrivals. *International Journal of Computer Integrated Manufacturing*, 23 (10) : 942-956.
[22] Cheraghi, S. H., Vishwaram, V. and Krishnan. K.K. (2003). Scheduling a single batch processing machine with disagreeable ready times and due dates. *International Journal of Industrial Engineering*, 10(2) : 175-187.
[23] Chung, S.H., Tai, Y.T., and Pearn, W.L., (2009). Minimizing makespan on parallel batch processing machines with non-identical ready time and arbitrary job sizes. *International Journal of Production Research*, 47 (18) : 5109-5128.
[24] Coffman Jr, E.G., Garey, M.R and Johnson, D.S. (1987). Bin packing with divisible item sizes. *Journal of Complexity*. 3(4) : 406-428
[25] Coffman, E. G., Garey, M. R. and Johnson., D. S. (1997). Approximation Algorithms for Bin Packing: A Survey. In Approximation Algorithms for NP-Hard Problems, Dorit S. Hochbaum (editor), PWS Publishing Company, 1997, pp. 46-93.

[26] Damodaran, P., Velez-Gallego, M.C. and Maya, J., (2009). A GRASP approach for makespan minimization on parallel batch processing machines. *Journal of Intelligent Manufacturing*, doi:10.1007/s10845-009-0272-z.
[27] Damodaran, P. and Velez Gallego, M.C., (2009). Heuristics for makespan minimization on parallel batch processing machines with unequal job ready times. *International Journal of Advanced Manufacturing Technology,* 49 (9-12) : 1119-1128.
[28] Deng, X., H. Feng, Li, G., and Shi, B. (2005). A PTAS for semiconductor burn-in scheduling. *Journal of Combinatorial Optimization,* 9(1) : 5-17.
[29] Deng, X., Feng, H., Zhang, P., Zhang, Y., and Zhu, H. (2004). Minimizing mean completion time in a batch processing system. *Algorithmica,* 38(4) : 513-528.
[30] Deng, X., and Zhang, Y. (1999) Minimizing mean response time in batch processing system. *Lecture Notes in Computer Science*, 1627 : 231-240.
[31] Di Mascolo, M., Gouin, A. & Ngo Cong, K. (2006) Organization of the production of sterile medical devices, *Proceedings of the 12th IFAC Symposium on Information Control Problems in Manufacturing* - INCOM'2006
[32] Dupont, L. and Dhaenens-Flipo, C., (2002). Minimizing the makespan on a batch processing machine with non-identical job sizes: an exact procedure. *Computers and Operations Research*, 29 (7) : 807-819.
[33] Dupont, L. and Jolai Ghazvini. F. (1997). Branch and bound method for minimizing mean flow time on a single batch processing machine. *International Journal of Industrial Engineering : Practice and Theory*, 4 : 197-203.
[34] Dupont, L. and Jolai Ghazvini, F., (1998). Minimizing makespan on a single batch processing machine with non-identical job sizes. *European Journal of Automation*, 32 (4) : 431-440.
[35] Eduard, S. (2005). Contrôle des performances de lavage en laveur-désinfecteur : Sensibilité aux paramètres de lavage de trois tests de salissures et analyse comparative des tests de détection de souillures résiduelles, *Thèse à l'Université Claude Bernard – Lyon 1*
[36] EESS (2007). Enquête Electronique sur les Services de Stérilisation, *http://www.laspi.fr/ipi*
[37] Fang, Y., Liu, P., Lu, X. (2011). Optimal on-line algorithms for one batch machine with grouped processing times. *J. Comb. Optim*, 22 (4) : 509-516
[38] Fei, H., Chu, C., Meskens, N. and Artiba. (2008). Solving surgical cases assignment problem by a branch-and-price approach. *International Journal of Production Economics,* 112(1) : 96-108
[39] Féniès, P., Gourgand, M. and Rodier, S. (2005). Interoperable and multi-flow software environment : Application to health care supply chain. *Lecture Notes in Computer Science*, 4103(2006) : 311-322

[40] Fenina, A., Le Garrec, M-A., and Koubi, M. (2010). Les comptes nationaux de la santé en 2009. *DREES Direction de la recherche des études de l'évaluation et des statistiques, no. 736, Septembre 2010*
[41] Fernandez de la Vega, W. and Lueker, G.S. (1981). Bin packing can be solved within $1+\varepsilon$ in linear time. *Combinatorica*, 1 : 349-355
[42] Friesen, D.K. and Langston, M.A. (1986). Variable Sized Bin Packing, *Siam Journal on Computing*,15 (1) : 222-230
[43] Fontanili, F. (1999). Intégration d'outils de simulation et d'optimisation pour le pilotage d'une ligne d'assemblage multiproduit à transfert asynchrone, *Thèse de doctorat, Université Paris 13*
[44] Fu, R., Ji, T., Yuan, .J, Lin, Y. (2007). Online scheduling in a parallel batch processing system to minimize makespan using restarts. *Theoretical Computer Science*, 374(1-3) : 196-202.
[45] Fu, R., Tian, J., Yuan, J., He, C. (2008). On-line scheduling on a batch machine to minimize makespan with limited restarts. *Operations Research Letters*, 36(2) : 255-258.
[46] Fu, R., Tian, J., Yuan, J. (2008). On-line scheduling on an unbounded parallel batch machine to minimize makespan of two families of jobs. *J. of Sched*, 12(1) : 91-97.
[47] Fu, R., Cheng, T., Ng, C., Yuan, J. (2010). Online scheduling on two parallel-batching machines with limited restarts to minimize the makespan. *Information Processing Letters*, 110(11) : 444-450.
[48] Gfeller, B., Peeters, L., Weber, B., Widmayer, P. (2007). Single machine batch scheduling with release times. *J Comb Optim*, 17(3) : 323-338.
[49] Ghazvini, F.J. and Dupont, L. (1998). Minimizing mean flow times criteria on a single batch processing machine with non-identical job sizes. *International Journal of Production Economics*, 55 (3) : 273-280.
[50] Gilmore, P.C. and Gomory, R.E. (1963). A linear programming approach to the cutting stock problem – part II, *Operations Research*, 11 : 863–888
[51] Graham, R., Lawler, E., Lenstra, J., and Kan, A. (1979). Optimization and Approximation in Deterministic Sequencing and Scheduling: a Survey *Annals of Discrete Mathematics*, 5 : 287-326.
[52] Guinet, A. and Chaabane, S. (2003). Operating theatre planning, *International Journal of Production Economics*, 85 : 69-81
[53] Gupta, A. K., and Sivakumar, A. I. (2006). Optimization of due-date objectives in scheduling semiconductor batch manufacturing. *International Journal of Machine Tools and Manufacture*, 46 (12-13) : 1671-1679.
[54] Hochbaum, D.S. (1997). Approximation Algorithms for NP-Hard Problems, *PWS Publishing Company, Boston, United States*
[55] Hochbaum, D. S., and Landy, D. (1997). Scheduling semiconductor burn-in operations to minimize total flowtime. *Operations research*, 45(6) : 874-885.

[56] Ikura, Y., and M. Gimple. (1986). Efficient scheduling algorithms for a single batch processing machine. *Operations Research Letters*, 5(2) : 61-65.
[57] Johnson, D.S., Demers, A., Ullman, J.D., Garey, M.R. and Graham, R.L. (1974). Worst-Case Performance Bounds for Simple One-Dimensional Packing Algorithms, *SIAM Journal on Computing*, 3 : 299-325
[58] Jolai, F. (2005). Minimizing number of tardy jobs on a batch processing machine with incompatible job families. *European Journal of Operational Research*, 162(1) : 184-190.
[59] Jun, J. B., Jacobson, S. H. and Swisher, J. R. (1999). Application of discrete-event simulation in health care clinics: a survey. *The Journal of the Operational Research Society*, 50 (2) :109-123
[60] Karmarkar, N. and Karp, R.M. (1982). An effcient approximation scheme for the one-dimensional bin-packing problem. In *Proceedings of the 23rd Annual Symposium on Foundations of Computer* Science, 312-320
[61] Kashan, A.H., Karimi, B., and Jolai, F., (2006). Minimizing Makespan on a Single Batch Processing Machine with Non-identical Job Sizes: A Hybrid Genetic Approach. *Evo-COP*, 3906 (2006) : 135-146.
[62] Kashan, A.H., Karimi, B., and Jenabi, M., (2008). A hybrid genetic heuristic for scheduling parallel batch processing machines with arbitrary job sizes. *Computers and Operations Research*, 35 (4) : 1084-1098.
[63] Kashan, A.H., Karimi, B., and Fatemi Ghomi, S.T.M., (2009). A note on minimizing makespan on a single batch processing machine with nonidentical job sizes. *Theoretical Computer Science*, 410 (27-29) : 2754-2758.
[64] Kashan, A.H., Karimi, B., and Jolai, F., (2010). An effective hybrid multi-objective genetic algorithm for bi-criteria scheduling on a single batch processing machine with non-identical job sizes. *Engineering Applications of Artificial Intelligence*, 23 (6) : 911-922.
[65] Kempf, K.G., Uzsoy, R. andWang, C. (1998). Scheduling a single batch processing machine with secondary resource constraints. *Journal of Manufacturing Systems*, 17 (1) : 37-51.
[66] Kergosien, Y., Lenté, C., Piton, D. and Billaut, J.C. (2011). A tabu search heuristic for the dynamic transportation of patients between care units, *European Journal of Operational Research*, 214(2) : 442-452
[67] Koh, S-G., Koo, P-H., Ha, J-W. and Lee, W-S. (2004). Scheduling parallel batch processing machines with arbitrary job sizes and incompatible job families. *International Journal of Production Research*, 42(19) : 4091-4107.
[68] Kumar, M., Damodaran, P. and Srihari, K. (2009). Minimizing makespan in a flow shop with two batch-processing machines using simulated annealing. *Robotics and Computer-Integrated Manufacturing*, 25(3) : 667-679.
[69] Kurz, M. E., and Mason, S. J. (2008). Minimizing total weighted tardiness on a batch processing machine with incompatible job families and job ready times. *International Journal of Production Research*, 46(1) : 131-151.

[70] Lapierre, S. D. and Ruiz., A. B. (2007). Scheduling logistic activities to improve hospital supply systems. *Computers & Operations Research*, 34 : 624-641
[71] La stérilisation en accusation, *Le point*, 30 aout 2002, 1563 : 65-67
[72] Lawler, E.L., and Moore, J.M. (1969) A functional equation and its application to resource allocation and sequencing problems. *Management Science,* 16(1) : 77-84.
[73] Lee, C.C. and Lee, D.T. (1985). A simple on-line bin-packing algorithm. *Journal of the ACM*, 32(3) : 562-572
[74] Lee, C.Y., Uzsoy, R. and Martin Vega, L.A. (1992). Efficient algorithms for scheduling semiconductor burn-in operations. *Operations Research,* 40(4) : 764-775
[75] Lee, C. -Y, and Uzsoy, R. (1999). Minimizing makespan on a single batch processing machine with dynamic job arrivals. *International Journal of Production Research,* 37(1) : 219-236.
[76] Li, C.L., and Lee, C.Y. (1997). Scheduling with agreeable release times and due dates on a batch processing machine. *European Journal of Operational Research,* 96(3) : 564-569.
[77] Li, S., Li, G. and Zhang, S. (2005). Minimizing makespan with release times on identical parallel batching machines. *Discrete Applied Mathematics,* 148(1) : 127-134.
[78] Li, S., Li, G., Wang, X. and Liu, Q., (2005). Minimizing makespan on a single batching machine with release times and non-identical job sizes. *Operations Research Letters*, 33 (2) : 157-164.
[79] Li, L., and F. Qiao. (2008). ACO-based scheduling for a single batch processing machine in semiconductor manufacturing. *4th IEEE Conference on Automation Science and Engineering,* CASE 2008 : 85-90.
[80] Lili, L., and Feng, Z. (2008). Minimizing number of tardy jobs on a batch processing machine with incompatible job families. *Proceedings - ISECS International Colloquium, on Computing, Communication, Control, and Management,* 3 : 277-280.
[81] Li, S., and Yuan, J. (2010). Parallel-machine parallel-batching scheduling with family jobs and release dates to minimize makespan. *Journal of Combinatorial Optimization,* 19(1) : 84-93
[82] Liberatore, M. J. and Nydick, R. L. (2008). The analytic hierarchy process in medical and health care decision making: a literature review, *European Journal of Operational Research*, 189 (1) : 194-207
[83] Liu, Z., and Yu, W. (2000) Scheduling one batch processor subject to job release dates. *Discrete Applied Mathematics*, 105(1-3) : 129-136.
[84] Liu, Z., J. Yuan, and T. C. E. Cheng. (2003). On scheduling an unbounded batch machine. *Operations Research Letters*, 31(1) : 42-48.

[85] Liu, Z., and Edwing Cheng, T. C. (2005). Approximation schemes for minimizing total (weighted) completion time with release dates on a batch machine. *Theoretical Computer Science*, 347(1-2) : 288-298,

[86] Liu, L. L., Ng, C. T. and Cheng, T. C. E. (2007). Scheduling jobs with agreeable processing times and due dates on a single batch processing machine. *Theoretical Computer Science*, 374(1-3) : 159-169.

[87] Liu, L. L., C. T. Ng, and T. C. E. Cheng. (2009). Scheduling jobs with release dates on parallel batch processing machines. *Discrete Applied Mathematics*, 157(8) : 1825-1830.

[88] Liu, M., Xu, Y., Chu, C., Wang, L. (2009). Optimal Semi-online Algorithm for Scheduling on a Batch Processing Machine. *In: Proceedings of the 3rd International Conference on Combinatorial Optimization and Applications. Berlin, Heidelberg, Germany*, p. 346–353.

[89] Liu, P., Lu, X., Fang, Y. (2012). A best possible deterministic on-line algorithm for minimizing makespan on parallel batch machines, *Journal of Scheduling*, 15 (1) : 77-81

[90] Lu, S., Feng, H. and Li, X., (2010). Minimizing the makespan on a single parallel batching machine. *Theoretical Computer Science*, 411 (7-9) : 1140-1145.

[91] Malve, S., and Uzsoy, R. (2007). A genetic algorithm for minimizing maximum lateness on parallel identical batch processing machines with dynamic job arrivals and incompatible job families. *Computers & Operations Research*, 34(10) : 3016-3028.

[92] Marcon, E., Kharraja, S., Smolski, N., Luquet, B. and Viale, J.P. (2003). Determining the number of beds in the postanesthesia care unit: a computer simulation flow approach, *Anesthesia and Analgesia*, 96 :1415-1423

[93] Martello, S. and Toth, P. (1990). Knapsack problems: algorithms and computer implementations. *John Wiley and Sons*, ChichesterNew York.

[94] Mehta, S.V., and Uzsoy, R. (1998). Minimizing total tardiness on a batch processing machine with incompatible job families. *I.I.E. Transaction*, 31(2) : 165-178.

[95] Melouk, S., Damodaran, P. and Chang, P., (2004). Minimizing makespan for single machine batch processing with non-identical job sizes using simulated annealing. *International Journal of Production Economics*, 87 (2) : 141-147.

[96] Mendoza, G.A., Meimban, R.J., Luppold, W.G., Araman, P.A. (1991). Combining simulation and optimization models for hardwood lumber production, *In: Proceedings: The 1991 SAP National Convention, 4-7 august 1991, San Francisco, USA*

[97] Meng J., Lu, X. (2010). The single-machine parallel-batching scheduling problem with family jobs to minimize makespan. *In: Proceedings of the Second International Conference on Communication Systems, Networks and Applications (ICCSNA)*, p. 196-199.

[98] Ministère de l'emploi et de la solidarité, Ministère délégué à la santé. (2001). Bonnes pratiques de pharmacie hospitalière, *Technical report*

[99] Motwani, R. Lecture notes on approximation algorithms, Technical Report published *by Dept. of Computer Science, Stanford University* (1992)

[100] Mönch, L., Balasubramanian, H., Fowler, J.W. and Pfund, M.E. (2005). Heuristic scheduling of jobs on parallel batch machines with incompatible job families and unequal ready times. *Computers & Operations Research*, 32(11) : 2731-2750

[101] Neale, J.J. and Duenyas, I. (2003). Control of a batch processing machine serving compatible job families. *IIE Transactions*, 38 (8) : 699–710.

[102] Ngo Cong, K. (2009). Etude et amélioration de l'organisation de la production de dispositifs médicaux stériles, *Thèse do doctorat à l'Université Joseph Fourrier, Grenoble*

[103] Nong, Q., Cheng, T., Ng, C. (2008). An improved on-line algorithm for scheduling on two unrestrictive parallel batch processing machines. *Operations Research Letters*,36(5) : 584-588.

[104] Nong, Q. Q., Ng, C. T. and Cheng, T. C. E. (2008). The bounded single-machine parallel batching scheduling problem with family jobs and release dates to minimize makespan. *Operations Research Letters*, 36(1) : 61-66.

[105] Ouabiba, M., Mebarki, N. et Castagna, P. (2001). Couplage entre des Méthodes d'optimisation itératives et des modèles de simulation à évenements discrets, *3ème Conférence Francophone de MOdélisation et SIMulation «Conception, Analyse et Gestion des Systèmes Industriels» MOSIM'01 – du 25 au 27 avril 2001 - Troyes (France)*

[106] Ozturk, O., Espinouse, M-L., Di Mascolo, M. and Gouin, A. (2010). Optimizing the makespan of washing operations of medical devices in hospital sterilization services, *IEEE Workshop on Health Care Management (WHCM), pages 1-6, 18-20 Feb. 2010, Venice, Italy*

[107] Ozturk, O., Di Mascolo, M., Espinouse, M-L., and Gouin, A. (2010). Minimizing the sum of job completion times for washing operations in hospital sterilization services, *8th International Conference of Modeling and Simulation (MOSIM'10), pages 1-6,10-12 may 2010, Hammamet, Tunisia*

[108] Ozturk, O., Di Mascolo, M., Espinouse, M-L., and Gouin, A. (2010). Minimisation du temps moyen d'attente des dispositifs médicaux a l'étape de lavage d'un service de stérilisation hospitalier, *5ème Conférence Francophone en Gestion et Ingénierie des SystèmEs Hospitaliers, GISEH'10, pages 1-8, 2-4 septembre 2010, Clermont-Ferrand, France*

[109] Ozturk, O., Di Mascolo, M., Espinouse, M-L., and Gouin, A. (2011). Optimisation du chargement des laveurs dans un service de stérilisation, *4èmes Journées Doctorales/Journées Nationales MACS, pages 1-10, 9-10 june 2011, Marseille – France*

[110] Ozturk, O., Espinouse, M-L., Di Mascolo, M. and Gouin, A. (2011). Makespan minimisation on parallel batch processing machines with non-identical job sizes and release dates, publié en linge dans *International Journal of Production Research* (DOI : 10.1080/00207543.2011.641358)

[111] Parsa, N.R., Karimi, B., Lee, and Kashan, A.H., (2010). A branch and price algorithm to minimize makespan on a single batch processing machine with non-identical job sizes. *Computers and Operations Research*, 37 (10) : 1720-1730.

[112] Pedroso, M. C. and Nakano, D. (2009) Knowledge and information fows in supply chains: a study on pharmaceutical companies. *International Journal of Production Economics*, 122 : 376-384

[113] Potts, C.N., and Kovalyov, M.Y. (2000). Scheduling with batching: A review. *European Journal of Operational Research*, 120 (2) : 228-249.

[114] Poon, C.K., and Zhang, P. (2004). Minimizing makespan in batch machine scheduling. *Algorithmica*, 39(2) : 155-174.

[115] Poon, C.K., Yu, W. (2005). On-Line Scheduling Algorithms for a Batch Machine with Finite Capacity. *J Comb Optim*, 9(2): 167-186.

[116] Poon, C.K., Yu, W. (2005). A Flexible On-line Scheduling Algorithm for Batch Machine with Infinite Capacity. *Ann Oper Res*, 133(1-4) : 175-181.

[117] Poyet, A. (2005). Le dispositif médical : Aspects réglementaires et économiques ; Evaluations sur les dix dernières années. *Thèse à l'Université Claude Bernard – Lyon 1*

[118] Reymondon, F., Di Mascolo, M., Gouin, A., Pellet, B. and Marcon, E. (2008). Etat des lieux des pratiques de stérilisation hospitalière en Rhône-Alpes. *Conférence GISEH'08, Lausanne, 4-6 septembre 2008.*

[119] Reymondon, F., Pellet, B. and Marcon, E. (2008). Optimization of hospital sterilization costs proposing new grouping choices of medical devices into packages. *International Journal of Production Economics,* 112 (2008): 326-335

[120] Ridouard, F., Richard, P., Martineau, P. (2008). On-line scheduling on a batch processing machine with unbounded batch size to minimize the makespan. *European Journal of Operational Research*, 189(3) : 1327-1342.

[121] Rutala, W.A. and Weber, D.J. (2004). Disinfection and sterilization in health care facilities: What clinicians need to know. *Clinical infectious diseases*, 39(5) : 702-709.

[122] Rutala, W.A. and Weber, D.J. (2011). Sterilization, High-Level Disinfection, and Environmental Cleaning *Infectious Disease Clinics of North America*, 25(1) : 45-76

[123] Scheithauer, G. and Terno, J. (1997). Theoretical Investigations on the Modified Integer Round-Up Property for the One-Dimensional Cutting Stock Problem, *Op. Res. Letters,* 20(2) : 93-100

[124] Shmonin, G. and Sebő, A. (2011). Integrality gap for the bin-packing problem, *Rapport technique,Laboratoire G-SCOP*

[125] Shubin, X., and Bean, J. C. (2007). A genetic algorithm for scheduling parallel nonidentical batch processing machines. *Proceedings of the 2007 IEEE Symposium on Computational Intelligence in Scheduling*, 2007 : 143-150.
[126] Standard AFNOR FD S98-135 (2005). Stérilisation des dispositifs médicaux - *Guide pour la maîtrise des traitements appliqués aux dispositifs médicaux réutilisables*
[127] Sung, C.S., and Choung, Y. I. (2000). Minimizing makespan on a single burn-in oven in semiconductor manufacturing. *European Journal of Operational Research*, 120(3): 559-574.
[128] Sung, C. S., Choung, Y. I., Hong, J. M. and Kim, Y. H. (2002). Minimizing makespan on a single burn-in oven with job families and dynamic job arrivals. *Computers & Operations Research*, 29(8) : 995-1007.
[129] Tantchou, J and Grunénais, M-E. (2009). Quand les frontières du stérile et du non-stérile s'évanouissent, *Revue d'anthropologie des connaissances*, 3(3) : 458-484
[130] Tian, J., Fu, R., Yuan, J. (2007). On-line scheduling with delivery time on a single batch machine. *Theoretical Computer Science*, 374(1-3) : 49-57.
[131] Tian, J., Fu, R., Yuan, J. (2009). A best online algorithm for scheduling on two parallel batch machines. *Theoretical Computer Science*, 410(21-23) : 2291-2294.
[132] T'kindt, V. and Billaut, J.C. Multicriteria Scheduling: Theory, Models and Algorithms, *second ed., Springer-Verlag, New York*, 2006
[133] Uzsoy, R., (1994). Scheduling a single batch processing machine with non-identical job sizes. *International Journal of Production Research*, 32 (7) : 1615-1635.
[134] Uzsoy, R. (1995). Scheduling batch processing machines with incompatible job families. *International Journal of Production Research*, 33(10) : 2635-2708.
[135] Uzsoy, R., and Yang, Y. (1997). Minimizing total weighted completion time on a single batch processing machine. *Production and Operations Management*, 6(1) : 57-73.
[136] Valerio de Carvalho, J.M. (2002). LP models for bin packing and cutting stock problems. *European Journal of Operational Research*, 141 (2) : 253-273
[137] Wang, C.S., and Uzsoy, R. (2002). A genetic algorithm to minimize maximum lateness on a batch processing machine. *Computers & Operations Research*, 29(12) : 1621-1640.
[138] Wang, H. and Chou, F. (2010). Solving the parallel batch-processing machines with different release times, job sizes, and capacity limits by metaheuristics. *Expert Systems with Applications: An International Journal*, 37 (2) : 1510-1521.

[139] Webster, S., and Baker, K.R. (1995). Scheduling groups of jobs on a single machine. *Operations Research,* 43(4) : 692-704.
[140] Yongqiang, S. and Enyu, Y. (2005). On-line problems of minimizing makespan on a single batch processing machine with nonidentical job sizes, *Applied Mathematics – A Journal of Chinese Universities*, 20(3) : 297-304, DOI: 10.1007/s11766-005-0005-9
[141] Yuan, J., Li, S., Tian, J., Fu, R. (2007). A best on-line algorithm for the single machine parallel-batch scheduling with restricted delivery times. *J Comb Optim*,17(2) : 206-213.
[142] Yuan, J., Ng, C.T., Cheng, T.C.E. (2011). Best semi-online algorithms for unbounded parallel batch scheduling. *Discrete Appl. Math*, 159 : 838–847.
[143] Yuan, J., Fu, R., Ng, C.T., Cheng, T.C.E. (2011). A best online algorithm for unbounded parallel-batch scheduling with restarts to minimize makespan, *J of Scheduling,* 14(4) : 361-369
[144] Zhang, G., Cai, X., Lee, C.Y. and Wong, C.K., (2001). Minimizing makespan on a single batch processing machine with non-identical job sizes. *Naval Research Logistics*, 48 (3) : 226-240.
[145] Zhang G, Cai X, Wong C. (2001). On-line algorithms for minimizing makespan on batch processing machines. *Naval Research Logistics*, 48(3) : 241-258.
[146] Zhang G, Cai X, Wong CK. (2003). Optimal on-line algorithms for scheduling on parallel batch processing machines. *IIE – Transactions*, 35(2) : 175.
[147] Zhang, Y., Bai, C. and Wang, S. (2005). Duplicating and its Applications in Batch Scheduling, *In: Proceedings of the 5th international symposium on Operations Research and its application, Tibet, China, August 8-13, 2005*
[148] Zhang, Y., Bai, C., Bai, Q. and Xu, J. (2007). Duplicating in batch scheduling. *Journal of Industrial and Management Optimization*, 3(4) : 685-692.
[149] Zhang, Y., Cao, Z. and Bai, Q. (2005). A PTAS for scheduling on agreeable unrelated parallel batch processing machines with dynamic job arrivals. *Lecture Notes in Computer Science*, 3521 : 162-171.
[150] Zhang, X., Zhang, Y., Cao, Z. and Cai, Z. (2007). Approximation schemes for scheduling a batching machine with non-identical job size. *Journal of systems science and complexity*, 20 : 592-600
[151] http://www.afs.asso.fr/, *site web de l'association française de stérilisation*
[152] http://www.legifrance.gouv.fr/, *site web pour le service public de la diffusion du droit*
[153] http://fr.wikipedia.org/wiki/Louis_Pasteur
[154] http://www.sterilisation-hopital.com

~ Annexe A ~

Glossaire des abréviations

~*a*~

Algo. exacte	algorithme exact
Algo. approx.	algorithme d'approximation
APTAS	« asymptotic polynomial time approximation scheme » en anglais, « schéma d'approximation polynomial asymptotique » en français
AFPTAS	« asymptotic fully polynomial approximation scheme » en anglais, « schéma d'approximation entièrement polynomial asymptotique » en français

~*b*~

B & B	algorithme de « séparation et évaluation » (*branch and bound* en anglais)
BF	algorithme best fit
BFM	algorithme best fit modifié

~*c*~

CHPSM	Centre Hospitalier Privé Saint Martin de Caen
CHU	Centre Hospitalier Universitaire

~f~

FF	algorithme first fit
FFD	algorithme first fit decreasing
FFI	algorithme first fit increasing
FFM	algorithme first fit modifié

~n~

NF	algorithme next fit
NFM	algorithme next fit modifié

~p~

PD	programmation dynamique
Perf	performance
PTAS	« polynomial approximation scheme » en anglais, « schéma d'approximation polynomial asymptotique » en français
p-batch	ordonnancement par batch où les tâches d'un batch sont exécutées ensemble (*parallel batching* en anglais)

~s~

s-batch	ordonnancement par batch où les tâches d'un batch sont exécutées une par une (*serial batching* en anglais)

~w~

WF	algorithme worst fit
WFM	algorithme worst fit modifié

~ Annexe B ~

Glossaire des notations utilisées

~*b*~

b (ou B)	capacité de machine

~*c*~

C_{max}	durée totale d'exécution (*makespan* en anglais)
C_j	date de fin d'exécution de la tâche j

~*d*~

d_j	date de fin d'exécution souhaitée de la tâche j
$đ_j$	date de fin d'exécution impérative de la tâche j

~*e*~

ε	un très petit nombre positif

~*f*~

F	famille de tâches
F, p_j *comp*	famille des tâches telle que les tâches d'une même famille ont la même durée d'exécution
$F, setup_j$ *comp*	famille des tâches telle que les tâches d'une même famille ont la même durée de réglage de machine

~*l*~

L_j	retard algébrique de la tâche j ($L_j = C_j - d_j$)
L_{max}	retard algébrique maximal

~*m*~

m	indice des machines
M	nombre de machines

~*n*~

N (également n)	nombre de tâches

~*p*~

p_j	durée d'exécution de la tâche j
$p_j = p$	toutes les tâches ont la même durée d'exécution
$p_j \nearrow d_j$	relation de concordance entre p_j et d_j des tâches
P	machines parallèles identiques

~*r*~

r_j	date de disponibilité de la tâche j (dans notre étude, la date de disponibilité d'une tâche correspond au moment où elle peut être exécutée dans une machine)

~*t*~

T_j	retard vrai de la tâche j ($T_j = \max(0, C_j - d_j)$)
T_{max}	retard vrai maximal

~*u*~

U_j	indicateur de retard de la tâche j ($U_j = 1$ si $C_j > d_j$, 0 sinon)

~*w*~

w_j poids de la tâche *j* (i.e. importance de la tâche *j*)

~*v*~

v_j taille/volume de la tâche *j*

~ Annexe C ~

Liste des notations utilisées dans les tableaux sur la littérature d'ordonnancement

C.1. Notations dans les tableaux 3.3, 3.4 et 3.5

b	nombre maximum de tâches qui peuvent être mises dans un batch dans le cas où les tâches sont de tailles unitaires
ε	un nombre très petit
$đ_j$	date de fin impérative pour les tâches
F	familles de tâches incompatibles
F, p_j *comp.*	familles incompatibles pour les tâches où les tâches d'une même familles ont toutes la même durée d'exécution
mach. Ind.	machines indépendantes (le cas où la durée d'exécution d'une tâche dépend des machines sans qu'il y ait un ratio d'uniformité entre les machines)
$p_{mj} \geq p_{m'j}$	la durée d'exécution des tâches d'indice petit est supérieure ou égale à la durée d'exécution des tâches d'indice grand
$p_j \nearrow d_j$	les durées d'exécution sont en bonne concordance avec les dates dues
$p_j = p$	durées d'exécution égales
$r_j \nearrow d_j$	les dates de disponibilité sont en bonne concordance avec les dates *dues*

$r_j \nearrow p_j \nearrow d_j$	les dates de disponibilité, les durées d'exécution et les dates dues sont en bonne concordance
$p_j \nearrow v_j$	les tailles des tâches sont en bonne concordance avec les dates dues
$p_j = 1$	les durées d'exécution sont unitaires

C.2. Notations dans les tableaux 7.1 et 7.2

F2, p_j comp.	seulement deux familles incompatibles pour les tâches et les tâches d'une même famille ont toutes la meme durée d'exécution
F, $setup_j$, p_j=p comp	familles de tâches incompatibles avec des tâches d'une même famille ayant la même durée d'exécution et la même durée de réglage de machine
*préemption**	contrairement à la préemption classique, si un batch est préempté, tout le travail fait est perdu
prec	contraintes de précédence
p_j=p	durées d'exécution égales
$p_j \geq c p_{j+1}$	*c* étant un nombre positif supérieur à 1
$p_j \nearrow d_j$	les durées d'exécution sont en bonne concordance avec les dates dues
$p_j \in [p, (1 + \varphi)p]$	φ étant égale à $(\sqrt{5}+1)/2$ et *p* ayant une valeur quelconque, toutes les durées d'exécution sont dans l'intervalle $[p, (1 + \varphi)p]$
r_0-r_1	cas particulier où il y a seulement deux dates de disponibilité différentes.
$r_j \nearrow p_j$	les dates de disponibilité et les durées d'exécution sont en bonne concordance

$r^*(t)$	au moment t, la date de disponibilité de la tâche de plus grande durée d'exécution est connue à l'avance

~ Annexe D ~

Liste des nos publications

D.1. Articles dans des journaux internationaux

Ozturk, O., Espinouse, M-L., Di Mascolo, M. and Gouin, A. (2011). Makespan minimisation on parallel batch processing machines with non-identical job sizes and release dates, accepté pour la publication dans *International Journal of Production Research*

(En cours) Onur Ozturk, Marie-Laure Espinouse, Maria Di Mascolo, Alexia Gouin, «Minimizing the predisinfection excess time of medical devices in a hospital sterilization service», *soumission prévue pour IEEE-Transactions on Automation Science and Engineering*

D.2. Articles dans des conférences internationales avec comité de lecture

Ozturk, O., Espinouse, M-L., Di Mascolo, M. and Gouin, A. (2010). Optimizing the makespan of washing operations of medical devices in hospital sterilization services, *IEEE Workshop on Health Care Management (WHCM), 18-20 Feb. 2010, Venice, Italy*

Ozturk, O., Espinouse, M-L., Di Mascolo, M. and Gouin, A. (2010). Makespan minimization with job splitting on a single parallel batching machine for washing operations of hospital sterilization services, *PMS'10, 12th International Workshop on Project Management and Scheduling, 26-28 april 2010, Tours - Loire Valley, France, France*

Ozturk, O., Di Mascolo, M., Espinouse, M-L., and Gouin, A. (2010). Minimizing the sum of job completion times for washing operations in hospital sterilization services, *8th International Conference of Modeling and Simulation - MOSIM'10 - 10-12 may 2010 - Hammamet - Tunisia*

Ozturk, O., Di Mascolo, M., Espinouse, M-L., and Gouin, A. (2010). Minimisation du temps moyen d'attente des dispositifs médicaux a l'étape de

lavage d'un service de stérilisation hospitalier, *5ème Conférence Francophone en Gestion et Ingénierie des SystèmEs Hospitaliers, GISEH'10, 2-4 septembre 2010, Clermont-Ferrand, France*

Ozturk, O., Di Mascolo, M., Espinouse, M-L., and Gouin, A. (2011), Optimisation du chargement des laveurs dans un service de stérilisation, *4èmes Journées Doctorales/Journées Nationales MACS, 9-10 june 2011, Marseille – France*

Ozturk, O., Di Mascolo, M., Espinouse, M-L., and Gouin, A. (2012). A semi-online algorithm for optimizing the pre-disinfection duration of medical devices in a hospital sterilization service, *9th International Conference of Modeling and Simulation - MOSIM'12 – 8-10 june 2012 - Bordeaux - France*

D.3. Articles sans acte dans des conférences internationales sans comité de lecture

Ozturk, O., Espinouse, M-L., Di Mascolo, M. and Gouin, A. (2009). A mixed integer linear programming framework for optimizing the makespan of washing operations in hospital sterilization services, *International Conference on Operational Research Applied to Health Services, ORAHS, 12-17 july 2009, Leuven, Belgique*

Ozturk, O., Di Mascolo, M., Espinouse, M-L., and Gouin, A. (2011), An Online Algorithm for minimizing the Pre-disinfection Excess Time of Medical Devices in Hospital Sterilization Services, *12e congrès annuel de la Société Française de Recherche Opérationnelle et d'Aide à la Décision, 2-4 Mars 2011, Saint-Etienne, France*

Ozturk, O., Sebo, A., Espinouse, M-L., and Di Mascolo, M. (2011). An optimal bin packing algorithm to minimize the number of washing cycles in a hospital sterilization service, *International Conference on Operational Research Applied to Health Services, ORAHS, 24-29 july 2011, Cardiff, United Kingdom*

Résumé : Dans cette thèse, nous nous intéressons au problème de chargement des laveurs dans un service de stérilisation de dispositifs médicaux réutilisables (DMR). Ce problème de chargement des laveurs a été considéré comme un problème d'ordonnancement par batch. Nous présentons, dans un premier temps, des études offline pour lesquelles nous avons développé des algorithmes, exacts et approchés, ainsi que des modèles PLNE pour certains cas particuliers et pour des cas généraux. Nous présentons ensuite des études semi-online et online pour lesquelles nous avons développé des heuristiques. Nous avons également conçu des modèles de simulation afin de tester l'impact de nos heuristiques sur l'ensemble du service de stérilisation. Nous proposons, en dernier lieu, l'implémentation d'une approche de type bin packing pour le cas d'un service de stérilisation externe afin de minimiser le nombre de cycles de lavage lancés.

Mots clé : Service de stérilisation hospitalière, ordonnancement par batch, algorithmes exacts et approchés, heuristiques, PLNE, simulation

Abstract : In this dissertation, we are interested in the problem of loading of washing resources in hospital sterilization services of reusable medical devices (RMD). Through our studies, we modeled the problem of loading of washing resources as a batch scheduling problem. We began our research with offline studies for whom exact algorithms are proposed for some special cases and then heuristics, approximation algorithms and linear models are proposed for general cases. Afterwards, we continued our research with semi-online and online studies for whom heuristics are developed. We inserted these heuristics also in simulation models in order to test their efficiency on the whole sterilization service. We finished our studies with a final work aiming at minimizing the number of washing cycles for an external sterilization service where we adopted a bin packing approach.

Key words: Hospital sterilization service, batch scheduling, exact and approximation algorithms, heuristics, MILP, simulation