Méthodes et outils pour l'ordonnancement d'ateliers avec prise en compte des contraintes additionnelles : énergétiques et environnementales

Damien Lamy

ED SPIC n° d'ordre : 826

Université Clermont Auvergne

ECOLE DOCTORALE DES

SCIENCES POUR L'INGENIEUR DE CLERMONT-FERRAND

THESE

Présentée par

Damien Lamy

Ingénieur en Informatique, diplômé de l'ISIMA (2014)

Pour obtenir le grade de

Docteur d'Université

Spécialité : INFORMATIQUE

Méthodes et outils pour l'ordonnancement d'ateliers avec prise en compte des contraintes additionnelles : énergétiques et environnementales

Soutenue publiquement le 4 décembre 2017 devant le jury :

Rapporteurs :	Christelle GUERET	Professeur	Université d'Angers
	Imed KACEM	Professeur	Université de Lorraine
Examinateurs :	Marie-José HUGUET	Professeur	INSA de Toulouse
	Pierre CASTAGNA	Professeur	Université de Nantes
	Damien TRENTESAUX	Professeur	Université de Valenciennes et du Hainaut-Cambrésis
Directeurs de thèse	Sylverin KEMMOE	Maître de conférences	Université Clermont Auvergne
	Nikolay TCHERNEV	Professeur	Université Clermont Auvergne
	Alain QUILLIOT	Professeur	Université Clermont Auvergne

Remerciements

Je tiens à remercier chaleureusement Nikolay Tchernev, dont la façon de diriger cette thèse restera pour moi un exemple à reproduire lorsque j'en aurai l'occasion. Je le remercie infiniment pour son implication dans ces travaux de recherche et pour m'avoir laissé choisir les directions dans lesquelles je souhaitais aller. Quand bien même je préférais « manger du sel », ou ne « pas regarder la bourse posée sur le pont » il m'a toujours soutenu dans mes choix et n'a jamais abandonné d'essayer de me faire entendre raison ; je pense qu'une grande partie de ses enseignements a porté ses fruits et m'a permis de m'épanouir durant ces trois années.

Je remercie également Sylverin Kemmoé-Tchomté pour son encadrement et son apport sur des questions techniques mais aussi humaines, qui, au final, constituent le fondement d'une thèse. Je me souviendrai toujours de ces moments à parler de tout : politique, économie et autres, car cette thèse ne se serait pas déroulée de la même manière sans ces conversations qui je l'espère ne s'arrêteront pas de sitôt.

Je remercie chaleureusement les membres du jury pour leur relecture attentive. En particulier, j'adresse mes remerciements à Christelle Guéret, Professeure à l'université d'Angers, pour sa lecture minutieuse du manuscrit et à Imed Kacem, Professeur à l'université de Metz, pour les suggestions concernant les suites à donner à ces travaux, notamment concernant le Job-shop Flexible. Je remercie également Damien Trentesaux et Marie-José Huguet pour avoir accepté de participer au Jury. Enfin, je remercie Pierre Castagna avec qui j'ai eu l'occasion d'échanger à de nombreuses reprises au cours du projet ECOTHER.

Je tiens à remercier Alain Quilliot qui m'a permis d'effectuer cette thèse au LIMOS ainsi que Philippe Mahey qui m'a fait confiance à deux reprises : lors de mon entrée à l'ISIMA, et ensuite durant le master recherche. Mes remerciements vont aussi à Philippe Lacomme avec qui j'ai pu travailler sur certains sujets de recherche et qui m'a permis d'encadrer des projets d'étudiants à l'ISIMA.

Je remercie aussi Pierre Féniès, pour m'avoir fait découvrir le monde de la recherche et donné envie de faire cette thèse, ainsi que pour m'avoir encouragé à postuler sur un poste d'ATER au sein de l'Université Paris Nanterre.

Je remercie mes amis et collègues de travail : Benjamin, Benjamin, Benjamin (Eh oui, ils sont nombreux), Maxime, Matthieu, Rafael (vous êtes nombreux …), Raksmey, Marina, Libo, David, Cédric, Arnaud, Meriam (j'arrête là, vous êtes vraiment trop nombreux) pour les discussions riches d'enseignements que nous avons eues ensemble. Tous m'ont été d'une grande aide, tant scientifique que professionnelle et personnelle.

Je remercie les femmes de ma vie, qui m'ont supporté dans ce parcours et ceux qui ont précédé. Leur soutien moral indéfectible et leur implication dans ce projet de vie qu'est la thèse - je pense notamment aux diverses corrections apportées aux articles et à ce manuscrit - sont inestimables.

Enfin, j'ai une pensée particulière pour mon père qui, bien que disparu, a pour moi en ce moment ce petit sourire espiègle qui lui était propre ; je lui dédie ce modeste travail.

Résumé

Ce travail de doctorat aborde trois thématiques: (i) l'ordonnancement des systèmes de production à cheminements multiples et plus particulièrement le Job-shop soumis à un seuil de consommation énergétique ; (ii) la résolution d'un problème d'ordonnancement et d'affectation dans le contexte d'un système flexible de production sous la forme d'un Job-shop Flexible ; (iii) les méthodes de couplage entre la simulation et l'optimisation dans le cadre des problèmes de Job-shop avec incertitude. Différentes approches de résolutions sont appliquées pour chaque problème : une formalisation mathématique est proposée ainsi que plusieurs métaheuristiques (GRASP×ELS, VNS, MA, NSGA-II hybride et GRASP×ELS itéré) pour le Job-shop avec contrainte énergétique. Une extension du GRASP×ELS, notée GRASP-mELS, est ensuite proposée pour résoudre un problème de Job-shop Flexible ; différents systèmes de voisinages utilisés lors des phases de diversification et d'intensification des solutions sont également présentés. Les résultats montrent que les performances du GRASP-mELS sont comparables à celles de la littérature à la fois en terme de qualité et de temps de calcul. La dernière thématique concerne les méthodes de couplage entre optimisation et simulation avec deux problèmes étudiés : 1) un Job-shop Stochastique et 2) un Job-shop Flexible Réactif. Les méthodes de résolution reposent sur des métaheuristiques et sur le langage de simulation SIMAN intégré dans l'environnement ARENA. Les résultats montrent que les deux approches permettent de mieux prendre en compte les aspects aléatoires liés à la réalité des systèmes de production.

Mots-clés : Recherche opérationnelle, ordonnancement, métaheuristiques, heuristiques, simulation, consommation énergétique.

Abstract

This doctoral work addresses three themes: (i) the scheduling of multi-path production systems and more specifically the Job-shop subjected to a power threshold; (ii) the resolution of a scheduling and assignment problem in the context of a flexible production system modelled as a Flexible Job-shop; (iii) the coupling methods between simulation and optimisation in the context of Job-shop problems with uncertainty. Different resolution approaches are applied for each problem: a mathematical formalisation is proposed as well as several metaheuristics (GRASP×ELS, VNS, MA, hybrid NSGA-II and iterated GRASP×ELS) for the Job-shop with power requirements. An extension of the GRASP×ELS, denoted GRASP-mELS, is then proposed to solve a Flexible Job-shop problem; different neighbourhood systems used during the diversification and intensification phases of solutions are also presented. The results show that the performances of the GRASP-mELS are comparable to the methods presented in the literature both in terms of quality of solutions and computation time. The last topic concerns the coupling methods between optimisation and simulation with two problems: 1) a Stochastic Job-shop and 2) a Reactive Flexible Job-shop. The resolution methods are based on metaheuristics and the SIMAN simulation language integrated in the ARENA environment. The results show that both approaches allow to better take into account the random aspects related to the reality of production systems.

Keywords: Operational research, scheduling, metaheuristics, heuristics, simulation, energy consumption

Sommaire

Introduction générale

L'optimisation des systèmes de production représente un enjeu important pour les industriels. Parmi les problèmes couramment abordés dans la littérature, les tournées de véhicules et l'ordonnancement sont au cœur de nombreux sujets de recherche. Dans cette thèse nous nous concentrons particulièrement sur les problèmes d'ordonnancement des systèmes de production industriels. Ces problèmes consistent, de manière générale, à définir un calendrier de passage des opérations au sein d'un système de production. Celui-ci peut être sensiblement différent d'un domaine d'activité à un autre ; ainsi, la définition d'un planning pour un bloc opératoire (système de production de services) fait partie de ces problèmes au même titre que la planification d'opérations ou de tâches à effectuer par des machines (systèmes de production de biens).

De manière générale, la finalité derrière la résolution de tels problèmes consiste à optimiser des critères de performances qui peuvent être de différentes natures en fonction des objectifs à atteindre : maximiser la productivité d'un atelier, réduire les retards par rapport à des dates d'échéances, et bien d'autres. Depuis quelques années, d'autres objectifs ont également émergé avec la thématique de l'industrie 4.0, tels que la réduction de la consommation énergétique, la diminution des coûts en profitant de tarifs négociés avec les fournisseurs d'électricité ou la considération de sources d'énergies renouvelables qui peuvent affecter la capacité d'un système de production à fonctionner correctement (chutes de puissance, …). En plus de ces objectifs, certaines contraintes sont à respecter telles qu'empêcher la trésorerie de devenir négative, ou ne pas dépasser un seuil de puissance autorisé. Si l'on considère le nombre de produits à réaliser sur le système de production, il est bien souvent très difficile, voire impossible, de trouver une solution exacte à ce type de problèmes. En effet, les problèmes d'ordonnancement sont des problèmes combinatoires. Evaluer la qualité de toutes les solutions n'est donc généralement pas envisageable, sans compter le fait que les systèmes de production sont soumis à des aléas et qu'une solution optimale risque de ne pas être applicable entièrement. Face à ces problématiques qui concernent la difficulté des problèmes et les techniques qui peuvent être mises en œuvre pour leur résolution, de nombreuses techniques approchées sont décrites dans la littérature. Dans cette thèse, les travaux de recherche que nous avons menés sont principalement orientés vers ces méthodes de résolution approchées (heuristiques, métaheuristiques). S'il n'est généralement pas possible d'assurer que les solutions retournées par une métaheuristique sont optimales, elles permettent d'atteindre des solutions de bonne qualité en des temps de calcul plus compétitifs que des approches exactes. Ces méthodes ont été décrites et appliquées dans les cinq chapitres qui composent ce manuscrit.

Le premier chapitre vise à définir les bases nécessaires à la compréhension de cette thèse. Il consiste en une définition des systèmes de production, ainsi qu'en une présentation des différents problèmes classiques abordés dans la littérature. La notion de modèle et les formalismes de modélisation sont également introduits car ils sont au cœur de l'ensemble des problèmes traités dans la littérature, et constamment utilisés dans cette thèse. Le processus général d'optimisation mis en œuvre est également présenté. La dernière partie de ce chapitre introduit les notions de problèmes à un ou plusieurs objectifs, les différentes méthodes de résolutions approchées et les principes d'amélioration des solutions (voisinages et recherche locale). Des métaheuristiques classiques de la littérature sont exposées, qu'elles soient orientées mono ou multiobjectifs. Enfin, différentes mesures de qualité sont présentées pour évaluer les performances des algorithmes développés dans cette thèse.

Le second chapitre présente un problème d'optimisation d'atelier à cheminements multiples : le Job-shop. Les techniques couramment appliquées dans la littérature sont exposées. La formalisation mathématique et un modèle linéaire du problème sont introduits. Les notions de

classe d'ordonnancement et de codage des solutions sont données. Ensuite, l'ensemble des outils généralement utilisés dans la littérature sur le problème étudié sont présentés : la représentation sous forme de graphe, l'évaluation d'une solution, les notions de chemin critique et de voisinages sont abordées. Un aperçu de quelques méthodes efficaces reposant sur ces outils est donné. Par la suite une métaheuristique (GRASP×ELS) est proposée pour la résolution du problème. Elle intègre différentes briques algorithmiques : une heuristique de construction randomisée, une procédure de recherche locale permettant d'améliorer la qualité des solutions, et une procédure d'exploration de l'espace basée sur la création de voisins. Les expérimentations numériques qui ont été conduites montrent que le GRASP×ELS est une méthode pertinente pour la résolution du problème et permet d'obtenir des résultats proches de ceux obtenus par les meilleures méthodes publiées.

Le troisième chapitre est au cœur de la problématique abordée dans cette thèse, à savoir la proposition de méthodes de résolution pour les problèmes d'ordonnancement avec contraintes énergétiques. Deux problèmes d'ordonnancement sont introduits. Le premier est mono-objectif : le Job-shop avec seuil de puissance variable. Ce problème consiste à définir des solutions dans le cadre d'un système de production où la puissance instantanée autorisée peut varier au fil du temps. Cette contrainte permet de modéliser à la fois la puissance contractualisée, qui peut être variable, mais aussi les chutes de puissance qui peuvent être à prendre en compte dans le contexte d'un environnement faisant intervenir des ressources énergétiques renouvelables tel que le solaire. Dans le cadre d'une météo défavorable, il faut alors être en mesure de continuer la production tout en prenant en compte cette contrainte. Différentes métaheuristiques sont proposées, dont une reposant sur un GRASP×ELS et aboutissant à des résultats de meilleure qualité. Le second problème prend deux objectifs en compte : le temps total de traitement et la puissance maximale autorisée en entrée du système de production. Il s'agit donc de définir un front de Pareto des solutions envisageables. Deux métaheuristiques sont proposées : un GRASP×ELS adapté pour le problème et un NSGA-II hybride. Les résultats montrent la complémentarité des deux approches, la métaheuristique GRASP proposant de meilleures solutions lorsque le seuil de puissance est élevé, tandis que la métaheuristique basée sur le NSGA-II présente en moyenne de meilleurs résultats sur l'ensemble du front tout en étant moins coûteuse en temps de traitement.

Le quatrième chapitre traite du problème de Job-shop Flexible. Il est plus difficile que le problème de Job-shop car il fait intervenir un problème d'affectation en plus du problème d'ordonnancement. La formalisation mathématique du problème est proposée. Une représentation pour l'affectation est introduite. Des outils propres au problème de Job-shop Flexible sont présentés, tels que la représentation sous forme de graphe et un système de voisinage. Un ensemble de méthodes actuelles et performantes est présenté. Afin de résoudre le problème, une métaheuristique est proposée. Elle repose sur un GRASP-mELS (multi-level ELS), qui apporte une meilleure exploration du voisinage que le GRASP×ELS classique. Une heuristique de construction adaptée au problème est introduite ainsi qu'une recherche locale et les différents voisinages utilisés. Les expérimentations numériques montrent que la métaheuristique développée permet l'obtention de résultats de bonne qualité et comparables à ceux de la littérature.

Le cinquième et dernier chapitre aborde les notions de couplage entre optimisation et simulation. Deux problèmes sont abordés. Le premier est un problème de Job-shop stochastique. La procédure de résolution proposée repose sur un GRASP×ELS qui utilise un modèle de simulation basé sur le langage SIMAN. La métaheuristique vise à explorer les solutions de l'espace déterministe et la simulation évalue la robustesse des solutions obtenues si des temps de traitement suivant une loi Normale sont appliqués. Les résultats montrent que les solutions ne présentent pas toutes le même intérêt lorsque des durées aléatoires sont considérées. Si les solutions optimales (déterministes) paraissent être de bonnes solutions après simulation, certaines solutions optimales sont meilleures que d'autres ce qui permet de conclure que la structure des ordonnancements a un

impact sur leur robustesse. Aussi, certaines solutions optimales sont de moins bonne qualité après simulation que des solutions dont le temps de traitement est supérieur. La deuxième partie de ce chapitre aborde une autre approche dans le cadre d'un Job-shop Flexible réactif. Dans ce problème, un modèle de simulation fait appel à une métaheuristique afin de proposer un réordonnancement des opérations lorsque de nouvelles entités entrent dans le système. L'approche est comparée à une méthode faisant intervenir des règles de gestion dynamiques. Les résultats montrent que, si l'utilisation d'une métaheuristique dans un contexte réactif est intéressante, les temps de calcul observés sont bien supérieurs à ceux d'une approche reposant sur des règles de gestion, avec des différences dans la qualité des solutions non significatives.

Pour terminer, une conclusion revient sur l'ensemble des travaux réalisés et présente un ensemble de perspectives de recherche.

Chapitre I : Présentation de la problématique

Ce chapitre présente la problématique des travaux de recherche réalisés durant ce doctorat, à savoir l'optimisation pour la planification des systèmes de production industrielle.

Sommaire

Liste des figures

Liste des algorithmes

1. Introduction

Ce chapitre est dédié à la présentation de la problématique étudiée dans cette thèse : l'optimisation pour la planification des systèmes de production. La complexité des problèmes présentés réside dans l'utilisation de domaines complémentaires : la connaissance des systèmes de production sans laquelle aucune étude n'est réellement possible, la modélisation dont la complexité principale est de délimiter le périmètre du modèle, et l'optimisation qui a trait à la résolution des problèmes modélisés.

La connaissance des systèmes de production est primordiale car elle permet de faciliter les échanges avec toute personne intervenant sur le système, que ce soient des opérateurs sur une chaîne de production, ou les experts du système. Ces derniers sont des interlocuteurs privilégiés dans les phases d'analyse, de spécifications, de validation et de conception d'un outil d'optimisation car ce sont eux qui connaissent les besoins de l'entreprise et qui ont l'expérience ou bien du système existant, ou bien du système qui doit être réalisé, et par conséquent l'environnement dans lequel le système va évoluer. C'est pour cela que la connaissance des systèmes de production permet d'instaurer un cadre lors des échanges et permet d'éviter tout problème de communication, tout en permettant d'orienter les échanges vers les questions pertinentes et de filtrer les informations superflues. Ceci est particulièrement utile pendant les phases d'analyse et de spécification, car celles-ci sont l'occasion de détecter de nouveaux problèmes et d'exprimer les intérêts, qui peuvent être contradictoires, des différents interlocuteurs.

La modélisation a pour objectif d'appréhender la complexité des systèmes de production. Elle permet de les représenter selon plusieurs critères afin de répondre aux attentes du plus grand nombre d'experts impliqués sur le projet industriel. Le but est de faire apparaître les informations jugées pertinentes lors des phases d'analyse, et de les structurer pour identifier les règles de fonctionnement du système de production. La principale difficulté de cette étape consiste à bien délimiter le périmètre du modèle. En effet, un modèle trop peu détaillé peut conduire à l'échec du projet industriel car il risque d'omettre des informations importantes, et la solution associée au problème modélisé risque d'être inapplicable. A l'inverse, un modèle prenant en compte des contraintes dont l'intérêt, dans le cadre de la résolution de la problématique posée, est limité, risque de le rendre trop complexe pour être résolu efficacement. Il faut donc trouver le bon niveau de granularité du modèle proposé.

Dans le cadre de cette thèse, les problèmes étudiés sont des problèmes d'ordonnancement. Pour la plupart, ces derniers se ramènent à des problèmes d'optimisation dont l'objectif est d'améliorer les performances, voire de déterminer les performances optimales d'un système. C'est pourquoi nous nous intéressons aux techniques permettant la résolution des problèmes de planification/d'ordonnancement. Lorsqu'il n'est pas possible de déterminer une solution optimale à un problème d'optimisation, il faut pouvoir proposer une solution de bonne qualité (pas forcément optimale). Cette notion de « qualité » est un point à définir avec les experts du système. En sus, le modèle d'optimisation peut ne pas prendre en compte certains aspects inhérents aux systèmes de production industriels, que peuvent être les pannes, ou de nouveaux produits à fabriquer. Lorsque la solution initialement prévue ne peut être entièrement mise en œuvre, une modification de la solution est à prévoir, ou alors la redéfinition d'une nouvelle solution. Ces aléas qui conduisent à la modification de la solution initiale peuvent avoir été omis pour faciliter la démarche de résolution du problème, et être traités manuellement, ou automatiquement ensuite. Suivant les cas, différents types de méthodes d'optimisation sont appliqués.

Ce chapitre est structuré en cinq parties. Dans la première partie (section 2), les systèmes de production sont présentés. Après une description générale, les trois classes de problèmes de planification des systèmes de production sont introduites, ainsi qu'un exemple de système de production rencontré.

La seconde partie (section 3) contient une présentation des principaux modèles théoriques pour l'ordonnancement des systèmes de production. Outre les principaux problèmes traités dans la littérature, quelques extensions majeures sont aussi présentées.

La troisième partie (section 4) met en évidence la démarche de modélisation des systèmes de production pour l'aide à la décision. Cette démarche consiste à séparer les solutions partielles et complètes, ainsi que les modules d'évaluation des performances et d'optimisation.

Dans la quatrième partie (section 5), les méthodes de résolution qui peuvent être appliquées aux problèmes traités sont décrites. Une attention particulière est portée aux métaheuristiques permettant de résoudre des problèmes mono-objectifs aussi bien que multiobjectifs. Des critères de performance pour la comparaison des métaheuristiques sont également donnés.

Pour conclure, la cinquième partie récapitule les différents paragraphes ainsi que le contexte de la thèse.

2. Les systèmes de production

2.1. Présentation

Un système de production est un ensemble d'unités, matérielles ou humaines, organisées dans le but de produire des biens ou des services. Un tel système est soumis à une charge qui correspond aux différents biens ou services qui doivent être produits, i.e. la demande. Il est également soumis à une capacité, qui représente la quantité de biens qui peuvent être réalisés sur une période donnée. Une ou plusieurs ressources peuvent être utilisées afin de répondre à la demande. Les systèmes de production étant des systèmes réels, ils ne peuvent donc pas répondre à une charge illimitée de travail. En effet, ils sont contraints par la présence de ressources sans lesquelles la production ne peut avoir lieu. Ces ressources sont présentes en quantités déterminées : des machines, des opérateurs, des matières premières renouvelables ou non, des emplacements de stockage. Le nombre de ressources et leur type dépend intrinsèquement du système de production étudié. On qualifie ces systèmes de "système de production à partage de ressources".

Chaque système de production a pour objectif de réaliser des produits. Chaque produit qui requiert des ressources appartenant au système de production dans son ensemble, peut être fabriqué de manière individuelle ou par lot. Par la suite, dans le contexte de la terminologie des problèmes théoriques d'ordonnancement, chaque pièce (ou lot de pièces) qui sera ordonnancé, est appelé *job*. Un job est une entité qui va suivre la réalisation d'un produit au cours de toutes les étapes de sa fabrication. Chaque job a une gamme opératoire qui consiste en une suite d'opérations correspondant aux traitements nécessaires à effectuer pour arriver à un produit fini. Généralement, une gamme est associée à un type particulier de produits, et donc de jobs, et plusieurs jobs peuvent donc avoir une gamme identique. La gamme décrit donc les opérations à réaliser, et stipule :

- L'ordre dans lequel les opérations sont réalisées (en général) ;
- La ou les machines qui sont affectées à la réalisation de cette opération ;

- La durée nécessaire à la réalisation de chaque opération ;
- Les éventuelles ressources additionnelles consommées (type de la ressource et quantité requise) ;
- Les éventuelles contraintes qui peuvent exister entre opérations de la gamme (durée autorisée entre l'exécution de deux opérations, etc…).

C'est par la réalisation de toutes les opérations appartenant à la gamme opératoire d'un job que le système produit. En fonction du système de production, ces opérations peuvent être très différentes, tous comme les entités intervenant dans leur réalisation. Il peut y avoir des opérations de transformation (chariotage, fraisage, …), de création directe (injection, ...), d'assemblage, ou de transport... Utiliser une terminologie générale pour représenter toutes les opérations intervenant sur les systèmes de production semble donc compromis en raison du vocabulaire utilisé qui est souvent dédié au système de production étudié, et donc peu générique. Par exemple, le terme « machine » utilisé de manière classique dans un système de production de biens, peut être assimilé à un chirurgien dans un système hospitalier, à un processeur dans le domaine de l'informatique, ou encore à un caissier pour une industrie orientée services. Les opérations effectuées sont également différentes avec par exemple du fraisage dans le cadre d'un système industriel, un acte chirurgical pour le système hospitalier, ou des calculs pour un processeur. Dans la suite de cette thèse, la terminologie relevant des systèmes industriels de production est principalement utilisée.

Les systèmes évoqués dans le paragraphe précédent ont été conçus par l'homme. Cependant, la nature de ces systèmes est bien plus complexe à appréhender que ce que l'on pourrait supposer de prime abord. Preuve en est la difficulté d'obtenir dans la plupart des situations des solutions optimales de manière manuelle. S'il peut être possible d'obtenir des solutions de bonne qualité dans certains cas et/ou sur une sous-partie du problème étudié, la résolution du problème dans sa globalité s'avère souvent irréalisable en un temps raisonnable pour une personne seule. La raison de cette difficulté est en grande partie liée à la taille des problèmes traités et aux interactions complexes qui lient les entités du système de production. De plus, si la résolution de problèmes déterministes (où toutes les données du problème sont connues à l'avance) est effectivement difficile, les systèmes de production sont soumis à des aléas de toutes sortes. On parle alors de problèmes stochastiques où il est encore plus difficile de trouver des solutions de bonnes qualités du fait de l'aspect incertain du problème.

La gestion des systèmes de production relève de trois problématiques : leur conception, leur planification et leur contrôle. La conception consiste à définir les caractéristiques futures d'un système de production. Dans cette phase particulière, les questions concernant le dimensionnement du système, les machines utilisées, ou encore le routage des entités dans le système de production sont abordées. La planification a trait à la détermination des différentes ressources utilisées par les opérations à réaliser ainsi qu'à définir la façon et les dates auxquelles ces opérations doivent être exécutées. Enfin, le contrôle consiste à corriger les éventuelles perturbations inhérentes à la réalité des systèmes de production. Ces perturbations peuvent être des pannes, de nouveaux ordres de fabrications à réaliser, ou de manière plus générale, tout événement aléatoire qui entrave la réalisation du planning déterminé. Ces trois étapes-clés du fonctionnement des systèmes de production sont exécutées consécutivement et en fonction de la fréquence des problèmes rencontrés durant la partie 'contrôle', une nouvelle phase de conception peut intervenir.

Les techniques d'optimisation peuvent être appliquées dans toutes les phases présentées précédemment. Dans ce travail de doctorat sont principalement étudiés les problèmes

d'ordonnancement des systèmes de production, et donc la partie planification des étapes mentionnées ci-dessus. La planification peut être appliquée aux problématiques rencontrées et ce à plusieurs niveaux en fonction de l'horizon de temps choisi. Les différents niveaux de planification sont décrits dans le paragraphe suivant.

2.2. Niveaux de planification

Planifier revient à déterminer l'ordre de traitement d'un ensemble d'opérations, et à affecter à ces opérations les ressources nécessaires à leur réalisation. Généralement, trois niveaux de planification sont recensés en fonction de l'horizon de temps souhaité : long terme (stratégique), moyen terme (tactique) et court terme (opérationnel) (Anthony, 1965; Van Looveren et al., 1986). Ces trois niveaux de planification ont des objectifs différents mais participent à l'optimisation d'une entreprise dans sa globalité.

Les décisions stratégiques sont prises sur un horizon de temps important, plus de deux ans en général, en fonction du système de production. Dans ce niveau de planification, les problèmes traités représentent des enjeux importants pour une entreprise tels que la décision de produire de nouvelles références, de construire une nouvelle usine, de modifier la structure d'une ligne de production, d'acquérir des équipements onéreux...

Les décisions tactiques sont prises sur un horizon de temps moindre, entre six et dix-huit mois en général en fonction du système de production. Ce niveau de planification a trait à la résolution de problèmes de distribution ou d'approvisionnement en matières premières. De manière générale la planification tactique met en œuvre des solutions afin de faire respecter les objectifs pris lors de la planification stratégique.

Les décisions opérationnelles sont prises sur un horizon de temps limité, de un à plusieurs jours en fonction du système de production. A ce niveau, la planification opérationnelle concerne directement le pilotage et le contrôle du système de production en intégrant les problématiques liées à la fabrication des pièces. Parmi les problèmes rencontrés peuvent se trouver des plans de transport, la définition des tailles de lots (bien que ce point puisse également être pris en compte au niveau tactique), la gestion des ressources humaines, l'ordonnancement d'atelier …

Dans tous les cas, les décisions prises dans les niveaux de planification évoqués ci-dessus sont des décisions prévisionnelles liées aux commandes à venir ou en cours et n'intègrent généralement pas les éventuels événements aléatoires qui peuvent altérer le bon fonctionnement du système de production dans son ensemble. Parmi les difficultés qui peuvent être rencontrées, les retards, les défauts de fabrication, la priorité donnée à un nouveau client sur un ancien, sont autant de problèmes qui peuvent avoir une incidence sur les objectifs fixés dans chaque niveau de planification. Du fait de ces événements aléatoires, le niveau de finesse de la modélisation va fortement dépendre de l'horizon de planification et des objectifs de l'étude. Cependant, il est important de noter que les outils développés pour un niveau de planification peuvent être adaptés et utilisés dans un niveau de planification supérieur.

Cette thèse traite principalement de la planification opérationnelle des systèmes de production et plus particulièrement de l'ordonnancement. Du fait de la présence d'événements aléatoires comme ceux mentionnés ci-dessus, il est important de définir des méthodes d'optimisation rapides qui peuvent être utilisées afin de réorienter les décisions prises pour qu'elles restent applicables à l'environnement de production actuel s'il a changé.

Dans le paragraphe suivant, un système de production est présenté afin d'illustrer nos propos et justifier le choix des modèles théoriques étudiés dans cette thèse.

2.3. Exemple d'un système de production : projet ECOTHER

La problématique présentée ici est liée au projet ECOTHER, projet ayant pour objectif la réduction des consommations énergétiques dans les systèmes de production de l'industrie du caoutchouc. Une partie du projet porte sur l'ordonnancement et l'évaluation de la performance. Cette partie est conjointement étudiée avec le laboratoire IRCCyN (Institut de Recherche en Communications et Cybernétique de Nantes), afin de proposer un outil décisionnel couplant optimisation et simulation. Le système de production a été modélisé mathématiquement et un outil décisionnel a été développé et évalué sur un cas d'étude, afin de proposer un outil d'aide à la décision pour la planification et l'ordonnancement. La description suivante met en évidence certains modèles théoriques sous-jacents qui sont plus amplement détaillés dans la partie suivante.

Parmi les entreprises parties prenantes, SACRED est une entreprise réalisant des pièces en polymères. Parmi les produits réalisés, des mélanges, de nombreuses pièces à destination de l'industrie automobile, de l'industrie électrique (pièces isolantes), mais aussi des semelles de chaussures. Le site principal, implanté à Saint Lubin des Joncherets, dispose de machines variées permettant la réalisation de différents produits. Les différentes pièces nécessitent des procédés de fabrication tels que l'injection, l'extrusion ou le mélangeage. Un atelier faisant intervenir ces différents procédés a été étudié.

L'atelier de mélangeage peut comprendre de un à plusieurs silos. Les mélanges peuvent subir jusqu'à deux traitements. Le premier est effectué au sein d'un mélangeur dit « interne », le second étant traité sur un mélangeur « externe ». Les durées de traitement varient en fonction du mélange réalisé, et des opérateurs spécialisés sont présents lors des opérations de mélangeage. Il ne peut y avoir d'attente entre les traitements d'un job s'il doit passer par un mélangeur interne puis un mélangeur externe. Les mélanges réalisés peuvent être vendus à un client, ou utilisés en tant que matière première dans les ateliers d'injection et d'extrusion.

En plus des mélanges, des inserts peuvent être requis pour la réalisation de certaines pièces. L'atelier de création d'inserts peut être composé de plusieurs machines. Les durées nécessaires pour la création des inserts varient en fonction des inserts réalisés. Chaque insert subit plusieurs traitements en fonction de la forme désirée. Des opérateurs spécialisés sont affectés à l'atelier d'inserts.

L'atelier d'injection consiste en un ensemble de machines (des presses) et de moules. Les moules permettent de réaliser une à plusieurs pièces d'un type de produit donné, en fonction du nombre d'empreintes du moule. Un moule ne peut être monté que sur une presse à la fois, et une presse ne peut supporter qu'un moule à la fois. Chaque moule peut être monté sur un ensemble de presses compatibles. Un moule nécessite un opérateur de montage afin d'être positionné sur une presse. S'ensuit une étape de préchauffage du moule, puis une étape de fabrication des pièces. En fonction des pièces réalisées, un mélange spécifique et des inserts sont nécessaires. L'étape de fabrication requiert également un opérateur spécialisé.

L'atelier d'extrusion consiste en une ou plusieurs lignes de production composées de différentes machines, principalement des fours, qui cuisent la ou les matières sortant de l'extrudeuse. Une ligne d'extrusion ne peut traiter qu'un type de job à la fois. Une étape de préparation de la ligne et de préchauffage des fours à la bonne température est requise. En fonction

des pièces réalisées, un ou plusieurs mélanges spécifiques sont nécessaires. L'étape de fabrication est encadrée par un opérateur spécialisé.

Une fois que les produits sont fabriqués, ils sont traités par un atelier de packaging. Cet atelier consiste en plusieurs postes. Chaque produit passe sur certains de ces postes en fonction des étapes requises.

On s'intéresse au problème de planification à court terme d'un atelier intégrant mélangeage, création d'inserts, injection, extrusion et packaging. Actuellement, ce problème est résolu manuellement par le responsable de la production. Chaque planning est réalisé sur un horizon d'une semaine. Nonobstant la difficulté d'ordonnancer finement une production répartie en différents pôles, l'entreprise souhaite optimiser sa consommation énergétique, et plus précisément limiter les pics de puissance rencontrés. En effet, les différents ateliers sont gérés simultanément dans l'objectif de répondre au mieux aux attentes des clients. Ceci sous-entend que toutes les machines peuvent être démarrées en même temps, et elles sont relativement énergivores. Pour éviter tout surcoût lié à des dépassements dans les niveaux de puissance indiqués dans les contrats, l'entreprise souhaite pouvoir mettre en œuvre des ordonnancements permettant de respecter le seuil maximal de puissance autorisé. C'est pourquoi l'outil d'aide à la décision développé a pour objectif d'assister les responsables de la production de l'usine dans la réalisation de plannings énergétiquement responsables. Les spécifications et une description précise du système ont été faites par l'IRCCyN qui a réalisé un cas d'étude complet présentant un système de production générique à partir des informations recueillies. Cette usine peut être représentée par un ensemble d'ateliers, comme dans la Figure 1 ci-dessous.

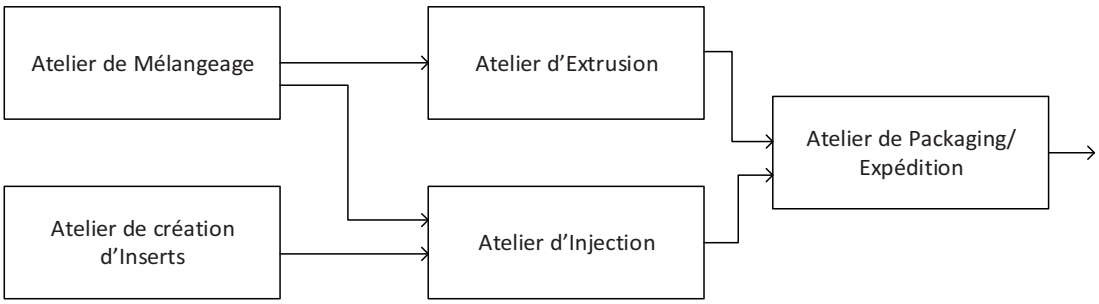

Figure 1 Macro-représentation d'un atelier d'injection/extrusion

A partir de ce cas d'étude, trois sous-systèmes ressortent : un sous-système physique qui représente les ressources impliquées dans la réalisation des différentes opérations, un sous-système logique qui décrit les entités traitées par le système de production et un sous-système décisionnel qui permet de structurer les décisions liées au pilotage du système de production. Les spécificités de ces sous-systèmes sont précisées ci-dessous.

Sous-système physique : Les machines sont regroupées en différents ateliers. La préemption n'est pas autorisée. Les machines sont toujours disponibles. Les opérateurs peuvent être indisponibles mais cette contrainte n'est pas prise en compte dans l'outil d'optimisation.

Sous-système logique : l'atelier contient des pièces regroupées en plusieurs types de lots : des lots concernant l'atelier d'injection, des lots sur l'atelier d'extrusion, des lots pour l'atelier de mélangeage et de création d'inserts. La taille des lots est définie par un système de planification en amont. Chaque lot est associé à un ordre de fabrication, qui correspond donc à un job. Un job dans l'atelier d'injection/extrusion/inserts est un lot de pièces traitées consécutivement par une machine. Dans l'atelier de mélangeage, un job est une quantité (en kg) à produire. Les jobs sont

imposés et connus à l'avance, et passent sur les machines pour y subir des opérations. Une opération décrit les machines possibles, les durées de traitement associées, et les éventuelles ressources additionnelles requises. Chaque job possède une gamme opératoire qui définit une liste ordonnée d'opérations à exécuter. Entre deux opérations sur une presse ou une extrudeuse, et pour lesquelles les jobs correspondent à des types de produits différents, un temps de reconfiguration et de préchauffage est nécessaire.

Sous-système décisionnel : Les décisions sont prises à différents niveaux et principalement par les responsables de production de chaque atelier. En fonction des dates auxquelles les produits doivent être livrés, une décision est prise sur la production des produits dans chaque atelier, comprenant l'affectation et l'ordonnancement.

De manière plus détaillée la Figure 2 présente un exemple d'une usine telle que définie dans le cas d'étude. Dans cette figure, 5 types de mélanges doivent être faits, ainsi que 3 types d'inserts. Les mélanges passent par les mélangeurs Mel_Int_1 ou Mel_Int_2 qui ont des capacités différentes, puis par le mélangeur Mel_ext_1. Cette dernière étape peut être optionnelle d'après le cas d'étude. Les inserts doivent passer par un ensemble de machines en fonction de leur type. Dans l'atelier de presse, cinq produits sont créés, ils nécessitent l'usage d'un moule et d'une presse compatibles, ainsi que du mélange et des inserts en fonction des produits. L'atelier d'extrusion requiert uniquement des mélanges, mais un produit peut nécessiter l'utilisation de plusieurs mélanges simultanément. Enfin, les produits passent par un atelier de finition/expédition.

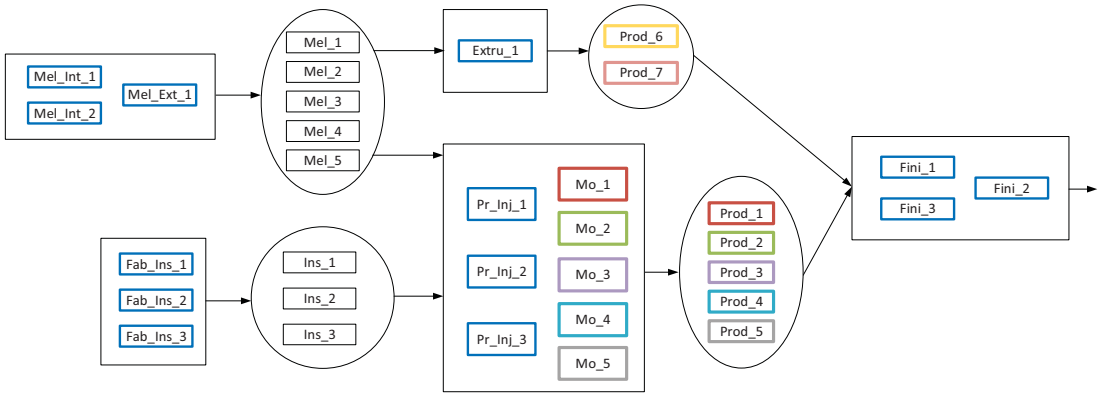

Figure 2 Représentation du processus au sein d'un atelier d'injection/extrusion

En résumé, résoudre un tel problème d'optimisation industriel nécessite la résolution de plusieurs sous-problèmes : un problème d'affectation des jobs aux machines (presses, ligne d'extrusion, …) ; un problème d'affectation des ressources aux jobs ; et un problème d'ordonnancement des jobs.

L'étude menée se focalise sur l'affectation des jobs aux machines et l'ordonnancement des jobs dans chaque atelier. Les hypothèses et les choix suivants ont été également faits :

- Les indisponibilités peuvent être négligées : sur un horizon d'une semaine, les ressources renouvelables (opérateurs) sont considérées disponibles sur toute la durée de l'horizon de planification,

- La date de disponibilité des jobs est connue et égale au début de l'horizon de temps. La planification s'effectue sur deux semaines.

- A la sortie d'un atelier d'injection ou extrusion, une pièce visite l'atelier de packaging/expédition. Il y a donc des contraintes de précédence entre deux jobs qui concernent les mêmes pièces sur ces deux ateliers. Afin de ne pas avoir à traiter ces précédences, un job intègre toutes les opérations le concernant, quels que soient les ateliers parcourus.

- Entre deux jobs, les temps de reconfiguration ne sont pas identiques et monopolisent non seulement la ou les ressources physiques (presse, moule, …), mais aussi un opérateur spécialisé. Ces durées ne sont pas intégrées dans les temps de traitement.

- Le seuil de puissance en entrée du système est constant au cours de l'horizon de planification.

- La puissance étant répartie sur l'ensemble du site industriel, les ateliers sont tous intégrés et considérés comme faisant partie d'un seul atelier les contenant tous.

3. Modèles théoriques pour l'ordonnancement

Les systèmes de production ont des structures différentes en fonction du cœur de métier des entreprises. Cependant, le nombre de modèles classiques référencés dans la littérature est plus restreint que les systèmes qu'ils représentent car ils se veulent génériques, afin d'être représentatifs du plus grand nombre d'environnements de production bien qu'ils représentent des simplifications de ces derniers. Ces modèles peuvent trouver des applications dans des systèmes de production dont les terminologies sont différentes (i.e. hôpitaux, …). Cela étant, les modèles présentés dans la littérature utilisent un vocabulaire orienté vers les systèmes de production de biens. Il est donc fréquent de retrouver les termes machines, job, opération, … Les problèmes d'ordonnancement qui visent à optimiser le fonctionnement des ateliers dans ces modèles de systèmes de production ont plusieurs définitions parmi lesquelles celles de (Carlier and Chrétienne, 1988) ou (J. Blazewicz et al., 1996) :

D'après (Carlier and Chrétienne, 1988) :

> *Ordonnancer, c'est programmer l'exécution d'une réalisation en attribuant des ressources aux tâches et en fixant leurs dates d'exécution*

D'après (J. Blazewicz et al., 1996) (traduit de l'anglais) :

> *En général, les problèmes d'ordonnancement sont caractérisés par trois ensembles : l'ensemble T constitué de n tâches* $T = \{T_1, T_2, ..., T_n\}$, *l'ensemble* $P = \{P_1, P_2, ..., P_m\}$ *de m processeurs (machines) et l'ensemble* $R = \{R_1, R_2, ..., R_s\}$ *des ressources additionnelles. L'ordonnancement, de manière générale, est l'affectation des processeurs de P et (éventuellement) des ressources de* R *aux tâches de T dans le but de réaliser toutes les tâches en respectant les contraintes.*

D'après les deux définitions ci-dessus, ordonnancer revient à définir les dates de traitement d'un ensemble de tâches sur un système de production. En fonction du problème les tâches ou opérations relatives à un même produit sont regroupées au sein d'un job. Chaque opération d'un job doit être traitée par une machine. De manière générale, une opération ne peut être effectuée que sur une seule machine à la fois et une machine ne peut traiter qu'une seule opération à la fois. Dans la plupart des cas, l'ordre de traitement de ces opérations et leurs durées dépendent de la gamme opératoire et sont connues avant de débuter l'optimisation. D'après les informations disponibles dans la littérature, plusieurs problèmes classiques d'ordonnancement existent, tels que :

- Le problème à une machine (Graham et al., 1979), où chaque job ne contient qu'une seule opération ordonnancée sur l'unique machine pendant une durée donnée ; les durées peuvent varier d'une opération à une autre. Les environnements de production complexes sont souvent décomposés en problèmes à une machine (Pinedo, 2012).

- Le problème de Flow-shop (Garey et al., 1976), où chaque job contient un ensemble d'opérations. Tous les jobs ont la même gamme opératoire qui consiste à visiter toutes les machines dans un ordre donné. Le problème à une machine est un cas particulier du problème de Flow-shop.

- Le problème de Job-shop (Manne, 1960), où les jobs peuvent avoir des gammes opératoires différentes. Chaque opération doit être traitée par une et une seule machine. Le problème de Flow-shop est un cas particulier du problème de Job-shop.

- Le problème d'Open-shop (Gonzalez and Sahni, 1976), où la gamme opératoire des jobs est à définir, seules sont connues les machines sur lesquelles les opérations doivent être traitées. Le problème de Job-shop est un cas particulier du problème d'Open-shop.

- Le problème de Group-shop (Sampels et al., 2002), à l'interface entre le problème de Job-shop et d'Open-shop. Dans ce problème certaines opérations d'un job peuvent être regroupées et traitées comme dans un Open-shop, alors que d'autres opérations doivent suivre un ordre prédéterminé.

- Le problème du Resource Constrained Project Scheduling Problem (RCPSP) (Zdamar and Ulusoy, 1995), où chaque opération d'un job consomme un à plusieurs types de ressources, dans des quantités différentes. Les ressources sont disponibles en nombre limité et on distingue les ressources renouvelables, qui peuvent être allouées à d'autres opérations une fois qu'elles sont libérées, et les ressources non renouvelables qui ne peuvent être utilisées qu'une seule fois.

Les problèmes évoqués ci-dessus (problèmes à une machine, Flow-shop, Job-shop, RCPSP…) peuvent être étendus à d'autres problèmes. Ces derniers font bien souvent intervenir plusieurs ressources pouvant traiter une opération. Ainsi, les extensions considérées font intervenir un problème d'affectation en plus du problème d'ordonnancement initial, les rendant plus difficiles à résoudre. Quelques problèmes faisant partie de ces extensions sont recensés ci-dessous :

- Le problème à machines parallèles homogènes ou hétérogènes (Horn, 1973), où chaque job ne contient là aussi qu'une seule opération, mais présentant un problème d'affectation en plus du problème d'ordonnancement. Il faut donc définir à quelle machine est affecté chaque job. Le problème à une machine est un cas particulier du problème à machines parallèles.

- Les problèmes de Flow-shop Flexible (Brah and Hunsucker, 1991) et Flow-shop Hybride (Linn and Zhang, 1999), où chaque opération d'un job peut être traitée par un ensemble de machines regroupées en étages. Les machines sont identiques dans le cadre du Flow-shop Flexible, alors qu'elles peuvent être différentes pour le Flow-shop Hybride. Les problèmes à machines parallèles sont des cas particuliers des problèmes de Flow-shop Flexible/Hybride.

- Le problème du Multi-Mode Resource Constrained Project Scheduling Problem (MRCPSP) (Sprecher, 1994). Ce problème est une extension du problème de RCPSP à la

différence que chaque opération peut être exécutée selon différents modes. Chaque mode implique des durées et des consommations de ressources différentes.

- Le problème de Job-shop Flexible (Brucker and Schlie, 1990) où, contrairement au problème de Job-shop, chaque opération peut être traitée par une machine issue d'un ensemble de machines compatibles.

Si la plupart des articles traitent de ces problèmes dans leur forme déterministe et de manière prédictive (toutes les informations sont connues au début de l'optimisation ; le résultat de l'optimisation est donc une prédiction en fonction des données au moment de l'optimisation), de nombreuses études sont menées sur leur forme stochastique (Gu et al., 2009; Juan et al., 2014; Lei, 2011) et/ou réactive (Gabel and Riedmiller, 2008; Lin et al., 2012). Dans un problème stochastique, les durées de traitements peuvent être considérées incertaines, ou des pannes de machines peuvent être prises en compte ; le problème consiste alors à définir des solutions robustes, plus à mêmes de « résister » aux événements aléatoires. Dans un problème réactif, les décisions sont prises au fur et à mesures que les données arrivent. C'est par exemple le cas lorsqu'un nouvel ordre de fabrication est émis et doit être intégré aux ordres de fabrication dont le traitement a déjà commencé.

En plus des problématiques évoquées ci-dessus, de nombreuses extensions sont étudiées dans la littérature, tels que les problèmes avec Time-lags (Caumond et al., 2008), ou avec contraintes financières (Kemmoe et al., 2015a). Ces dernières extensions peuvent être à la frontière entre certains des problèmes mentionnés ci-dessus, comme par exemple l'inclusion de précédences dans le cadre d'un problème à machines parallèles. Il est important de mentionner également les problèmes faisant intervenir des ressources additionnelles. Sans reposer sur une modélisation de type RCPSP, ces problèmes peuvent considérer la présence de ressources telles que des moyens de transport/manutention et des opérateurs (Ahmadi-Javid and Hooshangi-Tabrizi, 2017), ou des ressources énergétiques utilisées lors de la production (Bruzzone et al., 2012; Mouzon et al., 2007; Pach et al., 2014).

Pour conclure cette partie, une typologie des problèmes d'ordonnancement adaptée de (Caumond, 2006) est présentée sur la Figure 3 :

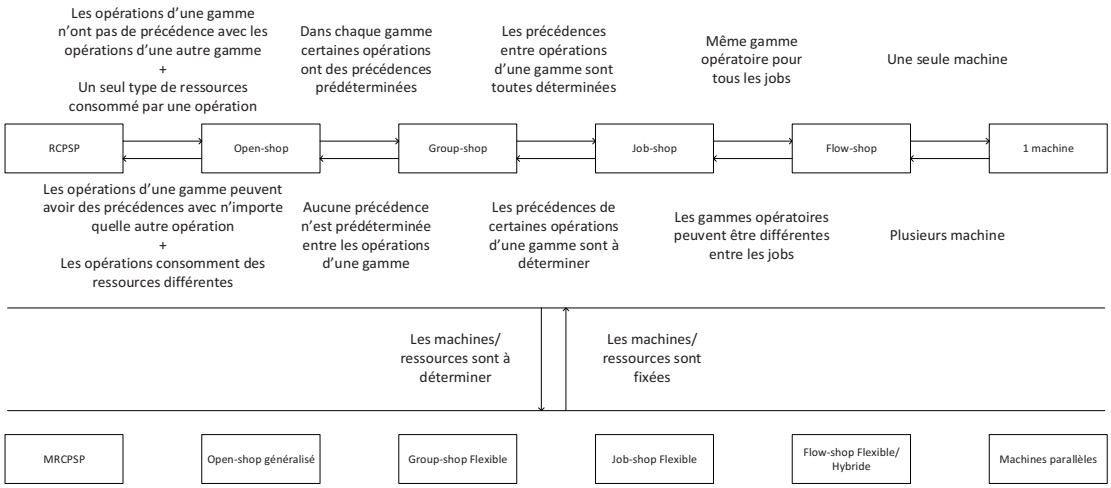

Figure 3 Typologie des problèmes classiques d'ordonnancement

Parmi les problèmes d'ordonnancement présentés, les problèmes de Job-shop et de Job-shop Flexible ont principalement été étudiés dans cette thèse car ils représentent de nombreux modèles d'ateliers. Le premier problème apparaît dans la littérature à la fin des années 1950 (Manne, 1960), alors que le deuxième a été formalisé en 1990 par (Brucker and Schlie, 1990). Un nombre conséquent d'algorithmes existe (Chaudhry and Khan, 2016; Jain and Meeran, 1999) afin de résoudre ces problèmes, comme il sera montré dans le prochain chapitre.

Cependant, tous les problèmes mentionnés précédemment font des hypothèses simplificatrices importantes : stock considérés infinis entre les machines, durées de traitements des opérations prédéterminées, pas de durées de configuration des machines, ... Il s'avère que ces simplifications conduisent aux modèles mentionnés précédemment (Flow-shop, …) et pour lesquels les résultats sont difficilement exploitables dans un contexte industriel. Ainsi, des modèles plus complets, et donc complexes à résoudre, doivent être étudiés. Le problème présenté dans la section précédente en est la preuve car il peut être vu comme un problème de Job-shop Flexible avec contraintes énergétiques, ressources additionnelles et durées de configuration. Il n'est pas possible d'appliquer directement à ce problème industriel des algorithmes d'optimisation dédiés au Job-shop ou au Job-shop Flexible, car ils doivent être modifiés de sorte à prendre en compte les contraintes ajoutées. La démarche de modélisation permettant de passer d'un système réel à un modèle le représentant, ainsi que la démarche d'optimisation d'un problème ainsi modélisé sont présentés dans la section suivante.

4. Modélisation pour l'aide à la décision

Les problématiques rencontrées dans l'industrie sont telles qu'optimiser l'ensemble du système de production reste aujourd'hui extrêmement difficile voire impossible. Ainsi, à partir du système réel étudié, une représentation fidèle doit être obtenue afin de pouvoir proposer des solutions permettant d'en améliorer la performance. La représentation d'un système de production passe par l'acquisition d'informations pertinentes qui constituent le « modèle ». Les informations retenues déterminent la finesse du modèle : plus il y a d'informations, plus le modèle est complet et donc complexe ; moins il y en a, plus le problème modélisé sera facile à résoudre. C'est également le modèle qui permet de définir les axes de recherche visant à la fois à déterminer la structure d'une solution, et l'évaluation de celle-ci afin de déterminer sa valeur, ainsi que les algorithmes d'optimisation qui ont pour objectif d'explorer un ensemble de solutions afin de trouver la

meilleure possible. Passer du système réel à un de ses modèles nécessite d'introduire des hypothèses simplificatrices jusqu'à atteindre un des modèles classiques de la littérature. Ensuite, les éléments supprimés sont réintroduits au fur et à mesure. La prise en compte de nouvelles contraintes peut requérir de modifier la façon dont les solutions sont évaluées, et également la façon dont l'espace des solutions est parcouru par les algorithmes d'optimisation proposés. Dans (Kellert et al., 1997) et (Tchernev, 1997) un modèle fonctionnel du processus de modélisation pour la résolution d'un problème industriel est proposé.

Le processus de modélisation est donc essentiel et nécessite des outils méthodologiques. Dans la suite, la notion de modèle est introduite. Puis le processus de modélisation est proposé, et adapté aux applications d'optimisation. Enfin, les trois types de modèles utilisés dans cette thèse sont présentés.

4.1. Notion de modèle

« Mais j'ai cru qu'il fallait chercher des représentations sur lesquelles on put opérer, comme on travaille sur une carte, ou l'ingénieur sur une épure… et qui puissent servir à faire. » (P. Valéry)

Les systèmes de production sont créés par l'homme mais leur complexité est telle qu'il n'est bien souvent pas possible pour un individu d'en déterminer un fonctionnement optimal. C'est là qu'interviennent les modèles, en tant que description de ces systèmes. Si en physique les modèles consistent en des lois et principes, (Pritsker et al., 1997) notent que la modélisation de systèmes industriels complexes est souvent rendue difficile par l'absence de suffisamment de lois physiques les représentant et par la présence de nombreux éléments difficiles à décrire et à prendre en compte. En cela, la construction d'un modèle est une tâche complexe car elle prend en compte non seulement la création, mais aussi la validation, la vérification, ainsi que l'utilisation du modèle. Parmi les modèles célèbres, les modèles de simulation figurent parmi les plus répandus. La conception d'un modèle de simulation consiste à réaliser un modèle conceptuel en vue de résoudre un problème qui est ainsi représenté sous une forme intelligible et plus facilement manipulable afin d'obtenir des solutions à l'aide d'un outil de simulation (Nance, 1994). Le terme « simulation » admet différentes significations en fonction de la problématique étudiée. Les plus notables sont :

i. La simulation de Monte Carlo : on obtient une solution d'un problème en répétant la simulation à partir de différents scénarii et en appliquant des techniques issues du domaine des statistiques.

ii. La simulation continue : les expérimentations sont effectuées avec un modèle qui progresse d'état en état de manière continue, sans arrêt, en reposant bien souvent sur des équations différentielles.

iii. La simulation à évènement discret : le modèle évolue d'état en état avec des durées variables entre chaque état, le rendant discontinu. Un exemple classique concerne les files d'attente.

Si la simulation met en œuvre des modèles destinés à différentes problématiques, le terme « modèle », définit des objets et des concepts très différents selon le milieu dans lequel il est employé. Il y a les modèles descriptifs, qui ont pour objectif la compréhension de systèmes complexes, mais aussi les modèles prédictifs, qui ont pour objectif de prévoir un événement à venir (les modèles météorologiques, l'expansion d'une épidémie, …). Ces deux approches sont liées, la seule description d'un phénomène n'étant guère utile si l'on ne peut anticiper ceux à venir. Un modèle peut avoir une forme concrète, manipulable (une maquette par exemple) mais peut aussi être immatériel ; c'est le cas des modèles mathématiques qui sont plus abstraits que leurs

homologues physiques. Cependant, dans la plupart des cas, un modèle ne peut représenter un système de manière exhaustive au risque d'être inutilisable.

Ainsi, dans le processus de modélisation, des simplifications par rapport au système observé (ou celui à concevoir) pour permettre d'arriver aux objectifs souhaités sont à considérer. Une simplification consiste à ne pas modéliser un élément du système réel s'il ne participe pas à l'amélioration du système de production dans les objectifs qui sont fixés. Ces simplifications font partie du processus de modélisation et ont vocation à permettre d'améliorer les performances du système de production. Cet objectif est bien résumé par la phrase énoncée plus haut de Paul Valéry : « Servir à faire ».

Pour illustrer ce que doit contenir un modèle, la Figure 4 qui reprend des éléments clés de la notion de modèle est introduite. Dans cette figure, un système réel est représenté par une forme aux contours irréguliers. Le modèle extrait à partir de cette forme est présenté en dessous et répond aux trois objectif représentés par les flèches : le respect de l'objectif du modèle, le niveau de détail admis, et les limites du modèle. En tenant compte de ces trois points, un acteur peut reconnaître la ressemblance du modèle avec le système. Le niveau de détail détermine le nombre de simplifications faites, et par la même occasion la complexité du modèle. On parle « d'hypothèses simplificatrices ». Ces hypothèses sont bien évidemment en adéquation avec les objectifs du problème. Si l'on souhaite optimiser le travail des opérateurs sur une ligne de production, on ne peut pas faire l'hypothèse que le système fonctionne sans opérateurs. Lorsque l'objectif est de modifier un système de production, le modèle représentant le système à un instant donné ne sera probablement pas le même modèle que celui modélisant le système vers lequel l'entreprise souhaite se diriger. Aussi, la conception d'un modèle consiste en une approche itérative, et il est judicieux de vérifier sa cohérence au fur et à mesure de son élaboration ainsi que de le valider en comparant les résultats qu'il permet d'obtenir avec ceux observés sur un système connu, si le système existe.

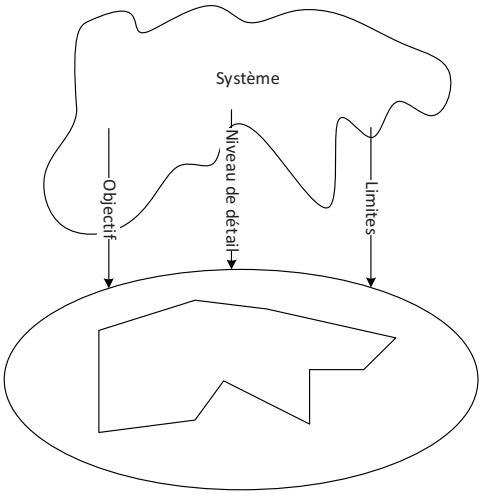

Figure 4 Passage d'un système réel à un de ses modèles

Les hypothèses simplificatrices réalisées sur un modèle dépendent en grande partie de la personne réalisant le modèle et de l'objectif à atteindre. Par exemple, si le problème consiste à déterminer le taux d'utilisation d'une ressource dans un système de production, il est préférable de prendre en compte les autres ressources qui travaillent en parallèle, par contre il n'est peut-être pas nécessaire de prendre en compte la gestion des stocks de l'entrepôt si ceux-ci ne sont jamais bloquants. Il n'y a pas de risques à faire des hypothèses dans la mesure où l'on reste conscient de leur portée : si le problème consiste à minimiser les stocks en entrée du système alors le modèle

sera très différent et ne pourra probablement pas faire l'impasse sur cette problématique. La modélisation requiert une vigilance particulière car elle est souvent l'origine de l'échec de nombreux projets industriels, dont la première étape consiste à avoir un modèle fidèle et représentatif de la situation. En plus de la ou des personnes réalisant le modèle, il est primordial d'instaurer un cadre de communication et d'échanges entre les différents experts du système de production modélisé.

En effet, les systèmes sont d'une complexité telle qu'aucune personne ne possède toute l'information sur la manière dont il fonctionne, et un expert peut négliger une donnée lui paraissant superflue, alors qu'elle est essentielle aux yeux d'un autre. En cela il est important de recouper les informations et de prendre en compte les différentes perceptions que les experts peuvent avoir à leur niveau de compréhension du système. La notion de « collecte d'information » dans le but de concevoir un modèle est essentielle, car si les informations obtenues ne sont pas suffisantes, ou pire, fausses, le modèle sera inutilisable. Le processus de modélisation requiert donc de réunir les connaissances des experts, de s'assurer de leur validité, de les confronter entre elles, de les utiliser et de les formaliser.

La formalisation a pour objectif d'extraire, à partir des descriptions littéraires d'un système de production, les éléments qui les constituent et de les regrouper en objets afin de rendre intelligible cette connaissance et de l'organiser. L'objectif est donc de passer d'un langage à un autre. Parmi les langages visant à formaliser des systèmes, il est possible de trouver par exemple Merise (Tardieu et al., 1994), UML (Unified Modeling Language (Booch et al., 2000)), le modèle Entités / Associations (Chen, 1976), … Cependant si ces formalismes permettent d'ordonner les informations et de définir les liaisons qui existent entre elles, ils ne permettent pas de définir les hypothèses à faire, ni si le modèle sera une représentation fidèle du système. Au final, seul un modélisateur aux compétences affirmées peut concevoir un modèle pertinent.

En fonction des problématiques évoquées par une entreprise il peut être intéressant de proposer un modèle dédié à chacune d'entre elles, si ces problématiques peuvent être traitées indépendamment. Une entreprise ayant deux sites de production qui ne dépendent pas l'un de l'autre, avec des procédés de fabrication qui leurs sont propres, aura intérêt à avoir un modèle représentant chaque site. Pourtant si l'on considère également le transport mutualisé des produits sortants des deux sites alors la problématique n'est plus la même. Il est donc important de définir des modèles adaptés à chaque situation et dont la portée correspond aux besoins réels. Aussi, il est important de se référer à ce qui existe dans la littérature afin de réutiliser des modèles déjà développés et pouvant être utilisés ou adaptés à la problématique étudiée.

4.2. Formalismes de modélisation utilisés

Dans ce paragraphe, trois formalismes mis en œuvre pour la résolution des problèmes traités dans cette thèse sont présentés. Une formalisation mathématique, une approche par graphe conjonctif-disjonctif et la simulation à événements discrets sont présentées. La finalité de la modélisation est l'élaboration des modèles du système étudié en vue d'apporter une aide à la décision. Il est possible d'inclure dans ces trois types de modèles les différentes contraintes qui constituent le ou les problèmes traités. Ces contraintes sont plus ou moins facilement représentables en fonction du type de modèle choisi. Ainsi, des contraintes qui se formalisent facilement de manière mathématiques peuvent être difficiles à représenter au sein d'un modèle de simulation, et vice-versa.

En fonction du problème traité, la possibilité d'avoir plusieurs approches visant à le résoudre est un réel avantage. Ainsi, alors qu'un modèle mathématique peut permettre de trouver des

solutions optimales, un modèle de simulation sera plus à même de faire apparaitre certaines caractéristiques d'un système nécessitant par exemple une solution non optimale au sens de la résolution du modèle mathématique. De plus, avec la taille grandissante des problèmes, un modèle de simulation peut mettre en évidence la qualité de règles de gestion au sein d'un système de production, permettant de préserver les performances du système, sans avoir pour autant à résoudre un modèle mathématique. Pourtant, un programme linéaire en nombre entier (PLNE) qui consiste en la mise en œuvre d'un modèle mathématique, peut permettre d'évaluer plus finement la qualité d'un modèle de simulation. Ainsi, ces approches sont complémentaires

4.2.1. Formalisation mathématique

La description formelle d'un système de production dans un formalisme mathématique (ou "modèle mathématique") permet de rendre univoque le modèle de connaissance préalablement établit, en représentant les interactions ayant lieu au sein du système de production à l'aide de notations mathématiques. Le modèle mathématique peut également être utilisé au sein d'un outil de résolution exacte. A ce titre, il est important de faire la distinction entre modèle mathématique et modèle linéaire. Un modèle mathématique vise à représenter mathématiquement un problème en intégrant un ensemble de contraintes. Ces contraintes peuvent être linéaires ou non. Les contraintes non linéaires font souvent apparaître des termes quadratiques, et le passage d'un modèle non linéaire à un modèle linéaire n'est pas toujours aisé.

Il est utile de disposer d'un modèle mathématique, qu'il soit transcrit en modèle linéaire ou non, pour déterminer la solution optimale d'un problème, ou pour vérifier ou valider la qualité des solutions obtenues à l'aide de méthodes approchées. Cependant, le modèle mathématique doit être vérifié et validé lui aussi pour s'assurer qu'il ne contient aucune erreur, et que toutes les contraintes ont été prises en compte. S'il peut être fastidieux de vérifier manuellement un modèle mathématique, ou encore d'analyser les résultats retournés, l'utilisation d'un support visuel de la solution tel qu'un diagramme de Gantt peut être appropriée. De même, les solutions retournées par un modèle mathématique doivent être confrontées aux experts du système qui peuvent détecter des incohérences. Pour faciliter l'étude de la validité d'un modèle, il peut être intéressant de développer des problèmes de petite taille, facilement vérifiables manuellement. Il est alors aisé de se rendre compte que des contraintes qui devraient être respectées sont violées. Une contrainte violée peut être par exemple l'utilisation de plus de ressources à un instant donné que celles réellement disponibles à ce moment-là.

Comme montré dans la Figure 4, un modèle est réalisé dans un but précis (l'objectif à atteindre). Ainsi il est essentiel de montrer que le modèle correspond aux objectifs pour lesquels il a été réalisé. Pour cela il est possible soit de comparer le modèle au système réel s'il existe, ou de montrer qu'une solution correspond bien aux contraintes auxquelles le modèle est soumis. Dans le premier cas, si une solution est déjà appliquée sur le système de production, il suffit de vérifier que l'évaluation par le modèle mathématique de la solution en cours retourne bien le résultat attendu. Dans le second cas, il faut utiliser une approche par expérimentation. En faisant varier les paramètres du modèle, il est possible de vérifier que celui-ci se comporte comme il est supposé le faire : par rapport à une solution donnée, si les paramètres sont plus contraignants, la nouvelle solution doit être dégradée ; s'ils le sont moins, la solution obtenue doit être de meilleure qualité. Dans tous les cas la solution doit être validée par des experts du système et ne saurait être appliquée sans précautions. En effet, l'application d'une solution sur un système de production représente des enjeux importants pour une entreprise, il faut donc s'assurer que, en répondant aux objectifs fixés, aucun risque n'est introduit quant au bon fonctionnement du système de production. On peut par exemple prendre la précaution d'appliquer la solution sur un ensemble réduit de produit et de l'étendre petit à petit à l'ensemble des références à fabriquer. Dans tous les cas, l'étape

consistant à utiliser un problème à taille réduite permet une analyse plus facile et un échange plus clair avec les experts.

Enfin, comme mentionné ci-dessus, un modèle mathématique permet de vérifier et de valider la qualité des solutions obtenues à l'aide de méthodes approchées, telles que des heuristiques, ou des métaheuristiques qui nécessitent souvent un paramétrage particulier en fonction des problèmes étudiés. Or, un modèle linéaire permet l'obtention de solutions exactes, il est donc possible de comparer les solutions retournées par une méthode approchée avec celles obtenues par le biais d'une méthode exacte de résolution sur des problèmes à tailles réduites. Si la méthode approchée présente le comportement attendu vis-à-vis des solutions exactes (c'est-à-dire qu'elle retourne les solutions exactes ou à défaut des solutions avec une faible déviation), alors une hypothèse peut être faite quant à sa capacité à retourner des solutions de bonnes qualité sur des problèmes de taille plus conséquente.

La formalisation mathématique permet donc de représenter le système de manière non ambigüe, d'obtenir des solutions optimales sur des problèmes à taille réduite, et enfin, de valider les performances d'une méthode approchée.

4.2.2. Graphe conjonctif - disjonctif

Le graphe conjonctif - disjonctif a été introduit par (Roy and Sussmann, 1964). Il permet de représenter un problème d'ordonnancement sous forme de graphe en faisant apparaître les contraintes liant les différentes opérations devant intervenir sur un système de production. Or, les problèmes d'ordonnancement comportent de nombreuses contraintes entre les opérations, notamment des contraintes concernant les dates de début d'opérations par rapport à d'autres. De plus, la présence de ressources mutualisées telles que des machines est particulièrement bien représentée par ce type de graphe.

Dans les problèmes d'ordonnancement, les dates de début des opérations doivent être fixées. Du fait de la présence de contraintes entre les opérations, il faut donc définir quelle opération est traitée avant quelle autre par une ressource (machine). Cette recherche visant à définir l'ordre de traitement des opérations consiste en un problème combinatoire de recherche de séquences. Une séquence entre deux opérations dans un graphe est représentée par un arc. De plus, dans les problèmes d'ordonnancement, une opération pourra commencer dès la fin de l'opération la précédant sur une ressource. Il est donc plus simple de considérer une représentation sous forme de graphe que d'essayer de fixer empiriquement les dates des opérations ce qui risque de conduire à des chevauchements. La modélisation sous forme de graphe conjonctif-disjonctif permet de lever toute ambiguïté sur les dates de traitement des opérations. Une fois que l'ensemble des arcs définissant les précédences sont fixés, une simple exploration de graphe permet d'obtenir la valeur du critère de performance de la solution ainsi exprimée.

Un graphe conjonctif-disjonctif consiste en un triplet $G(V, A, E)$. L'ensemble V correspond aux opérations à ordonnancer. Chacune de ces opérations est représentée par un nœud dans le graphe. L'ensemble A correspond aux arcs modélisant les précédences qui constituent les données du problème. Ainsi, un couple d'opérations $O_i, O_j \in A$ sera contraint par une équation de la forme $s_j \geq s_i + p_i$ où s_i représente la date de début de l'opération O_i et s_j la fin de l'opération O_j. p_i représente la durée de traitement de l'opération O_i. Chaque arc modélisant un couple d'opérations $O_i, O_j \in A$ est pondéré par p_i. Le graphe $G(V, A)$ est appelé graphe conjonctif. Enfin, l'ensemble E contient tous les couples O_i, O_j liées par une arrête, c'est-à-dire les opérations pour lesquelles l'ordonnancement est à définir. L'objectif est donc d'orienter ces arrêtes afin de définir qui de O_i

ou O_j est traité avant l'autre. Il faut donc vérifier la contrainte suivante : $s_j \geq s_i + p_i \vee s_i \geq s_j + p_j$. Le graphe est alors appelé conjonctif-disjonctif.

L'objectif vise donc à rechercher les dates de début des opérations de V tout en vérifiant toutes les contraintes liant les opérations. Pour les problèmes d'ordonnancement, cette recherche s'effectue le plus souvent par le calcul d'un plus long chemin, à l'aide d'un algorithme de type Bellman-Ford par exemple (Bellman, 1958).

La modélisation des problèmes d'ordonnancement repose souvent sur l'utilisation du graphe conjonctif-disjonctif. Dans les chapitres suivants, les problèmes étudiés seront modélisés à l'aide de ce graphe.

4.2.3. Simulation à événements discrets

La troisième approche utilisée dans cette thèse est une approche par simulation. Comme noté par (Pritsker et al., 1997), la simulation est le processus de conception d'un modèle d'un système réel sur lequel des expérimentations sont faites. Elle consiste donc à réaliser une abstraction du système réel, appelée « modèle de simulation » et à définir les expérimentations à effectuer sur ce modèle. L'avantage de l'approche par simulation est qu'elle permet d'extraire de la connaissance d'un système tout en respectant certaines caractéristiques :

- Dans certains cas, la simulation permet d'étudier des systèmes *sans avoir à les construire* (dans le cas où plusieurs propositions de systèmes sont faites).
- Elle permet d'analyser les systèmes *sans les perturber*. Il peut effectivement être délicat, voire dangereux, d'expérimenter directement sur le système de production (dont le coût peut être élevé, et dont la rentabilité dépend du bon fonctionnement) ou de manière plus générale, sur l'objet de l'étude (centrales nucléaires, hôpitaux, …).
- Enfin, elle est aussi utile afin de *ne pas détruire le système* étudié, quand l'objet de l'étude vise à déterminer la capacité du système à résister à des événements critiques, ou à fonctionner dans ses conditions limites d'utilisation.

Considérant ces trois critères, la simulation permet d'analyser des systèmes et d'évaluer leurs performances. La simulation nécessite de prendre en compte les états d'un système. Ces états doivent pouvoir être formalisés, ainsi que les entités du système. La simulation consiste alors à passer d'un état à un autre, afin d'observer le comportement du système. Le passage d'un état à un autre peut se faire en utilisant des règles préétablies et usuellement appliquées sur le système réel. Ces règles peuvent être par exemple de définir l'ordre de passage des produits en attente devant une ressource du système. Les changements au sein d'un système peuvent se faire de manière continue, ou de manière discrète. Bien que les procédures utilisées dans un cas ou dans l'autre ne soient pas les mêmes, les concepts visant à décrire les changements dans le système demeurent.

Pour simuler le comportement d'un système, des environnements dédiés à la simulation existent et permettent de faciliter la mise en œuvre des modèles réalisés. La principale force de ces langages consiste à fournir des éléments génériques permettant l'écriture des modèles de simulation. Ils incluent aussi de nombreux éléments visant à évaluer la performance du système modélisé, tels que le calcul du temps moyen passé par les entités dans le système, le taux d'utilisation des ressources, … Toutes ces informations sont facilement enregistrées sous forme de rapports détaillés. L'existence de tels outils permet donc un gain de temps non négligeable pour qui veut réaliser un modèle de simulation sans avoir à concevoir soi-même l'ensemble de l'architecture permettant de faire des simulations (structures de données, moteur de simulation,

…). Parmi les environnements de développement fréquemment utilisés et implémentant des langages de simulation, Arena, Simio, FlexSim ou encore AnyLogic sont très répandus.

Un modèle de simulation permet de répondre à une question du type « Que se passe-t-il Si… ? ». On dit également qu'on simule un scenario. S'il est possible de tester plusieurs scenarii, la recherche de la configuration du système dans laquelle il est le plus performant implique l'utilisation d'autres approches. Lorsque le nombre de scenarii testés est faible, une de ces approches consiste en un plan d'expérience ; mais ces plans d'expérience popularisés par la méthode Taguchi (Taguchi, 1987) sont vite limités lorsque le nombre de configurations à explorer augmente. La simulation de tous les scenarii est impossible du fait du temps d'exécution important généralement imputé aux outils de simulation ; vouloir utiliser un module de simulation dans le but d'optimiser un système requiert donc l'utilisation d'un module d'optimisation. On parle alors de couplage Optimisation-Simulation, ou Simulation-Optimisation. Dans le paragraphe suivant, plusieurs approches utilisées en optimisation sont présentées.

4.3. Processus de modélisation et optimisation

Comme mentionné précédemment, il n'est pas possible de définir des règles concrètes visant à formuler des hypothèses simplificatrices en vue de la modélisation d'un système de production. Cependant, il est possible de définir une approche visant à simplifier la démarche de modélisation. Ce processus est représenté sur la Figure 5, comme une démarche cyclique en quatre étapes qui n'est pas sans rappeler le processus d'amélioration continue Plan-Do-Check-Act (PDCA) popularisé par la roue de Deming (Deming, 1986; Shewhart, 1939).

La première étape, assimilable au « Plan » du processus PDCA, a pour objectif la construction du modèle de connaissances. Dans ce modèle sont recensées toutes les interactions qu'ont les entités du système de production entre elles, ainsi que leurs caractéristiques. Il contient également les objectifs à atteindre. Cette étape se veut être une description du problème et non de la solution, et aucun élément constitutif de cette dernière ne doit transparaître dans le modèle de connaissance.

La deuxième étape, assimilable au « Do » du processus PDCA, vise à concevoir le modèle qui a pour objectif de résoudre le problème. On parle de modèle d'action. Ce modèle d'action est construit sur la base du formalisme adopté (modèle mathématique, modèle de simulation,…) et contient le modèle du système étudié ainsi que l'approche de résolution choisie.

La troisième étape consiste à vérifier (« Check ») et exploiter les résultats obtenus par le modèle d'action. La phase de vérification est importante car elle permet de s'assurer de la qualité des résultats retournés et de leur pertinence. En fonction des résultats il peut être nécessaire d'adapter l'approche de résolution si celle-ci s'avère inefficace.

La quatrième étape consiste à agir sur le système (« Act ») en appliquant les préconisations retournées par les résultats obtenus dans l'étape précédente. L'action sur le système n'est pas du ressort du modélisateur mais des décideurs de l'entreprise dans laquelle le système évolue. Ci-dessous, la deuxième étape du processus de modélisation adaptée aux problèmes d'optimisation est détaillée. Cette approche de modélisation est basée sur un processus qui sépare la phase de modélisation et l'utilisation du modèle. Dans cette approche, principalement deux types de modèles sont mis en œuvre : le modèle de connaissance et le modèle d'action. Le modèle de connaissance décrit la structure et le fonctionnement du système de manière simple voire graphique. Si le système existe, le modèle de connaissance consiste à recueillir les informations

observées sur le système. Si le système n'existe pas encore, le modèle de connaissance comporte alors les spécifications du futur système (Gourgand, 1984).

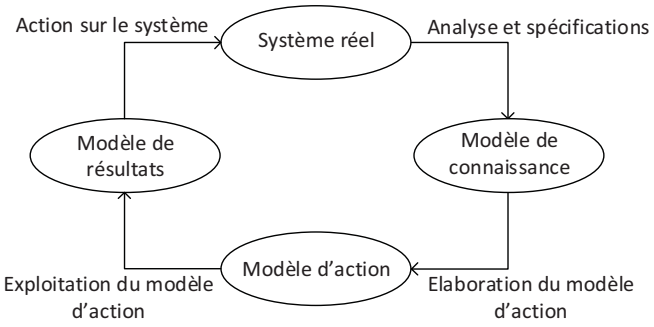

Figure 5 Processus de modélisation

L'objectif de l'optimisation est d'améliorer les performances d'un système de production. Une telle démarche correspond à l'étape « Modèle d'action » de la Figure 5. A partir du modèle de connaissances, il est possible de connaître ce qui définit une solution du problème étudié. Cependant, travailler directement sur une solution peut avoir des conséquences sur la flexibilité du module d'optimisation à évoluer dans le temps. Pour pallier ce problème, il est courant de travailler sur des solutions partielles.

4.3.1. Solutions partielles

D'après (Grangeon, 2001), une solution est dite "partielle" lorsqu'elle ne contient pas toutes les informations décrivant la solution entièrement. Une solution partielle peut être un ordre total des opérations, un ordre des opérations sur les machines, des matrices de précédence, … Alors qu'une solution « complète » contiendra également les dates de début de chacune des opérations devant être traitée par la machine, et tout autre critère dérivant de la solution partielle. De manière générale il est plus facile de travailler sur une solution partielle que sur une solution complète. Le passage d'une solution partielle à une solution complète se fait en évaluant ou en calculant différents critères de performance. Dans la suite, le terme solution est utilisé pour désigner une solution complète.

4.3.2. Evaluation d'une solution partielle

L'évaluation d'une solution partielle se fait par un module d'évaluation des performances. Ce module retourne la solution définitive avec tous les critères qui la représentent. Il existe plusieurs types de modules d'évaluation (Sarramia, 2002). On trouve par exemple, les modules contenant les fonctions triviales, en d'autres termes les fonctions mathématiques, qui à un ensemble de paramètres en entrée retournent une solution. Ensuite viennent les « procédures d'évaluation des performances ». Ces procédures réalisent des calculs complexes à partir d'une solution partielle, parmi lesquels le calcul d'un plus long chemin dans un graphe. Cette procédure est considérée comme une procédure déterministe : à une solution partielle ne correspond qu'une solution quel que soit le nombre d'exécutions de la procédure d'évaluation. Finalement, le dernier module est constitué de modèles de simulation. Un tel module prend tout son intérêt lorsque des événements aléatoires sont pris en compte dans le modèle. Comme mentionné précédemment, ces modèles peuvent être développés à l'aide d'outils spécialisés (langages de simulation, bibliothèques dédiées,

…) ou avec un simulateur spécifiquement conçu pour le problème. En sortie du module d'évaluation, une solution ainsi que les critères de performance qui la définissent est obtenue.

On dira d'un module d'évaluation qu'il est optimal s'il retourne, à partir d'une solution partielle, les meilleurs critères associés à cette solution. Si ce n'est pas le cas, le module est dit heuristique. Il semble évident qu'un module d'évaluation optimal est préférable ; c'est par exemple le cas d'un module qui, à une affectation des opérations aux machines et à un ordonnancement, retourne les dates de débuts au plus tôt dans un problème de Job-shop Flexible. Si l'on ne prend plus en compte l'affectation et que celle-ci est effectuée à partir de règles heuristiques au cours de l'évaluation de la solution partielle, on est en présence d'un module heuristique. De manière générale, un module optimal est préféré, mais en fonction de la complexité des problèmes, certaines décisions peuvent être prises de manières approchées.

Une solution partielle admet une représentation au sein du module d'évaluation. On parle également de « solution codée » pour désigner cette représentation. On fait la distinction avec les « solutions évaluées » ou « solutions stockées », aussi appelées simplement « solutions ». Le module d'évaluation repose sur des algorithmes dont le but est de passer d'une solution codée (appartenant à l'espace de codage) à une solution évaluée (appartenant à l'espace des solutions). Ils sont appelés "algorithmes de décodage". La réciproque de ces algorithmes, permettant de passer d'une solution évaluée à une solution codée, est appelée "algorithme de codage". La complexité d'un problème d'optimisation réside en partie sur les représentations choisies pour les solutions partielles. Ces solutions partielles peuvent être représentées par des tableaux à une ou plusieurs dimensions qui vont contenir les ordres de passage des opérations, leurs affectations, et toute donnée jugée utile.

En fonction de l'algorithme de décodage utilisé, et de la représentation choisie pour une solution partielle, différents cas de figures peuvent être rencontrés, comme indiqué dans la Figure 6.

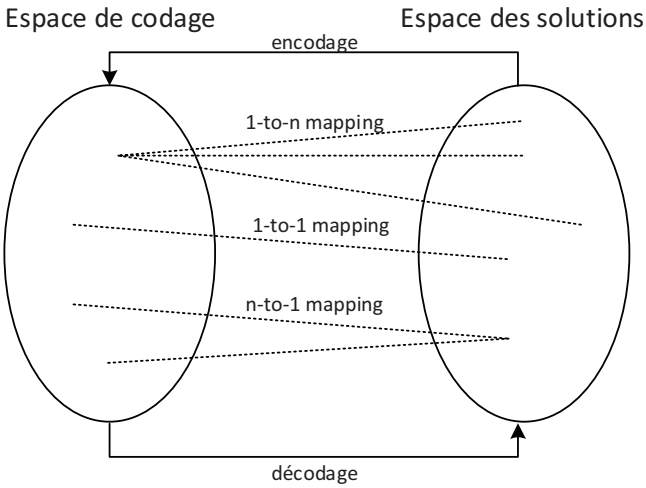

Figure 6 Relations entre l'espace de codage et l'espace des solutions

La Figure 6 est une adaptation de deux graphiques proposés par (Cheng et al., 1996) afin de représenter les liens entre l'espace de codage et l'espace des solutions dans les problèmes d'ordonnancement de type Job-shop. Le terme « mapping » définit les associations qui peuvent exister entre les éléments d'un espace avec l'autre. Ainsi, une représentation de type « 1-to-n » est la moins désirée, alors qu'une représentation en « 1-to-1 » est la meilleure de toutes celles présentées parce qu'elle permet de passer d'un espace à l'autre de manière équivoque. En fonction

du codage choisi, l'algorithme de décodage peut ne pas être capable de retourner une solution respectant toutes les contraintes du problème. On parle de représentation « surreprésentée ». La façon de passer de l'espace de codage à l'espace des solutions se fait par décodage, tandis que la représentation d'une solution dans l'espace de codage se fait par une procédure d'encodage. Dans l'idéal, une représentation ne doit retourner que des solutions faisables, comme représenté dans la Figure 7, inspirée de (Cheng et al., 1996).

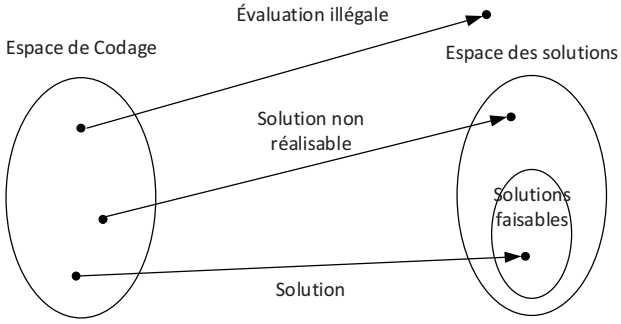

Figure 7 Faisabilité des solutions après évaluation

Si le module d'évaluation permet d'évaluer la qualité d'une solution partielle proposée par un algorithme d'optimisation, ce dernier doit proposer plusieurs solutions partielles afin de trouver la meilleure solution possible. Il est alors fréquent d'avoir une approche cyclique qui consiste à passer du module d'évaluation au module d'optimisation, qui va mettre à profit la connaissance obtenue à la suite du passage dans le module d'évaluation afin de proposer d'autres solutions partielles à évaluer. Cette démarche est mentionnée dans (Grangeon, 2001) et représentée dans la Figure 8 ci-dessous.

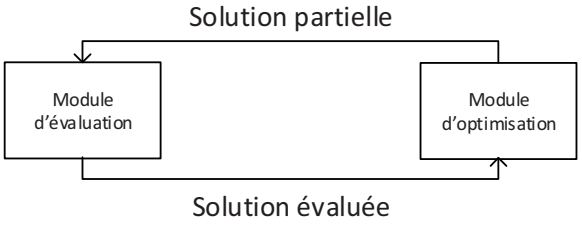

Figure 8 Processus cyclique optimisation/évaluation

4.3.3. Module d'optimisation

Le module d'optimisation permet de construire et de rechercher une solution de bonne qualité. Dans tous les cas, un module d'évaluation des performances utilisé pour évaluer la ou les solutions rencontrées est lié au module d'optimisation. Si des hypothèses sont faites au niveau du module d'évaluation afin de répondre aux objectifs fixés, il en est de même pour le module d'optimisation. En effet, en fonction de la méthode d'optimisation choisie, il peut être approprié de faire des simplifications. Par exemple, les événements aléatoires sont mal gérés par la programmation linéaire, et, si un problème contient de très nombreuses décisions à effectuer ou dont la taille est excessive il peut être préférable d'utiliser une heuristique de construction plutôt qu'une métaheuristique. Ces algorithmes sont décrits plus en détail dans la prochaine section.

Le module d'optimisation doit également contenir les méthodes permettant d'améliorer la qualité d'une solution. Ces méthodes dépendent des objectifs de l'étude et des hypothèses simplificatrices réalisées sur le système réel, mais également sur le problème lui-même. Par exemple, si l'objectif est de réduire la durée du traitement des jobs lors de l'ordonnancement (la longueur d'un chemin critique dans un graphe), il peut être nécessaire de modifier le chemin en échangeant l'ordre des arcs. Le module doit donc contenir une méthode qui permet cette modification. Ou encore, si plusieurs ressources peuvent traiter une opération, une amélioration peut consister à affecter à l'opération une autre ressource issue de l'ensemble des ressources valides. Ces méthodes implémentées dans le module d'optimisation reposent sur la façon dont sont représentées les solutions partielles au sein du module d'optimisation.

Dans tous les cas de figure, il est judicieux d'adopter une approche itérative dans la conception d'un module d'optimisation en partant du modèle le plus simple vers le plus difficile. Le module d'évaluation jouant également un rôle important, une approche itérative est à préférer lors de son développement.

Dans le cadre d'un problème d'ordonnancement tel que le Job-shop par exemple, le module d'optimisation et le module d'évaluation fonctionnent de concert afin d'arriver à une solution de bonne qualité. Le module d'optimisation propose des solutions partielles qui doivent être évaluées par le module d'évaluation. Ces solutions partielles peuvent être de différentes natures en fonction de la représentation choisie. Une fois que le module d'optimisation a sélectionné une solution partielle, le module d'évaluation retourne la solution définitive avec toutes les valeurs des critères de performance associés à la solution. Pour le Job-shop, une solution est un ordonnancement qui associe à chaque opération une date de début. Le critère de performance peut être différent en fonction du problème étudié. Pour le Job-shop, de nombreux travaux visent à optimiser le makespan – la date à laquelle toutes les opérations ont été traitées, qui est un critère évaluant la productivité d'un atelier. Un autre critère couramment utilisé est la minimisation de la somme des retards, lorsque des dates limites sont définies pour le traitement des opérations ou des jobs auxquels elles appartiennent. Ces informations sont ensuite utilisées dans le module d'optimisation pour explorer de nouvelles solutions partielles.

Pour plus d'informations sur la démarche de modélisation dans les problèmes d'optimisation, le lecteur peut se référer à (Caumond, 2006). La section suivante présente différentes méthodes de résolution de problèmes d'optimisation.

5. Méthodes de résolution

Il existe de nombreuses approches pour résoudre des problèmes d'optimisation. Dans cette section certaines des méthodes utilisées sont introduites. Les fondements de ces méthodes résident dans les problèmes d'optimisation combinatoire. Ces problèmes sont donc présentés dans cette section. Différentes méthodes sont ensuite présentées, avec une attention particulière portée sur les métaheuristiques car ce sont les méthodes principalement utilisées dans cette thèse. Des outils permettant de comparer les solutions retournées par ces méthodes d'optimisation sont introduits, afin de pouvoir par la suite comparer les méthodes proposées.

5.1. Problèmes mono- et multiobjectifs

Définition : Problème d'optimisation combinatoire mono-objectif (Papadimitriou and Steiglitz, 1998)

Soit un problème P *et* N *un ensemble discret et fini. Soit* R *un sous-ensemble de* N *constituant les solutions réalisables de* P*. Soit* f *une fonction appelée "fonction objectif".* $max\{f(S)|S \in R\}$*et* $min\{f(S)|S \in R\}$*sont des problèmes d'optimisation combinatoire.*

Les problèmes d'optimisation à un critère traités dans cette thèse ont des modélisations discrètes. Ils rentrent donc naturellement dans le cadre des problèmes d'optimisation combinatoire mono-objectif qui reposent sur des ensembles discrets. Comme défini ci-dessus, un problème d'optimisation combinatoire mono-objectif consiste à minimiser une fonction objectif ou à la maximiser. Dans cette thèse les problèmes de minimisation sont considérés. Cependant, les remarques effectuées dans cette section restent vraies dans le cas de problèmes de maximisation.

Dans la définition énoncée ci-dessus, l'objectif est de trouver une solution S qui appartient à l'espace des solutions réalisables. Cependant, dans le cadre des problèmes d'optimisation combinatoire, en fonction de la méthode choisie ou du problème étudié, l'exploration de l'espace de recherche passe par la considération de solutions dites « irréalisables ». En d'autres termes, lors de la réalisation du module d'optimisation il est possible d'introduire, volontairement ou non, des points qui ne représentent pas des solutions du problème de départ. Ces solutions irréalisables induisent une augmentation de l'espace de recherche. Pourtant, elles peuvent faciliter la recherche, en tant que transitions entre les solutions réalisables. Ainsi, le passage d'une solution réalisable à une autre peut ne pas être possible en fonction des choix algorithmiques effectués, mais il peut être possible de passer de l'une à l'autre en considérant l'ajout de solutions irréalisables. Cette approche est résumée dans la Figure 9 ci-dessous.

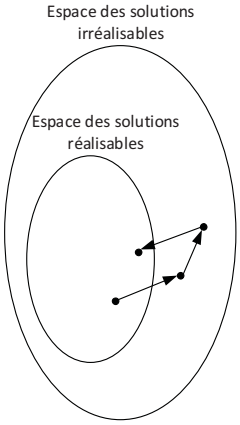

Figure 9 Transitions possibles au sein de l'espace des solutions

Dans la Figure 9, le passage entre les deux solutions de l'espace des solutions réalisables n'est pas possible. Il faut sortir de l'espace des solutions réalisables afin de pouvoir y retourner par la suite. Il faut être vigilant, cependant, à la solution finale retournée par un algorithme d'optimisation. En effet, s'il est autorisé d'explorer l'espace des solutions irréalisables, la ou les solutions finales doivent être réalisables. Pour éviter qu'une solution irréalisable ne soit acceptée par l'algorithme d'optimisation, plusieurs approches peuvent être envisagées. Par exemple, l'ensemble des solutions de ce type peut être exclu de l'espace de recherche a priori ou a posteriori. Si la solution est gardée, la valeur de sa fonction objectif peut être fortement dégradée (ajout d'une valeur « extrême »), afin de pousser l'algorithme à retourner vers l'espace des solutions réalisables. Dans tous les cas l'algorithme d'optimisation doit retourner une ou plusieurs solutions réalisables.

La recherche d'un ensemble de solutions est rencontrée principalement lorsque l'objectif n'est pas d'optimiser un critère mais plusieurs. On parle alors de problèmes multiobjectifs. L'énoncé d'un tel problème est donné ci-dessous :

Définition : Problème d'optimisation combinatoire multiobjectif (Collette and Siarry, 2002)

Soit un problème P et N un ensemble discret et fini. Soit R un sous-ensemble de N constituant les solutions réalisables de P. Soit un ensemble C de critères à optimiser et f_i la fonction objectif associée au i-ème critère de C à optimiser. $\text{opt}\{f_i(S)|S \in R\}_{i\in C}$ est un problème d'optimisation combinatoire multiobjectif.

Dans la définition proposée, le terme 'opt' correspond au sens du critère d'optimisation, à savoir minimiser ou maximiser. En effet, un problème multiobjectif ne cherche pas toujours à minimiser ou maximiser tous les critères. Par exemple, un opérateur de télécommunication qui souhaite maximiser la zone couverte par ses antennes-relais tout en voulant investir le moins d'argent possible dans les infrastructures est confronté à un problème multiobjectif de type min-max. Un autre exemple peut-être de minimiser le coût d'achat d'un logement tout en en maximisant la superficie. Cependant, de nombreux problèmes peuvent être ramenés à la minimisation ou maximisation de tous les critères. Ainsi si on veut maximiser les économies réalisées par l'acquisition des antennes, le problème précédent peut être ramené à un problème de double maximisation. Tout dépend donc des informations qui sont à la disposition du modélisateur. Dans cette thèse un problème où deux critères sont minimisés est proposé. Dans un tel problème l'objectif est de retourner un ensemble de solutions pouvant être utilisées. Ces solutions particulières sont dites non dominées, en opposition avec les solutions dominées qui sont définies ci-dessous.

Définition : Solutions dominées (Collette and Siarry, 2002)

Soit un problème P et N un ensemble discret et fini. Soit un ensemble C de critères à optimiser et f_i la fonction objectif associée au i-ème critère de C à optimiser. Soit R un sous-ensemble de N constituant les solutions réalisables de P. Soient S et S' deux solutions de R. On dit que S domine S' si, pour toute fonction objectif f_i, $f_i(S) \leq f_i(S')$ et s'il existe une fonction f_i pour laquelle $f_i(S) < f_i(S')$. S est dite solution dominante (ou non dominée) et S' est dite dominée.

A partir de la définition énoncée ci-dessus, il peut être déduit qu'il existe au moins une solution non-dominée pour un problème multiobjectif. Le but d'un tel problème consiste alors à déterminer un ensemble de solutions non-dominées entre-elles. Un tel ensemble est appelé front de Pareto et est défini ci-dessous ; un exemple est donné sur la Figure 10.

Définition : Front de Pareto (Collette and Siarry, 2002)

Soit un problème P et N un ensemble discret et fini. Soit un ensemble C de critères à optimiser et f_i la fonction objectif associée au i-ème critère de C. Soit R un sous-ensemble de N constituant les solutions réalisables de P. L'ensemble $FP = \{S \in R|\forall S' \in R, \exists i \in C, f_i(S) < f_i(S')\}$ constitue l'ensemble des solutions non dominées d'un problème d'optimisation multiobjectif et est appelé front de Pareto.

Dans la Figure 10, l'ensemble des points reliés par des segments constituent le front de Pareto des solutions d'un problème biobjectif visant à minimiser les fonctions objectifs f_1,f_2. Ici, la cible optimale, et inatteignable, du problème est l'origine du repère cartésien utilisé. A noter qu'un front de Pareto admet des cibles différentes en fonction du problème multiobjectif traité.

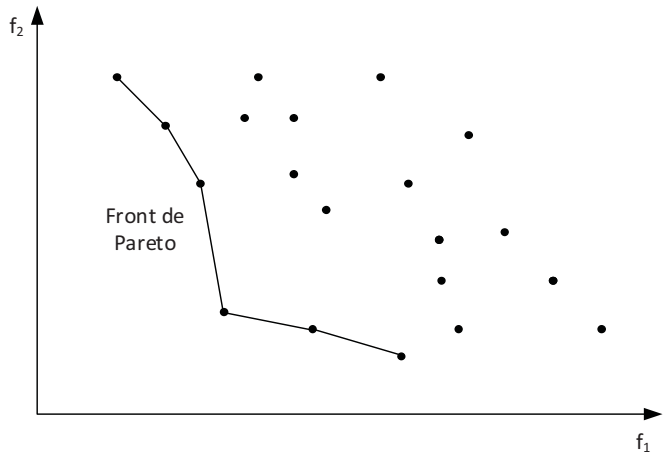

Figure 10 Exemple de front de Pareto d'un problème biobjectif

Les algorithmes utilisés pour résoudre les problèmes d'optimisation combinatoire sont organisés en fonction de la classe de problèmes qu'ils sont à même de résoudre. La détermination de cette classe de problèmes réside dans la modélisation de l'ordinateur mettant en œuvre leur exécution. La machine de Turing est communément admise comme représentation d'un tel ordinateur. Deux types de machines de Turing existent, déterministe et non-déterministe (Turing, 1937). La première exécute les étapes de l'algorithme les unes après les autres, chaque étape étant « déterminée » par l'exécution de l'étape précédente. La machine de Turing non-déterministe quant à elle propose, à une étape donnée, un ensemble d'étapes possibles à exécuter ensuite, sans déterminer explicitement quelle étape doit être exécutée. Comme la notion de complexité induite par l'exécution d'un algorithme est souvent associée au nombre d'actions effectuées dans l'algorithme ; la complexité d'un problème est donc liée à la plus faible complexité des algorithmes qui permettent sa résolution (Toussaint, 2010). De manière générale les algorithmes souhaités sont dits polynomiaux.

Définition : Algorithme polynomial (Garey and Johnson, 1979)

Un algorithme est dit polynomial, ou polynomial-temps, si l'ensemble des actions qu'il effectue suit une fonction polynomiale. Cette complexité est souvent notée à l'aide des notations de Landau $O(n^k)$ *avec* k *un nombre entier et* n *la taille des données.*

On parle également de complexité constante lorsque $k = 0$, de complexité linéaire lorsque $k = 1$. La première classe de problèmes repose directement sur la notion d'algorithme polynomial, et par conséquent sur la définition d'une machine Turing déterministe. Cette classe est appelée classe P.

Définition : Classe P (Garey and Johnson, 1979)

Un problème appartient à P*, et est dit Polynomial, ou Polynomial-temps, s'il peut être résolu en un temps polynomial par une machine de Turing déterministe.*

Un problème appartenant à la classe P est donc un problème solvable à l'aide d'un algorithme polynomial. La recherche d'un élément dans une liste appartient à la classe P. La classe de problèmes qui vient ensuite est la classe NP.

Définition : Classe NP (Garey and Johnson, 1979)

Un problème appartient à la classe NP (Nondeterministic Polynomial time) si une solution du problème peut être vérifiée en un temps polynomial.

Cette classe de problèmes contient la plupart des problèmes d'optimisation combinatoire. Il est important de noter que les problèmes de la classe P, sont également des problèmes de la classe NP, car si un problème peut être résolu en temps polynomial, sa solution peut être vérifiée en temps polynomial, et donc $P \subseteq NP$. Un des problèmes les plus étudiés consiste à savoir si $P \subseteq P$, et donc de savoir si $P = NP$. Ce problème faisant partie des 7 problèmes du prix du millénaire, reste à résoudre. Après les problèmes de la classe NP, viennent les problèmes appartenant à la classe NP-complet. Ces problèmes reposent sur une propriété importante : la réduction polynomiale.

On dit d'un problème P qu'il peut être réduit à un problème P', si les données du problème P peuvent être transformées en données du problème P' à l'aide d'un algorithme polynomial. Ainsi, disposer d'un algorithme polynomial pour P' permettrait de résoudre tout problème P. C'est par cette notion de réduction polynomiale que l'on définit la classe des problèmes NP-complet.

Définition : Classe NP-complet (Garey and Johnson, 1979)

Un problème appartient à la classe NP-complet s'il appartient à la classe NP et si tout problème de NP peut être polynomialement réduit en ce problème.

Les problèmes de la classe NP-complet présentent donc un enjeu primordial car ils constituent les problèmes les plus difficiles de la classe NP, et la découverte d'un algorithme polynomial pour un problème NP-complet permettrait d'utiliser cet algorithme pour résoudre en temps polynomial tous les problèmes dans la classe NP. Actuellement, toute résolution exacte d'un problème NP-complet passe par l'utilisation d'un algorithme dit « exponentiel », en cela qu'il consomme des ressources en temps (et en espace) de manière non polynomiale. De manière générale, un algorithme exponentiel peut être assimilé à un algorithme dont la complexité est $O(2^n)$. Si l'on considère qu'une itération s'effectue sur un ordinateur en 10^{-6}secondes et que le problème est de taille 50, il faudrait plus de 35 ans pour toutes les effectuer. Les capacités actuelles des ordinateurs ne permettent donc pas de tester de manière exhaustive toutes les solutions d'un problème de ce type dès lors que la taille des problèmes croît.

De ce fait, les algorithmes exponentiels peuvent être appliqués à la résolution d'instances de petites tailles. Pour une application industrielle, les tailles de problèmes peuvent être telles qu'il n'est pas envisageable d'énumérer l'ensemble des solutions. Il est alors courant d'utiliser d'autres méthodes. Le paragraphe suivant présente différentes méthodes d'optimisation incluant les méthodes exactes et les méthodes approchées.

5.2. Méthodes d'optimisation

Certaines méthodes d'optimisation importantes sont présentées ci-dessous, parmi lesquelles : la programmation linéaire, les heuristiques de constructions, les méthodes de recherche locale et les métaheuristiques. Ces types de méthodes ont été appliqués pour la résolution des problèmes présentés dans cette thèse.

5.2.1. Programmation linéaire

La programmation linéaire consiste à résoudre un programme linéaire (ou modèle linéaire), généralement de manière exacte. Parmi les programmes linéaires, sont différenciables : les programmes linéaires simples, les programmes linéaires en nombres entiers, et les programmes linéaires mixtes. Tous ces programmes font intervenir des équations et/ou inéquations linéaires (pas de termes quadratiques) qui modélisent les contraintes du problème, et une fonction objectif écrite aussi linéairement.

Les programmes linéaires simples ont des variables dites "continues". Parmi les méthodes exactes permettant leur résolution, il existe la méthode du simplexe ou celle du point intérieur. Si la méthode du point intérieur est polynomiale, la méthode du simplexe est non-polynomiale dans le pire des cas, mais polynomiale en moyenne, et s'avère très performante en pratique.

Les programmes linéaires en nombres entiers (PLNE) font intervenir des variables entières. Ces programmes s'avèrent plus difficiles à résoudre que les précédents alors que l'espace des solutions est plus restreint, en raison de la difficulté à assurer le caractère entier d'une variable.

Les programmes linéaires mixtes (MILP) contiennent des variables continues et entières. La résolution de ces problèmes est proche de celle d'un programme linéaire en nombre entiers.

De nombreux solveurs existent pour résoudre les problèmes évoqués ci-dessus, qu'ils soient commerciaux ou non. Dans les faits, bien que des travaux reposent sur l'utilisation de ces solveurs (tels que CPLEX de IBM (Bruzzone et al., 2012), Gurobi Optimizer (Fang et al., 2011)…) leurs performances ne permettent pas de les utiliser efficacement pour les problèmes d'ordonnancement parce qu'ils font intervenir des contraintes cumulatives mal prises en compte (Salido et al., 2016a).

5.2.2. Les heuristiques

D'après (Le Moigne, 1991), une heuristique est un raisonnement formalisé de résolution de problème dont on tient pour plausible, mais non pour certain, qu'il conduira à la détermination d'une solution satisfaisante du problème.

Les heuristiques de construction sont des exemples classiques de ce type d'approche. Pour l'ordonnancement, un tel algorithme consiste à ordonnancer au fur et à mesure les opérations en appliquant des règles de sélections. Ces heuristiques peuvent être déterministes, ou randomisées afin d'apporter plus de diversité dans les solutions générées. De manière générale, une heuristique est dédiée à un problème et n'est généralement pas optimale. Cependant, certaines heuristiques de construction le sont, comme l'algorithme de Kruskal (Kruskal, 1956) pour le problème de sous-arbre de poids minimum.

Parmi les heuristiques de construction, les algorithmes gloutons sont souvent utilisés soit pour apporter rapidement une solution à un problème soit en tant qu'initialisation pour des algorithmes plus complexes. Les heuristiques de construction permettent de générer une solution en prenant une décision à une étape de l'algorithme sans modifier les décisions prises dans les étapes précédentes. De ce fait, il est fréquent que les décisions prises, et semblant pertinentes à une étape donnée, conduisent à des solutions de qualité médiocre. Dans certains cas, il est parfois préférable de générer une solution de manière complètement aléatoire (en s'appuyant sur une application du second lemme de Borel-Cantelli (Borel, 1909)) que d'utiliser une heuristique conduisant à des solutions de bonne qualité mais très éloignées de la solution optimale empêchant une convergence vers cette dernière. Afin d'améliorer une solution obtenue à l'aide d'une

heuristique, il est courant d'appliquer une méthode de recherche locale dont les principes sont présentés dans la partie suivante.

5.2.3. Les méthodes de recherche locale

Les méthodes de recherche locale ont vocation à améliorer la qualité d'une solution (Hoos and Stützle, 2005). Pour ce faire, elles partent d'une solution donnée et vont appliquer des transformations élémentaires sur cette solution afin de passer à une autre solution (dans le cas où les modifications permettent de rester dans l'espace des solutions réalisables) (Toussaint, 2010). Le passage d'une solution à une autre repose sur la notion de voisinage et de transformation, dont les définitions sont données ci-dessous.

Définition : Transformation

Soit P un problème d'optimisation combinatoire et S une solution du problème donné. Une transformation T est une opération qui modifie la solution S en une nouvelle solution S'.

Définition : Transformation élémentaire

Soit P un problème d'optimisation combinatoire et S une solution du problème donné. Une transformation élémentaire T_e est une opération qui modifie légèrement la solution S en une nouvelle solution S'. Les transformations élémentaires sont des cas particuliers des transformations (sous-ensemble).

Définition : Voisinage

Soit P un problème d'optimisation combinatoire et S une solution du problème donné. Un voisinage de S, noté $V(S)$ est un ensemble de solutions atteignable à partir de S en appliquant une transformation élémentaire Te à S. Les solutions dans $V(S)$ dépendent fortement de la transformation élémentaire utilisée.

Définition : Voisin

Soit P un problème d'optimisation combinatoire et S une solution du problème donné. On appelle voisin toute solution appartenant à l'ensemble $V(S)$.

Une méthode de recherche locale est une méthode appliquant des transformations élémentaires successives sur une solution dans le but d'explorer le voisinage de la solution et d'atteindre une solution de meilleure qualité. Si, considérant une transformation élémentaire, il n'existe pas de solutions dans le voisinage de S ayant une meilleure qualité que celle de S, alors la solution S est dite localement optimale, ou optimum local.

Définition : Optimum local

Soit P un problème d'optimisation combinatoire et S une solution du problème donné. S est dite localement optimale (pour le voisinage considéré) si pour tout $S' \in V(S), f(S) \leq f(S')$ (dans le cadre d'un problème de minimisation).

Ainsi, par fonctionnement itératif, une méthode de recherche locale explore le voisinage d'une solution et sélectionne une solution de meilleure qualité dans le voisinage. En fonction de la méthode de recherche locale utilisée, ce 'voisin' n'est pas obligatoirement celui améliorant le plus le critère d'optimisation. Ainsi, les méthodes de recherche locale diffèrent principalement sur le choix de la solution vers laquelle l'algorithme se dirige. La Figure 11 illustre le passage d'une solution S à une solution S' en appliquant plusieurs fois une transformation élémentaire. Aussi, il est courant d'assimiler le voisinage V à la transformation élémentaire qui permet d'en obtenir les éléments, i.e. $V(S) \cong T_e(S)$. Sur la Figure 11, les ensembles contenant les solutions atteignables par application d'une transformation élémentaire ont une intersection non vide. Ici, trois voisinages sont mis en évidence V_1, V_2 et V_3. Il apparaît dans cette figure que l'intersection de $V(S)$ et $V(T_e(T_e(S)))$ est non vide ce qui suggère l'existence d'une solution à l'endroit de l'intersection. La composition des applications élémentaires est notée $T_e^n(S)$. Pourtant rien n'assure qu'une succession de transformations élémentaires qui passeraient par ce point dans l'optique de limiter le nombre de transformations à deux existe. Ce cas de figure représente l'aspect heuristique des méthodes de recherche locale. Aussi, il apparaît que les solutions peuvent être plus ou moins proches à la fin d'une succession de transformations. Par exemple, la solution notée S' est en terme de distance euclidienne plus proche de S que S ne l'est de toutes les solutions transitoires parcourues.

A noter également que, dans le cas de la Figure 11, toute solution S appartient au voisinage de la solution S', obtenue en appliquant sur S une transformation élémentaire. On dit d'un tel voisinage qu'il est réversible, i.e. $S \in V(S)$.

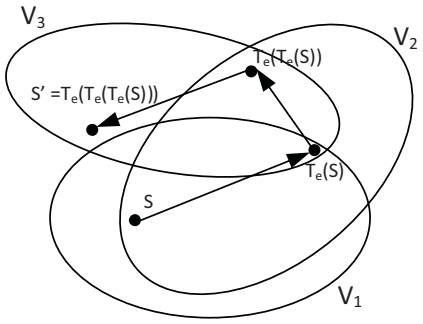

Figure 11 Solutions obtenues par applications successives d'une transformation élémentaire

L'objectif d'une recherche locale est de trouver un optimum local. Ces optima locaux ne sont pas toujours satisfaisants et l'objectif d'un problème d'optimisation est donc de trouver au moins un optimum global. La définition d'un optimum global est donnée ci-dessous.

Définition : optimum global

Soit P un problème d'optimisation combinatoire et R l'ensemble des solutions réalisables. Une solution S est dite optimum global si et seulement si $\forall S' \in R,\ f(y) \geq f(x)$ (i.e. S est meilleure que toutes les solutions de R).

La définition ci-dessus implique qu'un optimum global n'est pas nécessairement unique, ceci dépendant fortement du problème étudié. Les notions d'optima locaux et globaux sont essentielles lorsque l'on souhaite utiliser une recherche locale en tant qu'outil d'optimisation. Cependant, même si la plupart des recherches locales conçues sur la base de voisinages grossiers peuvent atteindre des optima locaux (ceux-ci étant des optima relatifs au voisinage utilisé), il est plus difficile

de concevoir des recherches locales permettant d'atteindre de manière certaine des optima globaux. In fine, la plupart des méthodes de recherche locale ont du mal à converger vers des optima globaux pour les problèmes NP-complets (pour lesquels il est difficile de trouver un optimum global quelle que soit la méthode employée). Ainsi un algorithme d'optimisation reposant sur l'utilisation d'une recherche locale trouvera des optima locaux, mais pas obligatoirement des optima globaux. Un optimum global correspond à la meilleure solution, non pas d'un problème industriel, mais de la modélisation qu'on en a faite et rien n'assure qu'un optimum global retourné par un algorithme d'optimisation puisse être appliqué sur le cas réel ayant permis la construction du modèle. Il ressort de ce constat que de bons optima locaux sont tout aussi intéressants qu'un optimum global. Cependant, la conception de recherches locales efficaces reste primordiale. Or, les recherches locales reposent sur la notion de voisinages, et certains sont plus adaptés que d'autres pour découvrir des solutions de bonne qualité. Pour atteindre cet objectif, certaines propriétés (en plus de la réversibilité énoncée plus haut) des voisinages sont à rechercher, telle que l'accessibilité dont une définition est donnée ci-dessous, suivie de la définition d'accessibilité faible.

Définition : accessibilité

Un voisinage V est dit accessible s'il est possible de passer de toute solution S de l'espace des solutions réalisables R à tout autre solution S' par un nombre fini de compositions de transformations élémentaires. $\forall S, S' \in R^2, \exists n \in N, S' = T_e^n(S)$.

Définition : accessibilité faible

Un voisinage V est dit faiblement accessible s'il est possible de passer de toute solution S de l'espace des solutions réalisables R à une solution globalement optimale S' par un nombre fini de compositions de transformations élémentaires. $\forall S \in R, \exists S^* \in R, \forall S' \in R, \exists n \in N, S^* = T_e^n(S), f(S^*) \leq f(S')$.

Deux types de voisinages existent : les voisinages généraux et les voisinages guidés. Les voisinages généraux sont basés sur des transformations élémentaires qui sont appliquées sans règles spécifiques ; ce peut être des permutations, des insertions, … Les voisinages guidés sont eux plus spécifiques au problème et reposent sur des restrictions des voisinages généraux aux éléments présentant normalement les meilleures caractéristiques du voisinage. Dans l'idéal, on voudrait que la restriction d'un voisinage V à un sous-ensemble V' de celui-ci implique que tous les éléments de $V' \backslash V$ soient meilleurs au sens de la fonction objectif considérée. Cela étant, il est fréquent de considérer une restriction contenant « en moyenne » des éléments de meilleure qualité que ceux présents dans $V \backslash V'$.

Si les voisinages guidés tendent à générer des solutions voisines de bonne qualité, la cardinalité d'un tel ensemble consiste en un compromis : un voisinage générant trop de solutions va permettre de parcourir plus d'espace mais elles risquent d'être de mauvaise qualité, et un voisinage restreint risque de produire peu de solutions, normalement de bonne qualité, mais pouvant empêcher la convergence vers un optimum global. Pire, un voisinage trop petit peut ne pas contenir la solution optimale. Il faut donc un compromis entre la taille du voisinage et la qualité des solutions qu'il contient.

Une méthode de recherche locale repose donc sur l'utilisation d'un voisinage de taille suffisante pour maximiser la probabilité de générer une solution de bonne qualité. Le choix de la solution à garder après une itération d'un algorithme de recherche locale est également important : soit le choix est déterministe, soit une solution du voisinage est choisie aléatoirement. Les recherches locales prenant des décisions déterministes présentent l'avantage d'avoir le même

comportement : une recherche locale de ce type appliquée deux fois d'affilée sur la même solution de départ produira le même résultat. Cela étant, l'ordre dans lequel les solutions du voisinage sont explorées peut conduire à des qualités de solutions différentes en fonction de l'implémentation de la recherche locale. L'ordre dans lequel le voisinage est exploré fait donc partie de la définition d'une recherche locale déterministe. En opposition à cette approche, les recherches locales randomisées vont accepter une solution de manière aléatoire sans avoir forcément fini d'explorer l'espace des solutions envisageables. Ainsi, chaque réplication d'une telle recherche locale sur une solution de départ peut conduire à des solutions de qualités très différentes.

Les recherches locales, qui tendent à converger vers une solution localement optimale à partir d'une solution donnée, sont également appelées algorithmes d'*intensification*. Leur seule utilisation ne permet bien souvent pas de converger vers la solution optimale d'un problème. Les méthodes de recherches locales sont donc fréquemment associées à des algorithmes dits de *diversification*, qui visent à explorer l'espace des solutions en cherchant des zones non encore visitées. Ces algorithmes portent le nom de métaheuristiques et sont présentés dans la section suivante.

5.2.4. Les métaheuristiques

En sciences, le terme 'méta' désigne généralement un niveau d'abstraction supérieur. Ainsi, là où les heuristiques sont généralement dédiées à un problème, comme mentionné précédemment, les métaheuristiques sont des schémas génériques de résolution de problèmes. Plusieurs de ces méthodes sont très connues et utilisés dans de nombreux problèmes d'optimisation combinatoire tels que le voyageur de commerce ou les problèmes d'ordonnancement. Les métaheuristiques ne sont pas toujours conçues à la suite de l'étude d'un problème d'optimisation, mais peuvent être le résultat d'observations sur des phénomènes naturels. Par exemple, l'algorithme génétique est sûrement la métaheuristique la plus connue car elle repose sur un phénomène biologique lié à l'évolution d'une population d'individus à travers les brassages et les mutations génétiques. De nombreuses métaheuristiques ont été proposées par le passé, et de nouvelles continuent d'émerger. Ceci est en partie lié à la nécessité, ressentie, de proposer de nouvelles méthodes de résolution plutôt que d'utiliser les anciennes, quitte à inonder la littérature de métaheuristiques aux noms saugrenus (Sörensen, 2012). Quelques-unes des métaheuristiques fréquemment utilisées dans la littérature, sont ici présentées avec leurs algorithmes de principe. Ces métaheuristiques présentent l'avantage d'être facilement mises en œuvre.

5.2.4.1. Variable Neighbourhood Search (VNS)

La recherche par voisinage variable a été présentée par (Mladenović and Hansen, 1997). Elle repose sur l'utilisation de plusieurs voisinages afin d'explorer au mieux l'espace des solutions. En effet, comme il a été mentionné précédemment, les voisinages ne produisent pas tous les mêmes résultats et il est donc intéressant d'utiliser un ensemble de voisinages permettant d'explorer au mieux l'espace de recherche, sachant que deux voisinages reposant sur des transformations élémentaires proches seront moins intéressants que deux voisinages avec des transformations élémentaires très différentes. La première étape de cette métaheuristique est de générer une solution initiale (à l'aide d'une heuristique de construction ou de manière aléatoire). Ensuite, la phase d'exploration de chaque voisinage (appelée shaking par les auteurs) est appliquée. Chaque voisinage est exploré l'un après l'autre afin de créer un voisin de la solution actuelle. Une recherche locale est généralement appliquée à la suite de l'obtention d'un voisin et si la solution est meilleure, la solution courante est actualisée. L'Algorithme 1 présente le principe de cette métaheuristique.

Algorithme 1 : Algorithme de principe du VNS

Sortie
S Meilleure solution rencontrée par l'algorithme
Variables
S' Solution temporaire

Début
1. Initialiser la solution S ;
2. **TANT QUE** critère d'arrêt non satisfait **FAIRE**
3. $i \coloneqq 1$;
4. **TANT QUE** tous les voisinages n'ont pas été appliqués **FAIRE**
5. Choisir S' dans le i-ème voisinage V_i de S ; //shaking
6. Appliquer une recherche locale sur S' ;
7. **SI** $f(S') < f(S)$ **ALORS**
8. $S \coloneqq S'$;
9. $i \coloneqq 1$;
10. **SINON**
11. $i \coloneqq i + 1$;
12. **FIN SI**
13. **FIN TQ**
14. **FIN TQ**
15. Retourner S ;

Fin

Des variantes de cette métaheuristique existent en modifiant la façon dont les voisinages sont parcourus. Ainsi, il peut être envisagé durant l'étape de shaking de choisir, non pas une solution aléatoirement, mais la meilleure solution considérant le voisinage utilisé à une itération donné. La métaheuristique ainsi construite est appelée VND, pour Voisinage à Descente Variable (Variable Neighbourhood Descent). Aussi, plutôt que d'actualiser la solution courante lorsqu'une meilleure solution est rencontrée, il est possible d'accepter une solution dégradée avec une probabilité donnée. La métaheuristique s'approche alors du Recuit Simulé (Simulated Annealing) qui est présenté ci-dessous.

5.2.4.2. Simulated Annealing (SA)

Cette métaheuristique, également appelée Recuit Simulé, a été développée indépendamment par (Černỳ, 1985; Kirkpatrick et al., 1983). Inspirée par l'industrie métallurgique, cette métaheuristique introduit la notion de température permettant d'accepter ou non des voisins de moins bonne qualité que la solution courante. L'Algorithme 2 présente le principe de cette métaheuristique et est inspiré de (Boussaïd et al., 2013). Dans cet algorithme la notion d'équilibre (ligne 4) peut représenter par exemple un nombre d'itérations pendant lesquelles aucune amélioration de la solution n'est rencontrée. Au début de l'algorithme la température est une valeur élevée, ce qui implique d'accepter une solution dégradée avec une probabilité élevée. Cette température décroît au fur et à mesure des itérations de l'algorithme, ainsi il est peu probable d'accepter une solution de moins bonne qualité que la solution courante après itérations successives de l'algorithme. La vitesse à laquelle décroît la température a aussi un impact direct sur le comportement de la métaheuristique. Des variantes du recuit simulé existent dans la littérature et jouent sur la vitesse de décroissance, sur les probabilités d'acceptation des solutions dégradées, ou sur le ratio d'amélioration des solutions.

Algorithme 2 : Algorithme de principe du Recuit Simulé

Sortie
S^* Meilleure solution rencontrée par l'algorithme
Variables
S, S' Solutions temporaires

Début
1. Initialiser la solution S ; Initialiser la température T ;
2. **TANT QUE** critère d'arrêt non satisfait **FAIRE**
3. Choisir S' dans voisinage V de S ;
4. **TANT QUE** l'équilibre n'est pas atteint **FAIRE**
5. **SI** $f(S') < f(S)$ **ALORS**
6. $S \coloneqq S'$; Garder la meilleure solution dans S^* ;
7. **SINON**
8. **SI** $e^{\frac{f(S)-f(S')}{T}} \geq p$ **ALORS**
9. $S \coloneqq S'$;
10. **FIN SI**
11. **FIN SI**
12. **FIN TQ**
13. Faire décroitre T ;
14. **FIN TQ**
15. Retourner S^* ;

Fin

5.2.4.3. Tabu Search (TS)

La recherche tabou est une métaheuristique populaire, proposée par (Glover, 1986). Cette métaheuristique présentée dans l'Algorithme 3 repose sur une liste, dite liste tabou, de mouvements interdits pour forcer l'algorithme à explorer des zones de l'espace non déjà visitées. La liste peut gérer différentes informations en fonction du comportement souhaité. Elle peut stocker les derniers mouvements effectués, ou les dernières solutions visitées (peu utilisé en pratique car trop coûteux en espace), … Elle fonctionne comme une liste circulaire, et les éléments qu'elle contient finissent par être exclus de la liste et redeviennent disponibles pour l'algorithme après un certain temps. De ce fait, la taille de la liste est un paramètre délicat à fixer : une liste trop petite risque de ne rien apporter à la recherche, en permettant la réintégration trop rapide d'éléments interdits dans la liste des éléments autorisés ; une liste trop importante peut empêcher l'exploration des solutions.

Algorithme 3 : Algorithme de principe de la recherche Tabou

Sortie
S^* Meilleure solution rencontrée par l'algorithme
Variables
S, S' Solutions temporaires
TL Liste tabou

Début
1. Initialiser la solution S ; $TL \coloneqq \emptyset$;
2. **TANT QUE** critère d'arrêt non satisfait **FAIRE**
3. $S \coloneqq$ meilleure solution dans voisinage V de S avec une transformation élémentaire $T_e \notin TL$;
4. $TL \coloneqq TL \cup T_e$;
5. $S^* \coloneqq$ Meilleure solution rencontrée;
6. **FIN TQ**
7. Retourner S^* ;

Fin

5.2.4.4. Evolutionary Local Search (ELS)

La recherche locale évolutionnaire a été proposée par (Wolf and Merz, 2007). Dans cette métaheuristique, une solution est initialement construite. Ensuite, un ensemble de solutions voisines de la solution courante est généré grâce à une transformation élémentaire (lignes 4 à 10 de l'Algorithme 4). Chacune de ces solutions obtenues est améliorée à l'aide d'une recherche locale. La meilleure des solutions ainsi rencontrées devient la solution courante. Le processus est réitéré jusqu'à ce qu'un critère d'arrêt soit rencontré. Cette métaheuristique est une évolution de l'ILS (Iterated Local Search) proposée par (Lourenço, Martin, and Stützle 2003), cette dernière ne produisant qu'un seul voisin à chaque itération de la boucle 1. L'algorithme de principe de cette métaheuristique est donné ci-après. Dans l'Algorithme 4 , la nouvelle solution courante devient la meilleure solution voisine rencontrée dans la boucle 2, quelle que soit sa qualité. Des alternatives existent, comme de choisir la solution voisine uniquement si elle améliore la solution courante, ou avec une probabilité donnée.

Algorithme 4 : Algorithme de principe de l'ELS

Sortie
S^* Meilleure solution rencontrée par l'algorithme
Variables
S, S', S'' Solutions temporaires

Début
1. Initialiser la solution S ;
2. **POUR** $iterELS \coloneqq 1$ **A** $maxELS$ **FAIRE** // boucle 1
3. $f(S'') \coloneqq \infty$;
4. **POUR** $iterVoisin \coloneqq 1$ **A** $maxVoisin$ **FAIRE** // boucle 2
5. $S' \coloneqq$ solution dans le voisinage V de S;
6. $S' \coloneqq$ solution retournée par une recherche locale ;
7. **SI** $f(S') < f(S'')$ **ALORS**
8. $S'' \coloneqq S'$; Garder la meilleure solution dans S^* ;
9. **FIN SI**
10. **FIN POUR**
11. $S \coloneqq S''$;
12. **FIN POUR**
13. Retourner S^* ;

Fin

5.2.4.5. Greedy Randomized Adaptive Search Procedure (GRASP)

Le GRASP a été proposé par (Feo and Resende, 1995) et constitue une métaheuristique à démarrages multiples. Elle consiste à générer une solution de départ à l'aide d'une heuristique de construction randomisée. Cette solution est ensuite améliorée par une recherche locale. L'algorithme répète cette phase jusqu'à ce qu'un critère d'arrêt soit rencontré. La facilité à mettre en œuvre cette métaheuristique l'a rendue très populaire dans les articles parus ces dernières années. L'Algorithme 5 présente le principe du GRASP.

Algorithme 5 : Algorithme de principe du GRASP

Sortie
S^* Meilleure solution rencontrée par l'algorithme
Variables
S Solution temporaire

Début
1. **TANT QUE** critère d'arrêt non satisfait **FAIRE**
2. $S \coloneqq$ solution produite par une heuristique de construction randomisée ;
3. $S \coloneqq$ solution retournée par une recherche locale ;
4. $S^* \coloneqq$ meilleure solution rencontrée ;
5. **FIN TQ**
6. Retourner S^* ;

Fin

5.2.4.6. GRASP×ELS

Les métaheuristiques peuvent également découler d'hybridations entre plusieurs métaheuristiques. C'est le cas du GRASP×ELS proposé par (Prins, 2009). Cette métaheuristique a depuis lors été utilisée dans plusieurs travaux de recherche. L'algorithme de principe de cette métaheuristique consiste en la concaténation des deux métaheuristiques qui le composent, et est exprimé par l'Algorithme 6. L'avantage d'une telle approche réside dans l'apport du GRASP en tant que méthode de diversification des solutions explorées, là où l'ELS est une méthode d'intensification qui permet de mieux converger vers une solution de bonne qualité.

Algorithme 6 : Algorithme de principe du GRASP×ELS

Sortie
S^* Meilleure solution rencontrée par l'algorithme
Variables
S, S', S'' Solutions temporaires

Début
1. **TANT QUE** critère d'arrêt non satisfait **FAIRE**
2. $S \coloneqq$ solution produite par une heuristique de construction randomisée ;
3. $S \coloneqq$ solution retournée par une recherche locale ;
4. **POUR** $iterELS \coloneqq 1$ **A** $maxELS$ **FAIRE** // boucle 1
5. $f(S'') \coloneqq \infty$;
6. **POUR** $iterVoisin \coloneqq 1$ **A** $maxVoisin$ **FAIRE** // boucle 2
7. $S' \coloneqq$ solution dans le voisinage V de S;
8. $S' \coloneqq$ solution retournée par une recherche locale ;
9. **SI** $f(S') < f(S'')$ **ALORS**
10. $S'' \coloneqq S'$; Garder la meilleure solution dans S^* ;
11. **FIN SI**
12. **FIN POUR**
13. $S \coloneqq S''$;
14. **FIN POUR**
15. **FIN TQ**
16. Retourner S^* ;

Fin

Les métaheuristiques présentées jusqu'à maintenant consistent à conserver une seule solution pendant leur exécution. Ce sont des métaheuristique « basées solutions uniques » (Boussaïd et al., 2013) et également appelées « méthodes à trajectoire ». Ces méthodes sont opposées aux

métaheuristiques à population qui manipulent un ensemble d'individus au cours du processus d'optimisation. Deux de ces méthodes sont proposées, l'une applicable dans le cadre des problèmes mono-objectifs, et la seconde dans le cas des problèmes multiobjectifs.

5.2.4.7. Memetic Algorithm (MA)

Un algorithme mémétique (Moscato, 1989) est une évolution du célèbre algorithme évolutionnaire dit algorithme génétique (Holland, 1975). Comme tout algorithme évolutionnaire, il s'inspire du domaine de la biologie en mimant les mécanismes d'évolution des espèces. Une métaheuristique évolutionnaire repose sur l'utilisation d'une population dont les éléments sont appelés individus. Au début de l'algorithme, une population initiale est construite. A chaque itération de l'algorithme, une nouvelle génération de la population est obtenue. Cette génération est constituée d'individus de la génération précédente et de nouveaux individus, les enfants, issus du couplage (croisement) entre deux individus de l'ancienne population. Un ou deux enfants sont en général conçus à partir de deux parents. Lors du couplage, une partie de l'information génétique des parents est transmise aux enfants. Des perturbations (mutations) peuvent également intervenir, modifiant le génome des enfants obtenus après la phase de croisements. La taille de la population est maintenue par des algorithmes de sélection qui permettent également d'obtenir des individus plus performants au fil des générations. En plus des mécanismes classiques de l'algorithme génétique, un algorithme mémétique contient également une recherche locale visant à améliorer la qualité des individus. Le principe de cette métaheuristique est donné par l'Algorithme 7.

Algorithme 7 : Algorithme de principe du MA

Sortie
S^* Meilleure solution rencontrée par l'algorithme
Variables
Pop, $PopC$ Populations utilisées dans l'algorithme

Début
1. Générer une population initiale Pop ;
2. **TANT QUE** critère d'arrêt non satisfait **FAIRE**
3. $PopC \coloneqq$ population des enfants obtenus après croisement d'individus de Pop;
4. $PopC \coloneqq$ individus de $PopC$ après mutation ;
5. $PopC \coloneqq$ recherche locale sur les individus de $PopC$;
6. $Pop \coloneqq$ Individus sélectionnés à partir de $Pop \cup PopC$
7. Garder la meilleure solution dans S^* ;
8. **FIN TQ**
9. Retourner S^* ;

Fin

5.2.4.8. Non-dominated Sorted Genetic Algorithm (NSGA-II)

Le NSGA-II est une métaheuristique proposée par (Deb et al., 2002), rendue célèbre par sa facilité d'implémentation à partir d'un algorithme génétique. La particularité derrière cette métaheuristique est qu'elle est capable de retourner un front de Pareto des solutions d'un problème multiobjectif. Trois procédures sont utilisées à cette fin : le tri non-dominé, le calcul des marges et la sélection.

Le tri non-dominé consiste à assigner un rang/front à chaque solution en commençant par les solutions non dominées qui appartiennent au front F_1. Ces solutions sont ensuite retirées de la population courante. Un front F_2 des solutions non dominées considérant la population réduite est alors construit. Le processus est réitéré jusqu'à ce que chaque solution appartienne à un front.

Un exemple de différents fronts est présenté dans la Figure 12. Le calcul des marges, quant à lui, consiste à assigner une valeur représentant l'espace couvert par une solution au sein d'un front. Le cuboïde ainsi construit est également représenté par un rectangle en pointillé sur la Figure 12. La valeur du critère pour un problème à deux objectifs peut être obtenue en utilisant la formule suivante :

$$marge(F(k)) = \frac{f_1\big(F(k+1)\big) - f_1(F(k-1))}{f_1^{max} - f_1^{min}} + \frac{f_2\big(F(k-1)\big) - f_2(F(k+1))}{f_2^{max} - f_2^{min}}$$

Dans cette formule, $F(k)$ désigne la k-ème solution du front F. A noter que la marge couverte par une solution à l'extrémité d'un front est infinie. Les valeurs f_i^{min} et f_i^{max} correspondent aux valeurs extrêmes des solutions du front F considéré.

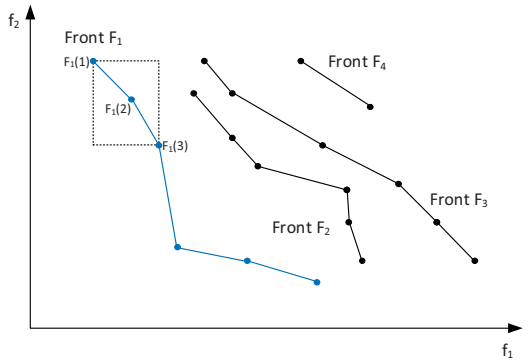

Figure 12 Principe de numérotation des fronts et représentation des marges

Le *rang* et la *marge* associés à une solution sont utilisés dans les processus de sélection des parents. Ne sont gardés que ceux qui présentent le meilleur rang. S'ils appartiennent au même front, on ne conserve que la solution ayant la marge la plus grande. L'algorithme de principe du NSGA-II est présenté dans l'Algorithme 8.

Algorithme 8 : Algorithme de principe du NSGA-II

Sortie

S^* Meilleure solution rencontrée par l'algorithme

Variables

Pop, $PopC$ Populations utilisées dans l'algorithme

Début

1. Générer une population initiale Pop de n individus ;
2. **TANT QUE** critère d'arrêt non satisfait **FAIRE**
3. $PopC \coloneqq$ population des enfants obtenus après croisement d'individus de Pop;
4. $PopC \coloneqq$ individus de $PopC$ après mutation ;
5. $PopC \coloneqq Pop \cup PopC$ //on fait grossir volontairement la taille de $PopC$
6. Tri non dominé des individus de $PopC$ en fonction de leurs critères;
7. Calculer les marges de chaque individu de chaque front;
8. $Pop \coloneqq \emptyset$;
9. Sélection des individus ajoutés dans Pop;
10. **FIN TQ**
11. Retourner $Pop.front(1)$;

Fin

5.3. Qualité d'une méthode d'optimisation

La section 5.2. présente de nombreux algorithmes, mais ils n'ont pas tous le même comportement lorsqu'ils sont confrontés à un problème d'optimisation. Ainsi, un algorithme peut être performant pour résoudre un problème et médiocre sur un autre. Suivant les objectifs du problème d'optimisation, et la qualité recherchée, certains algorithmes peuvent être proposés. Par exemple, si le temps de calcul n'est pas un problème, un algorithme convergeant lentement vers une solution de bonne qualité sera préféré à un autre algorithme présentant de bonnes solutions rapidement mais ne permettant pas d'obtenir le résultat souhaité par l'industriel. Ainsi, pour choisir quelle méthode implémenter, les algorithmes doivent pouvoir être comparés.

Cette section contient deux parties. Dans la première partie, une définition des mesures de comparaison des algorithmes mono-objectifs. La seconde est dévolue aux algorithmes multiobjectifs.

5.3.1. Mesures de qualité des problèmes mono-objectifs

De manière générale, les problèmes d'optimisation reposent sur l'utilisation de bornes inférieures et supérieures, une solution optimale étant obtenue lorsque la borne inférieure est égale à la borne supérieure.

Définition : borne inférieure

On appelle borne inférieure, et on note inf*, la valeur telle qu'aucune solution d'un problème d'optimisation ne peut avoir une fonction objective meilleure que* inf.

Définition : borne supérieure

On appelle borne supérieure, et on note sup*, la valeur telle qu'une solution optimale d'un problème d'optimisation a une fonction objectif inférieure à* sup.

Il est alors courant de définir la qualité d'une solution d'un problème à partir de la borne inférieure et supérieure de ce problème. On parle alors de déviation par rapport à la borne inférieure (supérieure).

Définition : Déviation

Soit S *la solution retournée par un algorithme d'optimisation combinatoire pour une instance* I *d'un problème* P. *Soit* inf *une borne inférieure de la fonction objectif* f *pour l'instance* I. *On a* $Dev_I(S) = (f(S) - inf)/inf$ *la déviation de la solution* S *par rapport à la borne inferieure* inf.

Le calcul de la déviation peut être fait par rapport à une borne supérieure, ou par rapport à une autre solution avec laquelle on souhaite faire une comparaison, sachant que n'importe quelle solution est une borne supérieure du problème traité. Evaluer la qualité d'un algorithme sur une seule instance présente peu d'intérêt en pratique, il est donc préférable de connaître la qualité de la méthode utilisée sur un ensemble d'instances. Ces instances peuvent être disponibles dans la littérature, lorsqu'un problème est très étudié, ou doivent être générées. Ce dernier cas est rendu fréquent par l'émergence de nombreux problèmes nouveaux pour lesquels aucune instance n'existe. Il est alors possible de faire évoluer d'anciennes instances reconnues, si le problème repose sur un problème classique de la littérature, ou alors elles peuvent être générées entièrement. Aussi,

du fait de l'aspect randomisé de certaines méthodes proposées, il est important d'effectuer plusieurs réplications pour un algorithme afin d'extraire une connaissance sur son comportement moyen.

Un autre critère très utilisé concerne le temps d'exécution des méthodes conçues. Une méthode plus rapide et présentant des solutions au moins aussi bonne qu'une autre méthode peut présenter un avantage certain. En pratique cette information doit être mise en perspective du fait de l'évolution constante des performances de calcul pour lesquelles il est difficile d'avoir des informations précises, et également des compétences du développeur ou encore du langage de programmation utilisé (Yuan et al., 2013). Cependant, s'il est délicat de comparer les performances d'algorithmes dont l'exécution a été faite sur des machines différentes, la comparaison des temps de calcul pour des méthodes implémentées sur un même ordinateur est plus représentative de la qualité d'un algorithme.

5.3.2. Mesures de qualité des problèmes multiobjectifs

Dans le paragraphe précédent, des critères communément admis dans la littérature pour les problèmes mono-objectifs ont été présentés. Il existe également plusieurs critères pour évaluer la performance d'algorithmes multiobjectifs. Par exemple, le nombre de solutions appartenant au front de Pareto obtenu est une information importante. En effet, un nombre important de solutions autorisera un choix plus large, alors qu'un faible nombre limitera les décisions qui peuvent être envisagées. Pour un algorithme M donné, ce critère sera noté NS^M dans la suite.

La répartition des solutions dans un front est également un critère important, car si les solutions sont toutes localisées au même endroit, les possibilités de choix sont limitées ; il est donc préférable d'avoir un front de Pareto avec une bonne dispersion des solutions.

Définition : Dispersion

Soit PF un front de Pareto obtenu par un algorithme d'optimisation combinatoire pour un problème P. On note DS la dispersion des solutions au sein du front PF, et on a $DS = \frac{1}{\overline{D}}\sqrt{\frac{1}{NS}\sum_{i \in PF}(D_i - \overline{D})^2}$, où D_i représente la distance euclidienne entre une solution i et son plus proche voisin dans le front, et D la distance moyenne entre les solutions du front.

A partir de la définition ci-dessus, il ressort qu'une faible valeur du critère DS suggère un front de Pareto dont les solutions sont bien réparties (Tan et al., 2006). Les deux critères énoncés permettent d'avoir des informations sur un algorithme donné. Cependant, comparer deux fronts de Pareto obtenus à l'aide de deux algorithmes différents est plus complexe. Ce critère est représenté par la notion de degré d'optimalité.

Définition : Degré d'optimalité

Soit un problème multiobjectif P et un algorithme de résolution M. On note NS^{M^*} le nombre de solutions de M qui ne sont dominées par aucune solution d'un autre algorithme.

On note OD le degré d'optimalité, et on a $OD = \frac{NS^{M^*}}{NS^M}$.

D'autres critères de performance peuvent être utilisés dans la littérature, tel que l'hypervolume, qui représente la surface couverte par le front de Pareto. Dans cette thèse les principaux critères sont les trois énoncés ci-dessus.

6. Conclusion

Dans ce chapitre, différents aspects de l'optimisation pour la planification des systèmes de production industrielle sont introduits. Ces aspects vont de la modélisation des problèmes aux outils de résolution utilisés et sont découpés en quatre grandes parties.

Dans les deux premières parties, la difficulté de modéliser les systèmes de production a été mise en évidence, et certains problèmes théoriques très répandus dans la littérature et qui ont fait l'objet de nombreuses modélisations ont été présentés.

Ensuite, différents types de modèles ont été exposés, tels que les modèles mathématiques et les modèles de simulation. Les démarches de modélisation et d'optimisation sont traitées séparément dans ce chapitre, même si de nombreuses interactions existent entre les deux. La démarche de modélisation consiste à concevoir à partir d'un problème réel, une abstraction de ce problème qui va être plus facile à manipuler. A ce titre, il est préférable de partir d'un modèle existant et d'ajouter des contraintes additionnelles afin de pouvoir réutiliser des méthodes existantes dans la littérature pour le problème support. La démarche d'optimisation repose sur la séparation entre deux modules : un module d'optimisation et un module d'évaluation des performances. Le module d'optimisation manipule des solutions partielles et vise à découvrir des solutions de bonnes qualités. Le module d'évaluation quant à lui, transforme une solution partielle en solution évaluée (complète), ce qui permet de prendre en compte les différentes données du problème séparément du module d'optimisation, et permet de rajouter de l'information au fur et à mesure en fonction du niveau de détail souhaité.

En utilisant les apports de ces deux démarches, un outil d'aide à la décision pour la planification des systèmes de production industrielle peut être élaboré. Cet outil consiste en un compromis entre la qualité des solutions recherchées et la facilité d'obtenir ces solutions. Un problème trop simplement modélisé sera plus facile à résoudre mais les solutions peuvent être inutilisables, voire conduire à une dégradation du système ; tandis qu'une modélisation trop fine peut conduire à des solutions très éloignées de la solution optimale à cause de la prise en compte de nombreuses contraintes difficiles à traiter par l'outil d'optimisation, ce qui conduit à une explosion combinatoire.

En appliquant la démarche de modélisation au problème ECOTHER, il a été choisi de s'intéresser principalement aux problèmes de Job-shop avec contraintes énergétiques et au Job-shop Flexible. Le premier modèle consiste en une extension du problème de Job-shop, alors que le second est un problème classique de la littérature.

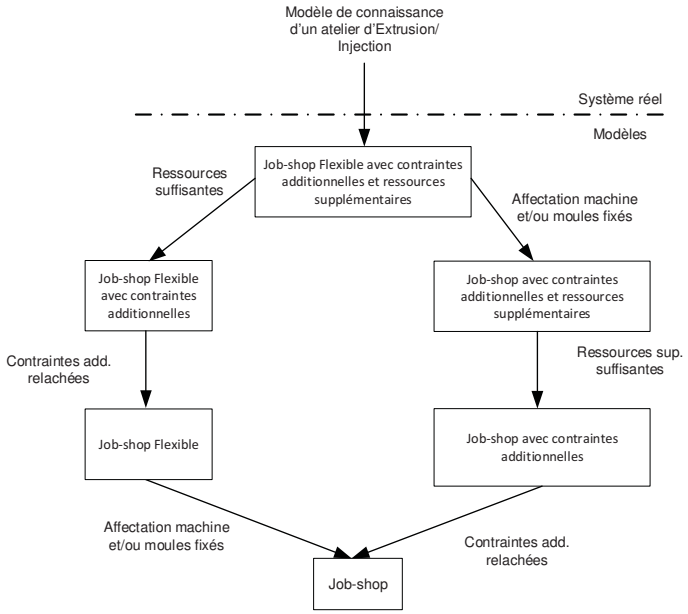

Figure 13 Démarche de modélisation appliquée au problème ECOTHER

Pris indépendamment, les ateliers d'un système de production injection/extrusion peuvent être modélisés différemment. Cependant, la nécessité d'intégrer la consommation énergétique du système de production dans son ensemble requiert une vision plus générique du système de production. Aussi, si le projet s'oriente principalement vers les transformateurs de caoutchouc, l'approche se doit d'être relativement flexible pour être adaptée par la suite à d'autres systèmes de productions. Ainsi, le problème ECOTHER se ramène à un problème de Job-shop Flexible avec contraintes additionnelles et ressources supplémentaires. Ces contraintes modélisent de nombreuses interactions entre les entités du système de production, que ce soient des temps de préchauffage des machines, la consommation énergétique, les ressources humaines, ou la gestion des matières premières. La Figure 13 est une représentation de la démarche appliquée au problème ECOTHER. Le schéma se lit du haut vers le bas. En haut est présenté le modèle de connaissances de l'atelier d'injection/extrusion. Ce modèle repose sur un modèle théorique connu qui est présenté dans sa forme simplifiée en bas du schéma : le problème de Job-shop. Différents problèmes théoriques sont présents entre le sommet et la base de la figure. Plus on est haut dans la figure plus le nombre de contraintes est important, et plus le problème est difficile à résoudre. Ainsi, chaque flèche permet de ramener un problème à une de ses simplifications en faisant une hypothèse simplificatrice. Si le problème le plus difficile est le problème de Job-shop Flexible avec contraintes additionnelles, le manque de travaux sur le Job-shop avec contraintes énergétiques ainsi que l'utilisation du Job-shop Flexible en tant que modèle support nous a entraînés vers l'étude de ces problèmes. Le chapitre suivant présente nos propositions pour les ateliers à cheminements multiples, dans le but d'optimiser la planification des processus industriels d'un atelier d'injection/extrusion pour lequel les contraintes additionnelles sont relâchées.

Chapitre II : Optimisation d'ateliers à cheminements multiples : le problème de Job-shop

Ce chapitre présente le problème du Job-shop et les principales techniques utilisées pour la résolution de ce dernier, telles que le graphe conjonctif-disjonctif, le chemin critique et différents systèmes de voisinages. Plus particulièrement, une modélisation linéaire en nombres entiers et une modélisation sous la forme d'un graphe conjonctif-disjonctif sont présentés. L'application d'une métaheuristique, le GRASP×ELS, est proposée afin de montrer la réutilisabilité des méthodes et outils présentés ici. Ces derniers sont utilisés tout au long de cette thèse.

Sommaire

Liste des figures

Liste des tableaux

Liste des algorithmes

1. Introduction

Dans ce chapitre, les techniques communément utilisées par la communauté scientifique pour la résolution du problème de Job-shop (JSP) sont présentées. Celui-ci est un problème classique de la littérature, et est toujours d'actualité (Cheng et al., 2016 ; Qing-dao-er-ji and Wang, 2012). Le Job-shop permet notamment de généraliser de nombreux problèmes classiques d'ordonnancement comme les problèmes à une machine ou le Flow-shop. Le Job-shop est également un sous-problème du Job-shop Flexible (Brucker and Schlie, 1990), ce dernier intégrant en plus du problème d'ordonnancement, un problème d'affectation. Le Job-shop est donc au cœur du Job-shop Flexible. Il semble donc important de s'intéresser au Job-shop dans un premier temps, ainsi qu'à certaines de ses extensions par la suite.

Ce chapitre est composé de quatre parties. Dans la première, le problème de Job-shop (section 2) est présenté de manière générale et formalisé sous la forme d'un programme linéaire. La deuxième partie (section 3) présente deux classes et deux représentations pour l'ordonnancement. La troisième partie (section 4) consiste en un état de l'art du problème de Job-shop. Cette section présente plus particulièrement le graphe conjonctif-disjonctif, et différentes méthodes utilisées pour résoudre le problème. La quatrième partie (section 5) consiste en la présentation d'une métaheuristique pouvant être appliquée sur le problème de Job-shop. Une synthèse des résultats obtenus sur des instances classiques de la littérature est également fournie. La dernière partie (section 6) conclut ce chapitre.

2. Définition du problème de Job-shop

Soit un problème de Job-shop. Soit un ensemble J de r jobs $J_1, J_2, \dots J_r$ et M un ensemble de m machines $M_1, M_2, \dots M_m$. Un ensemble O d'opérations doit être ordonnancé. Chaque job J_i est composé d'un ensemble de k_i opérations, notées $O_{i,1}, O_{i,2}, \dots O_{i,k_i}$. Une opération ne peut appartenir qu'à un seul job. Chaque opération $O_{i,j}$ est affectée à une machine $M_{i,j}$. Le temps de traitement d'une opération $O_{i,j}$ est noté $P_{i,j}$. Les opérations constituent la gamme opératoire d'un job, ainsi elles doivent être exécutées dans un ordre prédéterminé, et aucune opération ne peut être planifiée avant que l'opération précédente ne soit terminée. On note A l'ensemble des couples $(O_{i,j}, O_{i,j+1})$ des opérations qui doivent être exécutées successivement. Une opération $O_{i,j}$ ne peut être exécutée que sur une seule machine à la fois ; et une machine ne peut traiter qu'une seule opération à la fois. Aucune préemption n'est autorisée. On note E_v l'ensemble des opérations devant être affectées à la machine v. L'ordre de traitement des opérations de E_v doit être défini ; on dit que ces opérations sont en disjonction car les intervalles de temps pendant lesquels la machine est occupée sont disjoints.

Un ordonnancement μ est défini par les dates de début $s_{i,j}$ de chaque opération $O_{i,j}$. Pour le Job-shop, le nombre d'opérations de chaque job est constant. On a plus exactement $k_i = |M| = m$. Une instance d'un problème est donc une instance rectangulaire, et n est la taille d'un problème avec $n = r.m$. Dans la suite, et sans perte de généralité, il est noté $\{O_i\}_{i=[1,n]}$; les autres notations sont adaptées également. De plus, les hypothèses suivantes sont également retenues pour le problème de Job-shop :

- Les jobs sont disponibles à la date 0 ;
- Aucune indisponibilité des machines n'est considérée sur l'horizon de la planification (pas de pannes, pas de maintenance, …) ;
- Les temps de traitement sont déterministes ;

- Il n'y a pas de durées de préparation des machines (pas de montage/démontage, …) ;
- Les temps de transports entre les machines ne sont pas considérés ;
- Chaque machine ne peut réaliser à un instant donné qu'une seule opération ;
- Les zones de stockages intermédiaires sont considérées infinies ;
- Aucune ressource additionnelle n'est comprise ;

Certaines variantes du problème de Job-shop consistent à supprimer certaines de ces simplifications, ou à rajouter de nouvelles contraintes telles que des contraintes énergétiques par exemple, afin de se rapprocher au plus près des systèmes réels de production.

2.1. Formalisation mathématique

Afin de décrire le problème plus précisément, la formalisation mathématique proposée par (Manne, 1960) est ici adoptée ; elle repose sur les notations introduites dans la section précédente.

(i) $s_i - s_j \leq -p_i \; \forall (i,j) \in A$;

(ii) $s_i - s_j \leq -p_i \; oux \; s_j - s_i \leq -p_j, \forall (i,j) \in E_v, \forall v \in M$;

(iii) $s_i \geq 0, \forall i \in O$;

Dans cette formalisation, le *oux* correspond au *oux* exclusif, de ce fait ou bien l'équation de gauche est vérifiée ou bien l'équation de droite est vérifiée mais pas les deux simultanément.

L'objectif est de minimiser la date correspondant à la fin de toutes les opérations (i.e. le makespan), c'est-à-dire de minimiser le maximum des $s_i + p_i$:

(iv) $min(max_i(s_i + p_i))$.

2.2. Programme linéaire

La formalisation mathématique présentée ci-dessus peut être linéarisée en introduisant les variables binaires x_{ij}. La variable x_{ij} est définie par x_{ij}=1 si l'opération i est réalisée avant l'opération j sur la machine qu'elles ont en commun, x_{ijv}=0 sinon. La contrainte (ii) est alors remplacée par les contraintes suivantes :

(ii') $x_{i,j} \in \{0,1\}, \forall (i,j) \in E_v, v \in M$;

(ii") $x_{i,j} + x_{j,i} = 1, \forall (i,j) \in E_v, v \in M$;

(ii"') $s_j - s_i - p_i + H(1 - x_{i,j}) \geq 0, \forall (i,j) \in E_v, v \in M$.

Où H est une constante suffisamment grande : $H > \sum_{i \in O} P_i$.

L'ensemble des contraintes (i), (ii'), (ii"), et (ii"') représentent les contraintes du problème de Job-shop. Associées à la fonction objectif (iv), ces contraintes forment un programme linéaire en nombres entiers modélisant le problème de Job-shop.

La formalisation linéaire ci-dessus permet de résoudre tous les problèmes de Job-shop. Cependant, les temps de calcul nécessaires peuvent rapidement devenir prohibitifs. De ce fait, il est délicat d'aborder la résolution d'un problème de taille importante (dépassant 100 opérations) à l'aide d'un simple programme linéaire. Ainsi, la plupart des articles traitant du problème de Job-

shop reposent presque systématiquement sur des méthodes heuristiques et/ou métaheuristiques afin de proposer des solutions (Jacek Blazewicz et al., 1996 ; Jain and Meeran, 1999).

Les méthodes de résolution présentes dans la littérature proposent des ordonnancements de différentes natures et dont les représentations peuvent varier d'un article à un autre. Dans le paragraphe suivant certaines classes d'ordonnancement et différentes représentations des solutions sont présentées.

3. Classes d'ordonnancement et représentations

La suite de cette section est illustrée par un exemple de problème de Job-shop, composé de deux jobs. Chaque job comporte trois opérations. Chaque opération est à réaliser sur une machine donnée. Il y a trois machines dans le problème considéré. Les temps de traitement varient d'une opération à une autre. Les couples (machine ; durée) sont donnés dans le Tableau 1 ci-dessous. L'objectif du problème est de trouver un ordonnancement minimisant la durée totale de traitement de toutes les opérations (i.e. makespan).

Tableau 1 Instance exemple pour le Job-shop (P)

Job 1	$O_1 = (M_3 ; 4)$	$O_2=(M_1 ; 3)$	$O_3 = (M_2 ; 2)$
Job 2	$O_4 = (M_2 ; 7)$	$O_5 = (M_1 ; 6)$	$O_6 = (M_3 ; 5)$

Il est possible d'extraire à partir des données du problème deux informations : une borne inférieure et une borne supérieure. La première est égale à la durée permettant de traiter le job dont la somme des durées des opérations est la plus longue. Dans l'exemple présenté ici, une borne inférieure du problème vaut donc 18. Cette valeur correspond à la somme des durées de traitement du Job 2. Ainsi, une solution qui vaudrait 18 serait optimale.

De manière générale, il existe plusieurs ordonnancements répondant à un critère de performance. Sans forcément considérer des ordonnancements dont l'ordre des opérations diffère véritablement de l'un à l'autre, le simple fait de décaler une opération dans le temps revient à considérer un ordonnancement différent de l'ordonnancement original. Il est cependant possible de classer les ordonnancements qui respectent un critère de performance donné. (Baker, 1974) présente en détail ces classes et les définitions associées qui ne sont pas exposées ici. Cependant, deux classes d'ordonnancements parmi les plus répandues sont présentées : les ordonnancements semi-actifs et actifs.

Quelques outils de codage des solutions partielles (i.e. les solutions qui sont encodées - chapitre I, §4.3.1) d'un problème de Job-shop sont également décrits. L'objectif est de présenter le socle des outils utilisés dans l'ensemble des problèmes de cette thèse.

3.1. Classes d'ordonnancement

Comme énoncé dans le premier chapitre, les problèmes d'ordonnancement consistent généralement à minimiser (ou maximiser) un ou plusieurs critères de performances. Dans l'instance prise pour exemple, un ordonnancement de makespan minimal est recherché. Ces ordonnancements ne présentent pas tous les mêmes caractéristiques et deux classes

d'ordonnancements qui représentent des ordonnancements différents sont présentées, dont la première concerne les ordonnancements semi-actifs.

3.1.1. Ordonnancements semi-actifs

Définition : ordonnancement semi-actif

Un ordonnancement est dit-semi-actif s'il n'est pas possible de commencer une opération plus tôt sans modifier l'ordre dans lequel les opérations sont ordonnancées sur une machine (ou une ressource) et sans en détériorer le critère de performance.

Si un ordonnancement du problème énoncé dans le Tableau 1 est considéré, il est possible d'obtenir une solution telle que représentée dans la Figure 1. Dans cette figure, la partie A représente un ordonnancement optimal car il n'est pas possible de trouver un ordonnancement présentant un makespan inférieur en ce qui concerne le critère du makespan. Un ordonnancement consistant à décaler la date de début de deux unités vers la droite pour la première opération du Job 1 serait aussi optimal. Mais l'ordonnancement présenté dans la Figure 1(A) n'est pas semi-actif : il est possible d'avancer la première opération du Job 1, de telle sorte qu'elle débute à la date de début de toutes les opérations. L'ordonnancement B présente cette configuration et est donc semi-actif. En d'autres termes, pour un ordre de passage donné sur les machines (ressources) un ordonnancement semi-actif est un ordonnancement pour lequel la somme des dates de début des opérations est minimale.

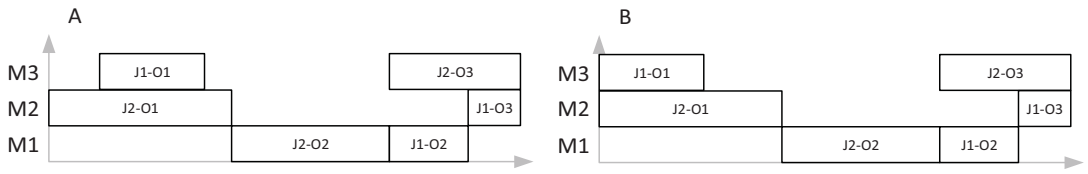

Figure 1 Ordonnancement optimal, non semi-actif (A) et l'ordonnancement semi-actif associé (B)

Il apparaît sur la Figure 1(B) que la deuxième opération du Job 1 peut être encore avancée. Il s'agit d'une autre classe d'ordonnancement : les ordonnancements actifs.

3.1.2. Ordonnancement actif

Il a été vu qu'à partir d'un ordre de traitement des opérations sur les machines, un ordonnancement semi-actif est obtenu en décalant à gauche les opérations, sans perturber l'ordre établi. Il est possible d'étendre cette notion en modifiant le critère d'arrêt qui consiste à stopper le décalage une fois qu'une opération débute directement après la fin de l'opération planifiée précédemment sur la machine. Si, en procédant par décalages successifs il n'est plus possible d'avancer la date d'une opération sans conduire à une violation des contraintes ou à une détérioration du critère de performance, alors l'ordonnancement considéré est dit actif.

Définition : ordonnancement actif

Un ordonnancement est dit actif s'il n'est pas possible d'avancer l'exécution d'une opération sans violer une contrainte ou sans retarder la date de début d'une autre opération.

A partir de cette définition, il est possible de concevoir une procédure visant à modifier la structure d'une solution afin d'obtenir une autre dont la fonction objectif est au moins aussi bonne que la fonction objectif de la solution précédente. La définition ci-dessus permet de concevoir une procédure pour construire les ordonnancements actifs. Or si une opération, après un décalage, est ordonnancée avant celle qui la précédait, l'opération de décalage est assimilable à une permutation. Il faut noter qu'en fonction de l'ordre dans lequel ces permutations sont effectuées il est possible d'obtenir des solutions très différentes. Cependant, la démarche de recherche d'un ordonnancement actif constitue une base intéressante à la définition d'une recherche locale efficace.

Dans le cadre de l'exemple présenté sur la Figure 1, il apparaît que la seconde opération du Job 1 peut être avancée avant la seconde opération du Job 2 sans détériorer la solution courante. La Figure 2 montre que les ordonnancements A et B diffèrent d'une permutation.

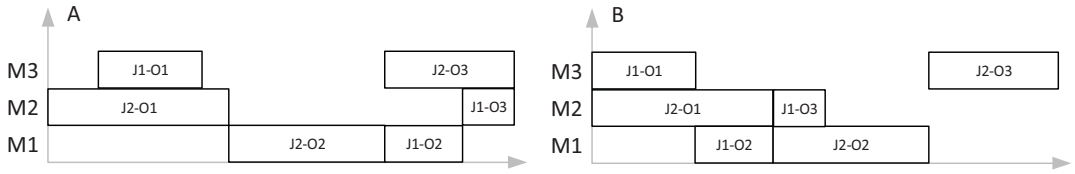

Figure 2 (A) est un ordonnancement optimal et non actif et (B) est l'ordonnancement actif associé

Dans cette thèse, la recherche d'ordonnancements semi-actifs est privilégiée. En effet, un ordonnancement actif est semi-actif, l'inverse n'étant pas vrai. De ce fait, il existe plus d'ordonnancements semi-actifs que d'ordonnancements actifs ce qui, dans le cadre d'une méthode d'optimisation, peut présenter un avantage, notamment dans le cadre de l'exploration de l'espace des solutions.

Les ordonnancements présentés ci-dessus sont exposés sous forme de diagrammes de Gantt. Cependant, il est rare de manipuler un tel objet, et il est préférable d'utiliser un codage de la solution, qui ne contient pas l'ensemble des informations. Ces codages, appelés solutions partielles, et évoqués dans le premier chapitre (§4.3.1), reposent sur des représentations particulières des solutions. Différentes représentations sont utilisées dans la littérature. Deux d'entre elles concernant l'ordonnancement sont présentées dans la suite, dont l'une est utilisée dans cette thèse.

3.2. Représentations pour l'ordonnancement

Les méthodes de résolution mises en œuvre dans la littérature reposent sur la notion de codage des solutions, c'est-à-dire la manière dont les solutions sont représentées au sein d'un modèle d'optimisation. De nombreuses représentations ont été définies dans la littérature, et deux d'entre elles sont présentées dans cette section étant donné qu'elles sont au cœur des méthodes d'optimisation proposées dans cette thèse.

3.2.1. Représentation par ordre total d'opérations

La représentation par ordre total consiste à créer un vecteur, représentant une solution partielle, et dont chaque cellule contient le numéro d'une opération. Un tel vecteur se lit/décode de gauche à droite, chaque opération étant ordonnancée en prenant en compte les opérations déjà visitées. A chaque nouvelle opération visitée, on affecte une date de début en fonction des

opérations précédentes dans la gamme, ou en fonction des opérations déjà planifiées sur la machine/ressource. Si un ordonnancement valide appartient à la classe des ordonnancements semi-actifs, la solution partielle ainsi codée peut représenter une séquence d'opération irréalisable. Dans le Tableau 2, trois ordonnancements stockés sous forme d'ordre total sont présentés. Les numéros des opérations font référence à l'exemple P (Tableau 1). Les premiers (A & B) permettent d'obtenir le diagramme de Gantt (B) de la Figure 1. Le troisième (C) n'est en revanche pas réalisable car l'opération 5 apparaît avant l'opération 4 dans le vecteur. Un vecteur présentant un ordre non strict des opérations d'un job est donc irréalisable. Il apparaît que la représentation par ordre total d'opérations permet de stocker bien plus de solutions partielles que de solutions du problème d'ordonnancement ; on parle de surreprésentation. De plus, il est possible de représenter de plusieurs manières la même solution. On parle alors de multireprésentation. C'est le cas des vecteurs A et B dans le Tableau 2. Le nombre de vecteurs qu'il est possible d'obtenir est égal à $n!$, où *n* représente le nombre total d'opérations. S'il n'est connu aucune référence bibliographique utilisant cette représentation, elle constitue un intermédiaire utile à la conception de méthodes ou de représentations plus efficaces (Caumond, 2006).

Tableau 2 Trois ordres totaux du problème P

A	1	4	5	2	3	6
B	4	5	6	1	2	3
C	1	5	2	3	4	6

3.2.2. Représentation par répétition

Si la représentation par ordre total présente des inconvénients en raison de la présence d'ordres non valides, il est possible de transformer un vecteur par ordre total en une représentation plus adaptée. En remplaçant un numéro d'opération par le numéro du job associé, la présence d'ordres non valides est supprimée. En effet, le vecteur ainsi constitué, lui aussi lu de la gauche vers la droite, permet de faire abstraction du numéro de l'opération considérée à chaque étape du décodage. Le numéro de l'opération n'est alors connu que lorsque l'opération est traitée par l'algorithme de décodage, où le numéro de job est converti en l'opération correspondant au nombre d'occurrences où le job a été déjà vu. De ce fait, le décodage d'un vecteur par répétition revient à décoder un vecteur par ordre total valide. Dans le Tableau 3 ci-dessous, un vecteur par répétition représentant la solution de la Figure 1 est donné. La partie A représente le vecteur A dans la représentation par ordre total (Tableau 2), le vecteur B est lui obtenu à partir du vecteur non valide (C) du même tableau. En repassant à un ordre total, le vecteur obtenu est présenté en partie C, qui représenterait la solution active de la Figure 2(B). Une telle représentation permet de stocker moins de solutions partielles que la représentation par ordre total, du fait de la redondance d'informations liées à la présence de valeurs identiques dans le vecteur. La représentation par répétition permet de stocker uniquement des solutions du problème d'ordonnancement, il n'y a donc pas de surreprésentation ; cependant les solutions sont multireprésentées en général. Pour le Job-shop, le nombre d'ordonnancements ainsi représentés vaut : $(\sum_{v\in M}|E_v|)!/\prod_{i\in J}k_i!$, qui se simplifie en $n!/m!^r$ pour les instances rectangulaires.

Le vecteur par répétition a été rendu célèbre par les travaux de (Bierwirth, 1995), où un algorithme génétique repose sur une représentation des chromosomes à l'aide d'un vecteur de ce type. Par rapport aux autres algorithmes génétiques proposés à l'époque dans la littérature, faisant intervenir des algorithmes d'amélioration des solutions, les solutions obtenues par l'auteur sont comparables en terme de qualité.

Tableau 3 Représentation par répétition du problème P et conversion en ordre total

A	1	2	2	1	1	2
B	1	2	1	1	2	2
C	1	4	2	3	5	6

3.3. Conclusion

Il existe d'autres classes d'ordonnancement et de nombreuses représentations dans la littérature ; seulement celles utilisées dans cette thèse sont présentées dans cette section. Pour plus d'informations sur les représentations, le lecteur peut se référer à (Cheng et al., 1996) où les représentations par matrice, ou par clé aléatoire sont également présentées.

Il est essentiel de prendre en compte la classe d'ordonnancement à laquelle appartiennent les solutions manipulées par un outil d'optimisation, ainsi que les représentations des solutions puisque certaines peuvent induire des ordonnancements non valides (ordre total). Les ordonnancements semi-actifs sont par exemple très utilisés dans la littérature. D'autres classes d'ordonnancements existent, comme les ordonnancements sans délai qui sont très utilisés dans le cadre de la simulation à événements discrets. Ils sont donc très répandus dans le cadre d'heuristiques avec une politique de gestion telle que first-in-first-out (FIFO). Si ces ordonnancements ont pour principal objectif d'éviter qu'une machine/ressource soit laissée inactive, ils ne contiennent pas obligatoirement la solution optimale d'un problème pour une fonction objectif donnée, comme le montre le dernier chapitre (chapitre V).

De manière générale, il ressort que les représentations utilisées sont généralement non dominantes (le nombre de solutions codées est important) et non compactes (le nombre de solutions stockées est important). En réalité il est assez difficile de trouver une représentation qui ne soit ni surreprésentée, ni multireprésentée. Il faut donc choisir une représentation qui présente le moins d'inconvénients en fonction du problème étudié. Pour le Job-shop, il n'y a pas à ce jour d'élément de codage permettant de représenter les solutions sans surreprésentation ni multireprésentation. Dans cette étude, le choix de la multireprésentativité prime sur celui de la sur-représentativité, afin d'être sûr de ne générer que des solutions partielles dont l'évaluation retourne des ordonnancements valides. Ainsi, les ordonnancements seront tous représentés à l'aide d'un vecteur par répétition.

La résolution d'un problème d'ordonnancement ne repose pas seulement sur le choix de la classe d'ordonnancement ou de la représentation des solutions. Dans la suite, un état de l'art permettant de présenter les méthodes et les outils utilisés dans la littérature sur les problèmes d'ordonnancement tel que le Job-shop est présenté.

4. Etat de l'art des problèmes de Job-shop

La littérature sur les problèmes d'ordonnancement et de Job-shop est très riche, citer tous les articles n'est donc pas envisageable. Cependant, certains d'entre eux sont présentés dans cette partie. S'il existe des états de l'art sur ces problèmes tels que ceux proposés par (Jacek Blazewicz et al., 1996) et par la suite par (Jain and Meeran, 1999) sur le Job-shop, ce dernier article fait toujours référence aujourd'hui, puisqu'il n'y a pas à notre connaissance de revue récente sur ce problème.

(Jain and Meeran, 1999) décrivent précisément le problème du Job-shop et présentent les différentes méthodes d'optimisation utilisées dans la littérature. Les méthodes sont rapidement présentées et les instances des problèmes, toujours utilisées dans les articles actuels, sont données. Des tableaux énoncent les résultats obtenus et présentent les articles ayant proposé les meilleurs résultats en fonction des instances, ainsi que les meilleures méthodes proposées alors. Les auteurs ont montré que, même si les premiers articles étaient principalement orientés vers les méthodes exactes et la complexité du problème, la tendance allait aux heuristiques et majoritairement vers les métaheuristiques à la fin du XXème siècle. S'il n'est pas possible d'extrapoler les informations recensées dans cet état de l'art aux méthodes actuelles du fait de la date à laquelle il a été effectué, il reste une référence couramment citée dans les articles. Un état de l'art récent serait cependant utile pour avoir un état des lieux plus précis.

Dans ce chapitre les outils sur lesquels reposent les algorithmes de résolution du problème de Job-shop sont présentés. Comme ce problème est NP-difficile au sens fort (Garey et al., 1976) la résolution d'instances de taille élevée n'est généralement pas possible à l'aide de méthodes exactes. Pour ce problème, de nombreuses métaheuristiques ont été proposées, parmi lesquelles la recherche tabou (Nowicki and Smutnicki, 2005 ; Zhang et al., 2007), le GRASP (Binato et al., 2000), les algorithmes évolutionnaires (Gonçalves et al., 2005 ; Hasan et al., 2009 ; Tsai et al., 2008) et bien d'autres. Cette partie met en évidence l'importance de différents points-clés parmi lesquels :

- Une représentation sous forme de graphe conjonctif-disjonctif ;
- Une représentation indirecte des solutions telle que la représentation par répétition ;
- Des mécanismes de recherches locales reposant sur le parcours des chemins critiques dans le graphe, et mettant en œuvre des mouvements telles que l'insertion ou la permutation d'opérations ;
- Un schéma générique permettant une exploration efficace de l'espace de recherche ;

L'objectif de cette section est donc de mettre en évidence les points-clés qui permettent la résolution des problèmes de Job-shop.

4.1. Outils pour la résolution des problèmes de JSP

4.1.1. Graphe conjonctif-disjonctif

Le modèle du graphe conjonctif-disjonctif a été initialement introduit par (Roy and Sussmann, 1964). Il permet de représenter les problèmes disjonctifs, c'est-à-dire ceux où les ressources permettant de traiter les opérations sont mutualisées, et où une ressource ne peut traiter qu'une seule opération à la fois. Dans ce type de problème, deux opérations sont en disjonction lorsqu'elles partagent la même ressource et qu'elles ne peuvent pas être exécutées simultanément.

Les intervalles d'exécution des opérations disjonctives ne peuvent se chevaucher : $]s_i, s_i + p_i[\cap]s_j, s_j + p_j[= \emptyset$, ce qui est représenté dans la formalisation mathématique par la contrainte $s_j - s_i \geq p_i \; oux \; s_i - s_j \geq p_i$. Si deux opérations en disjonction sont considérées, il faut arbitrer l'ordre de passage de chacune des opérations sur la ressource. On parle alors de disjonction arbitrée et dans le cas contraire de disjonction non arbitrée.

Le graphe modélisant le problème est alors appelé graphe disjonctif. Si l'on considère l'exemple du Tableau 1, le graphe disjonctif associé est obtenu en associant à chaque opération un sommet du graphe. Les termes opération et sommet seront utilisés indifféremment. Deux sommets fictifs sont ajoutés : le sommet 0 représentant une opération réalisée avant toutes les autres, et le sommet * représentant une opération effectuée après toutes les autres. Communément, 0 est appelé la source, et * est appelé le puits. Un arc est ensuite ajouté entre chaque opération successive d'un job ; cet arc appartient donc à l'ensemble des arcs qui constituent les données du problème et qui est noté A dans le chapitre I (§4.2.2). Ensuite, une arête est ajoutée entre chaque opération affectée à la même ressource. Le graphe ainsi obtenu à partir de l'exemple (P - Tableau 1) fourni au début de ce chapitre est présenté sur la Figure 3.

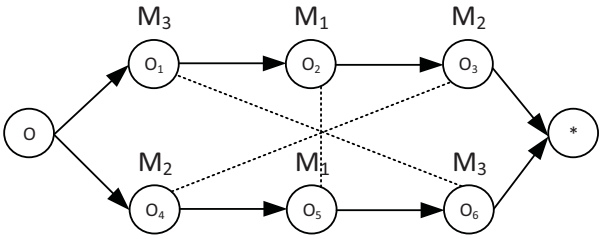

Figure 3 Graphe disjonctif d'un problème de Job-shop

Dans la Figure 3, les flèches représentent les précédences entre les opérations d'un même job. Les arêtes en pointillés représentent les disjonctions (les opérations O_2 et O_5 sont traitées sur la même machine M_1. On appellera *sélection* le fait d'arbitrer toutes les disjonctions d'un graphe qui modélise un problème. Chaque arête est alors transformée en arc représentant une disjonction arbitrée. On parle alors de graphe disjonctif orienté. Aussi, l'arête alors orientée prend le poids correspondant à la durée de l'opération de départ. Un graphe disjonctif orienté associé au graphe de la Figure 3 est présenté dans la Figure 4.

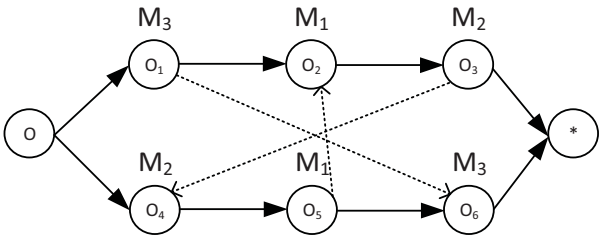

Figure 4 Graphe présentant un cycle pour un problème de Job-shop

Dans la Figure 4, la sélection effectuée a mené aux orientations $O_1 \rightarrow O_6$, $O_5 \rightarrow O_2$ et $O_3 \rightarrow O_4$. Cette sélection introduit cependant un cycle absorbant dans le graphe $O_3 \rightarrow O_4 \rightarrow O_5 \rightarrow O_2 \rightarrow O_3$. Un cycle absorbant est un cycle de longueur positive dans le graphe. Or, pour le problème de Job-shop classique, les arcs présents dans le graphe sont de poids positifs, et donc les cycles potentiels sont de longueur strictement positive. Ainsi, la sélection présentée dans la Figure 4 est appelée *sélection non-valide*. Les solutions recherchées pour le problème de Job-shop sont donc représentées

par un graphe disjonctif orienté acyclique. Par exemple, dans la Figure 5, aucun cycle n'a été introduit dans le graphe une fois l'arbitrage des deux disjonctions effectué. On parle alors de *sélection valide*. Une sélection valide conduit à un graphe que l'on appelle graphe *conjonctif*. Afin d'obtenir des graphes conjonctifs, il est intéressant de choisir des représentations n'envisageant que des sélections valides, c'est-à-dire sans surreprésentation. Ainsi, si la représentation par ordre total permet d'orienter les arcs, elle n'est pas adaptée au problème de Job-shop, et il est plus intéressant d'utiliser une représentation par répétition.

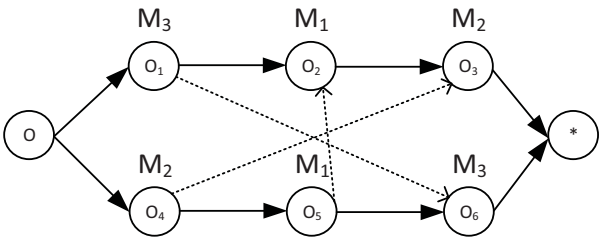

Figure 5 Graphe conjonctif pour le problème de Job-shop

Une fois le graphe conjonctif connu, il est nécessaire de définir les dates de début de chaque opération. Pour ce faire, un calcul de plus long chemin est requis. La procédure appliquée dans le cadre du Job-shop est décrite dans la partie concernant l'évaluation du graphe ci-dessous.

4.1.2. Évaluation du graphe

Une fois que l'on dispose d'un graphe conjonctif, il convient de définir les dates de début de chaque opération conformément à la sélection opérée. Dans le cadre des problèmes de Job-shop, l'algorithme permettant de fixer les dates de début consiste en un calcul de plus long chemin. Cet algorithme définit plus exactement les dates de début au plus tôt des opérations, ce qui permet d'obtenir à la fin un ordonnancement semi-actif associé au graphe conjonctif. Dans la Figure 6, la démarche générale pour obtenir un ordonnancement semi-actif à partir d'un problème est présentée. La première étape consiste à modéliser le problème sous forme de graphe disjonctif. Une fois une sélection valide définie, le graphe disjonctif devient un graphe conjonctif. Il convient alors d'évaluer ce graphe avec un algorithme de plus long chemin.

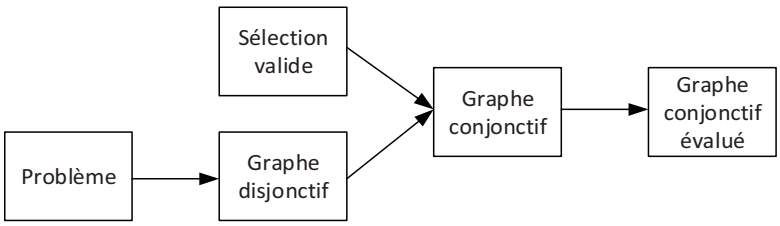

Figure 6 Démarche d'obtention d'un ordonnancement semi-actif pour le Job-shop

Il existe plusieurs algorithmes permettant de calculer des chemins dans un graphe. Ces algorithmes sont généralement dédiés à la recherche de plus courts chemins, tels que l'algorithme de Dijkstra ou celui de Bellman-Ford. Ils sont cependant facilement modifiables pour retourner le (ou les) chemin le plus long. La difficulté principale est de choisir un algorithme dont la complexité est la plus faible possible. En effet, dans une méthode d'optimisation telle qu'une métaheuristique qui manipule des graphes, il est probable que la fonction d'évaluation soit appelée à plusieurs reprises, notamment lors de l'exploration des voisinages. Or, la complexité temporelle d'un

algorithme d'évaluation se répercute directement sur les temps de calcul de la méthode globale. Il est donc important de sélectionner un algorithme de calcul de plus long chemin le plus efficace possible et pouvant s'affranchir des caractéristiques du graphe. Par exemple, l'algorithme de Bellman-Ford peut être appliqué sur un graphe contenant des cycles absorbant alors que ce n'est pas le cas pour l'algorithme de Dijkstra. Cette particularité implique que l'algorithme de Bellman-Ford a une complexité plus importante. Dans le cas général, la complexité de cet algorithme vaut $O(|N||A|)$ où N représente l'ensemble des nœuds et A l'ensemble des arêtes. En contrepartie la complexité de l'algorithme de Dijkstra vaut $O((|N| + |A|) \log |N|)$. Dans les deux cas, ces algorithmes ont une complexité polynomiale.

Dans le cadre d'un graphe tel que présenté dans la Figure 5, une évaluation par un algorithme de plus long chemin retournerait le graphe évalué ci-dessous :

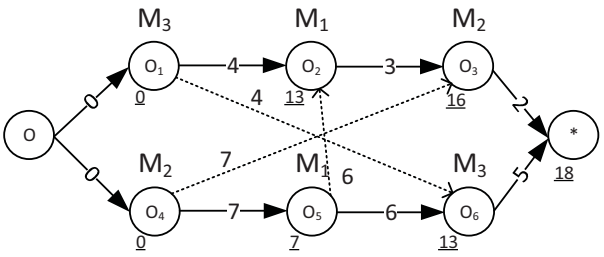

Figure 7 Graphe conjonctif évalué

Dans la Figure 7, chaque opération se voit assigner une date de début au plus tôt. En exécutant l'algorithme sur l'ensemble des opérations, la date de début de l'opération * vaut 18. Cette valeur correspond donc à la durée de traitement de l'ensemble des opérations, également appelé makespan. L'algorithme de plus long chemin de Bellman-Ford peut être utilisé dans de nombreuses situations, et également dans le cas des graphes avec cycles. Cependant, dans le cadre des problèmes de Job-shop, il est possible de définir un algorithme de plus long chemin qui exploite au mieux la représentation choisie pour modéliser les précédences dans le graphe conjonctif. Cet algorithme repose sur la manipulation du sous-ensemble des solutions partielles qui constituent les vecteurs par ordre total des opérations. Plus précisément, ce sous-ensemble contient les vecteurs dont les éléments sont triés topologiquement. On parle alors d'ordre topologique. Avec un tel ordre, une lecture du vecteur de gauche à droite respecte strictement les précédences liées aux arcs conjonctifs dans le graphe modélisant le problème. En d'autres termes, l'opération O_i apparaîtra avant l'opération O_j si un arc du graphe va de O_i vers O_j. Cette caractéristique permet de ne représenter que les graphes acycliques. Il est cependant nécessaire de connaître un tel ordre pour profiter de ses avantages, et tous les ordres totaux ne représentent pas des ordres topologiques. Le passage d'un ordre total à un ordre topologique peut requérir une procédure de réparation. Pour se prémunir d'une telle situation, il est donc préférable de ne manipuler que des vecteurs représentant des ordres topologiques. Or, l'ordre total des opérations obtenu à partir de n'importe quel vecteur par répétition correspond à un graphe acyclique, et par conséquent, ce vecteur est un ordre topologique des opérations. Il est donc possible de parcourir ce vecteur de gauche à droite sans jamais revenir sur une opération déjà visitée. Ainsi, on peut simplifier l'algorithme en supprimant les instructions relatives à la détection des cycles. La complexité de l'algorithme ainsi constitué devient donc linéaire en fonction du nombre d'opérations et vaut $O(n)$, où n est le nombre d'opérations. L'Algorithme 1 présente le principe de l'évaluation du plus long chemin pour le Job-shop.

Algorithme 1 : Evaluer_Job-shop

Entrée

$data$	Données du problème

Entrée/Sortie

S	Solution à évaluer

Variables

job	Job traité à chaque itération
m	Machine devant traiter l'opération du job
op	Opération courante
$dateJ, dateM$	Date de début au plus tôt en fonction de l'opération précédente du job ou de la machine
$prec\_mac, occu\_job$	Tableaux pour stocker la dernière opération sur chaque machine et les occurrences des jobs

Début

1. Initialiser les tableaux $prec\_mac$ et $occu\_job$;
2. **POUR** $i \coloneqq 1$ **A** $data.taille$ **FAIRE**
3. $job \coloneqq S.vecteur[i]$; // on récupère le *job* à la position *i* dans le vecteur par répétition
4. $op \coloneqq$ opération de job à traiter ;
5. $m \coloneqq$ machine devant traiter l'opération courante du job ;
6. **SI** une opération de job a déjà été traitée **ALORS**
7. $dateJ \coloneqq$ date de fin de la dernière opération traitée pour le job ;
8. **FIN SI**
9. **SI** une opération a déjà été traitée par m **ALORS**
10. $dateM \coloneqq$ date de fin de la dernière opération traitée par m ;
11. **FIN SI**
12. $S.date[op] \coloneqq max(dateJ\ ;\ dateM)$; // la date de début de l'opération est mise à jour
13. Stocker dans S toute autre information pertinente ;
14. Mettre à jour les tableaux $prec\_mac$ et $occu\_job$;
15. **FIN POUR**
16. Retourner S ;

Fin

Dans l'Algorithme 1 de plus long chemin décrit ci-dessus, la première instruction consiste à initialiser les tableaux temporaires. Ensuite, la boucle principale (POUR) correspond au calcul des plus longs chemins. Dans cette boucle, les jobs stockés dans le vecteur par répétition sont parcourus un à un. En fonction du nombre de fois que le job a déjà été vu, l'opération correspondante associée à l'occurrence du job est récupérée (calcul en O(1)), ainsi que la machine qui lui est associée. Ensuite (lignes 6-8), la date de début au plus tôt en fonction de l'opération du job précédemment ordonnancée est obtenue. Puis (lignes 9-11) la date au plus tôt en fonction de la machine. Si aucune opération du job n'a été vue, ou que la machine n'a traité aucune opération, ces valeurs valent donc 0 (ou la date de début du job si des 'release dates' sont considérées). La plus grande des deux valeurs est ensuite choisie comme étant la date de début de l'opération (ligne 12), et cette date est sauvegardée dans la solution. Comme l'algorithme manipule un ordre topologique, les opérations sont visitées de telle sorte que chaque mise à jour est définitive et donc, aucune itération de la boucle POUR ne remettra en cause les itérations passées. Il est ensuite possible de sauvegarder dans la structure de données stockant la solution d'autres valeurs pouvant être utiles par la suite, telle que l'opération précédant directement l'opération courante par exemple, aussi appelée *père*.

La connaissance de l'opération finissant au moment où l'opération courante commence permet de construire le chemin critique dans le graphe, c'est-à-dire le chemin où chaque opération est ordonnancée sans délai. Ainsi, toute opération subissant un décalage dans ce chemin impactera

négativement le critère de performance qu'est le makespan. Toute opération a au moins un prédécesseur : l'opération *O* modélisant la source du graphe. Aussi, dans le cas où deux opérations peuvent être choisies comme précédant directement l'opération courante, un arbitrage est effectué. Cet arbitrage peut être fait de manière déterministe (l'opération précédente sera soit toujours l'opération précédente dans la gamme du job, ou l'opération précédente sur la machine) ou de manière stochastique.

4.1.3. Chemins critiques

Lors de la procédure d'évaluation, il est possible de stocker un certain nombre d'informations en plus de la date de début des opérations (ligne 13 de l'Algorithme 1). L'opération dont la date de fin conditionne la date de début d'une autre opération fait partie de ces données pertinentes à sauvegarder. Comme mentionné précédemment, il peut exister plusieurs opérations ayant cette propriété, mais une seule est conservée. Cette opération particulière est appelée $père$ (en lien avec l'opération qu'elle précède, appelée $fils$). En stockant cette information unique dans un tableau $Père$, il est alors possible de connaître instantanément le père d'une opération op, par simple accès à la cellule du tableau $Père[op]$. Il est donc possible de connaître le père d'une opération en $O(1)$. Si l'on se réfère à un graphe conjonctif-disjonctif, la dernière opération du chemin critique est donc le père de l'opération $*$. Le père de cette opération est stocké dans la cellule d'indice $n+1$ dans le tableau $Père$, où n correspond au nombre d'opérations à ordonnancer sur les machines. Le chemin critique est donc parcouru depuis le sommet $*$, jusqu'au sommet d'origine, O. Ce sommet n'est pas explicitement stocké dans le tableau, mais chaque opération ayant son père initialisé à -1 au début de l'algorithme d'évaluation, le parcours du chemin critique s'arrête lorsque le père d'une opération vaut cette valeur.

Le chemin critique prend son nom des opérations qui le constituent. Ces opérations sont dites critiques, car tout retard dans la date de début d'une opération sur le chemin critique entraîne obligatoirement une modification à la hausse de la longueur du chemin. En d'autres termes, les opérations du chemin critique sont ordonnancées sans délai. De par sa nature, le chemin critique est le plus long chemin du graphe conjonctif-disjonctif. C'est notamment à partir de ce chemin que le makespan est obtenu. L'objectif de l'optimisation est de réduire la valeur de ce makespan. Or, toute modification d'un ordonnancement qui n'altère pas le chemin critique ne peut conduire qu'à l'apparition d'un nouveau chemin critique de longueur supérieure au chemin précédent, car ce dernier existe toujours dans le graphe. La modification du chemin critique est donc primordiale dans la diminution du makespan d'une solution. Cependant, il n'est pas assuré qu'une modification du chemin critique conduise en tout temps à un chemin critique de longueur inférieure.

Dans la Figure 8, un chemin critique du graphe présenté en Figure 7 est indiqué (arcs épais et noirs). Dans cette figure on peut facilement constater la présence de deux chemins critiques, car la suite d'opérations $O \rightarrow O_4 \rightarrow O_5 \rightarrow O_6 \rightarrow *$ est également un chemin critique. Il apparaît donc que le chemin critique passe par des opérations consécutives d'un même job ; il est évident que des modifications ne sont pas possibles sur ces arcs car ils constituent la gamme opératoire du job. Ainsi, les seuls arcs modifiables d'un chemin critique sont les arcs disjonctifs (i.e. $O_5 \rightarrow O_2$ par exemple).

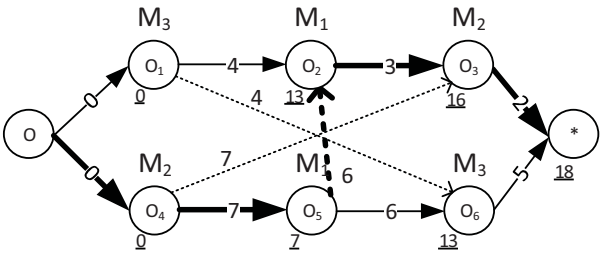

Figure 8 Exemple d'un chemin critique dans un graphe conjonctif

Comme mentionné précédemment, tout retard dans la date de début d'une opération sur le chemin critique entraînera un retard sur la date de début du sommet terminal *. Ceci vaut dans un contexte déterministe. Il est important de mentionner que toute variation de la durée de traitement d'une opération sur le chemin critique conduit également à une évolution du makespan (positive ou négative). Dans un tel contexte, dit stochastique, il est alors intéressant de chercher un ordonnancement le moins sensible possible aux fluctuations sur le chemin critique.

La plupart des systèmes de voisinage utilisés dans le cadre de recherches locales efficaces reposent sur l'utilisation du chemin critique. Certains de ces voisinages sont décrits dans les parties suivantes.

4.1.4. Systèmes de voisinages

4.1.4.1. Les blocs de Grabowski

(Grabowski et al., 1986) ont introduit la notion de bloc pour structurer le chemin critique. Un bloc constitue une suite d'opérations consécutives sur une même machine. Lorsque ces opérations appartiennent au chemin critique, on parle de bloc critique. Dans la suite le terme de bloc sera utilisé en lieu et place des blocs critiques car ce sont ces derniers qui sont particulièrement intéressants.

De manière plus spécifique, soit ρ un chemin critique dans le graphe G. On peut écrire $\rho = \{O_1^\rho, O_2^\rho, \dots, O_k^\rho\}$ où O_1^ρ est la première opération du chemin critique, ordonnancée à la date 0, et où O_k^ρ est la dernière opération du chemin critique, précédant directement l'opération $*$. Il est possible de partitionner ρ de telle sorte que chaque sous-suite d'opérations consécutives sur le chemin critique affectées à la même machine appartient à un ensemble fini noté B_i. Ainsi, on a $\rho = \{B_1, B_2, \dots, B_u\}$, où u est inférieur ou égal à k. Les blocs construits sont maximaux, on ne trouve donc pas deux blocs consécutifs dont les opérations sont traitées sur la même machine.

Dans la Figure 9 ci-dessous, un graphe est proposé. Ce graphe comporte plus d'opérations que l'exemple précédemment utilisé, afin de montrer plus facilement l'existence des blocs.

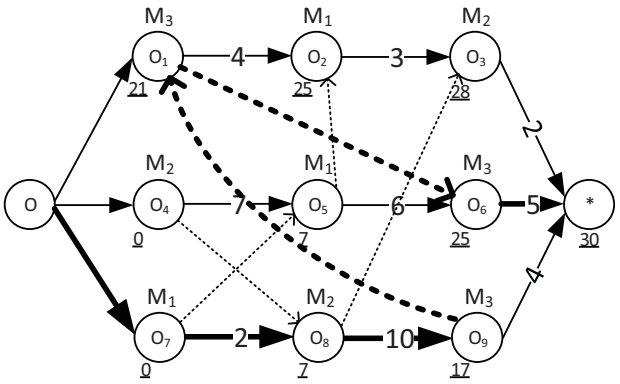

Figure 9 Présentation des blocs critiques dans un graphe conjonctif

Dans la Figure 9, le chemin critique est représenté à l'aide d'arcs noirs épais. Sur ce chemin, l'ensemble constitué des opérations $O_9 - O_1 - O_6$ est un bloc au sens de (Grabowski et al., 1986). Cette notion de bloc est récurrente dans la plupart des articles traitant de problèmes d'ordonnancement. Elle a été par exemple utilisée par (Nowicki and Smutnicki, 1996 ; Van Laarhoven et al., 1992).

4.1.4.2. Voisinages pour le Job-shop

Le voisinage de (Van Laarhoven et al., 1992) consiste à permuter deux opérations consécutives à l'intérieur d'un bloc. Dans le cas de la Figure 9, il est possible de permuter les opérations $O_1 - O_6$, ou $O_9 - O_1$. Sur un problème plus important, toutes les permutations sur les couples d'opérations consécutives appartenant à un même bloc sont effectuées. Les auteurs ont montré que le voisinage ainsi exploré est faiblement connecté, c'est-à-dire qu'il est possible de trouver la solution optimale en appliquant un nombre fini de permutations sur les opérations consécutives d'un bloc. Ce voisinage a été utilisé par les auteurs dans un recuit simulé.

Le système de voisinage ainsi construit peut être vide. Effectivement, si le chemin critique ne passe que par des opérations conjonctives, appartenant donc à un seul et même job, aucun bloc n'est construit. De ce fait un voisinage vide implique que la solution considérée est optimale. Cependant, trouver un chemin critique passant uniquement par les opérations d'un même job est relativement rare.

Le fait de structurer le chemin critique à l'aide des blocs permet de mieux détecter les opérations sur lesquelles opérer des modifications afin d'obtenir de meilleures solutions. Aussi, il apparaît au travers de cette notion qu'il est inutile de faire des modifications sur les arcs reliant des opérations à l'intérieur d'un bloc. Cette caractéristique est illustrée par la suite.

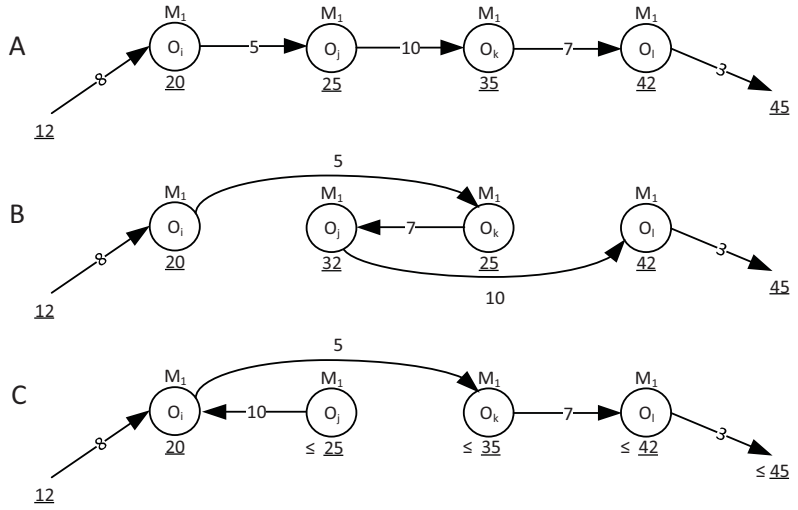

Figure 10 Exemple de permutation inutile au sein d'un bloc critique (B) et mouvement potentiellement améliorant (C)

Dans la Figure 10, un bloc formé de 4 opérations sur le chemin critique et traitées par la même machine (M_1) est considéré. Dans le chemin en haut de la figure, la date de début de l'opération précédant le bloc vaut 12, et la date de fin vaut 45. Or un échange des opérations O_j et O_k au sein du bloc produit la même valeur terminale, qui va conditionner la date de début de l'opération arrivant après le bloc considéré. Il n'est pas nécessaire, pour le problème de Job-shop, d'effectuer des permutations à l'intérieur des blocs. Les seules modifications qui peuvent améliorer l'ordonnancement concernent donc les opérations sur les bords des blocs. Ainsi, si l'arc disjonctif entre les opérations O_i et O_j est permuté le chemin critique n'est plus le même. En effet l'opération O_{i-1} et l'opération O_j ne peuvent pas être ensemble sur le chemin critique. La seule raison qui conduirait alors à une date de début inchangée pour l'opération O_{l+1} proviendrait des prédécesseurs conjonctifs des opérations O_j, O_k et O_l. Il apparaît donc que le voisinage exploré par (Van Laarhoven et al., 1992) contient des mouvements superflus. Ce voisinage a été amélioré par la suite par (Nowicki and Smutnicki, 1996). Cependant, le voisinage de (Van Laarhoven et al., 1992) peut être utilisé pour des extensions des problèmes considérés car il est rapide à mettre en œuvre et permet d'explorer finement le voisinage d'une solution dont on ne connaît pas de simplifications.

Le voisinage proposé par (Nowicki and Smutnicki, 1996) consiste en une amélioration du voisinage de (Van Laarhoven et al., 1992) pour le problème de Job-shop basée sur la constatation faite sur la Figure 10. Dans le voisinage proposé, seules les opérations en périphérie d'un bloc sont permutées. Cependant, les deux opérations à la fin du dernier bloc sur le chemin critique, ainsi que les deux opérations en tête du premier bloc sur le chemin critique, sont omises car ces modifications n'apportent aucune amélioration. Il a été montré par les auteurs que ce voisinage domine celui de (Van Laarhoven et al., 1992), car toute meilleure solution obtenue par ce dernier est atteignable par le voisinage de (Nowicki and Smutnicki, 1996) et il est également faiblement connecté. Du fait de la réduction en nombre des permutations effectuées, le voisinage de (Nowicki and Smutnicki, 1996) est également plus performant concernant les temps de calcul. D'autres voisinages développés pour des problèmes dont les objectifs sont différents existent dans la littérature, et peuvent être appliqués au problème de Job-shop. Pour plus d'information sur ces voisinages, le lecteur peut se référer à l'article de (Kuhpfahl and Bierwirth, 2016).

4.1.5. Conclusion sur les outils de résolution utilisés

Les voisinages ainsi que les différents outils proposés ci-dessus ont permis l'élaboration de méthodes efficaces au sein de recherches locales et de métaheuristiques. La plupart de ces méthodes reposent sur des propriétés du chemin critique, couramment utilisées dans les problèmes disjonctifs. Les outils mentionnés jusqu'à maintenant sont exploitables pour les problèmes de Job-shop. Dans la suite, certains articles qui mettent en œuvre ces outils dans le cadre de la résolution de Job-shop sont présentés.

4.2. Travaux récents sur le problème de JSP

Les méthodes de la littérature reposent principalement sur des outils particuliers que sont : le graphe conjonctif-disjonctif, le chemin critique, les systèmes de voisinage et les heuristiques/métaheuristiques. Ces outils permettent d'explorer efficacement l'espace des solutions en utilisant des caractéristiques connues sur les ordonnancements. Il est alors possible de développer des algorithmes efficaces pouvant obtenir des solutions de bonne qualité. Certaines méthodes reposant sur ces outils sont présentées ci-dessous.

Dans (Binato et al., 2000) un GRASP est utilisé. Les auteurs présentent la structure de l'algorithme qui consiste en la génération d'une solution, et en une recherche locale. L'apport principal de l'article concerne l'heuristique de construction randomisée mise en œuvre. Cette heuristique de construction randomisée est guidée par les solutions générées dans le passé. De plus, la méthode de construction est alliée à un principe d'optimalité approchée (POP – Proximate Optimality Principle) consistant à effectuer une recherche locale à certains stades d'avancement de l'heuristique afin d'effectuer une réparation partielle de la solution en cours de création. Ce principe permet d'éviter de mal placer certaines opérations dans la phase de construction à cause d'un mauvais choix lié à l'aspect aléatoire d'une telle procédure. Leurs résultats sont de bonne qualité, mais les temps de calcul s'avèrent relativement élevés.

(Zhang et al., 2008) proposent une approche combinant recuit simulé et recherche tabou pour résoudre le problème. Les auteurs utilisent un voisinage au sein de la recherche tabou basé sur celui proposé par (Jacek Blazewicz et al., 1996), consistant à échanger deux opérations d'un bloc critique dès lors que l'une d'entre elles est la première ou la dernière du bloc. Une procédure d'estimation de la qualité des solutions proposée par (Balas and Vazacopoulos, 1998) est également mise en œuvre. Les résultats obtenus sont de bonne qualité.

(Qing-dao-er-ji and Wang, 2012) proposent un algorithme génétique hybride. La procédure de croisement utilisée consiste dans un premier temps à diviser l'ensemble des machines en deux sous-ensembles complémentaires. L'étape suivante vise à construire deux descendants à partir de deux parents ; au début chaque descendant est une copie d'un parent. Ensuite, un des sous-ensembles de machines est sélectionné aléatoirement. Pour le premier fils (issu du parent A), la séquence de passage des opérations sur chaque machine du sous-ensemble sélectionné correspond à la séquence de passage de l'autre parent (parent B). Le deuxième descendant est construit en intervertissant les rôles des deux parents. La procédure de mutation consiste à effectuer des permutations sur le chemin critique d'une solution. Pour chaque couple d'opérations O_i et O_j consécutives sur le chemin critique et traitées sur la même machine, deux sous-ensembles sont construits : le premier est formé des deux opérations plus l'opération précédant O_i sur la machine. Le deuxième est formé de la même manière mais en ajoutant l'opération succédant O_j sur la machine. Ensuite, toutes les permutations conduisant O_j à être traitée avant O_i dans les deux sous-ensembles sont considérées. A l'issue de ces permutations, une des configurations plaçant O_j avant

O_i est choisie de manière aléatoire. Ce nouvel ordre de passage est utilisé dans la solution avec une probabilité donnée. La procédure est réitérée pour chaque couple d'opérations sur le chemin critique. Les auteurs proposent également un algorithme de recherche locale. Cet algorithme permet de construire un ensemble de solutions à partir d'une solution passée en paramètre. Chaque nouvelle solution de l'ensemble est obtenue à partir de la solution initiale en renversant l'ordre des arcs sur une machine. La meilleure solution ainsi obtenue est choisie pour devenir la nouvelle solution courante. La métaheuristique constituée de toutes ces procédures présente de bons résultats.

(Cheng et al., 2016) proposent un algorithme évolutionnaire associé à une recherche tabou. Un voisinage proposé par (Zhang et al., 2007) est utilisé ; il consiste à insérer, en tête ou en fin de bloc, une opération qui lui est intérieure, ou bien à déplacer la première ou la dernière opération à l'intérieur du bloc. Ce voisinage implique donc une exploration d'un espace important. Une estimation de la qualité des mouvements générés par le voisinage est mise en œuvre pour contrebalancer la taille de l'espace de recherche et accélérer la recherche tabou. La méthode de croisement consiste à générer des descendants en réinsérant dans leurs séquences la plus longue suite d'opérations communes entre les deux parents au même endroit. Les espaces vides sont comblés en introduisant les opérations provenant de l'autre parent en respectant leur ordre d'apparition. Aucune procédure de mutation n'est utilisée dans leur approche. Les résultats présentés sont d'excellente qualité.

(Kurdi, 2015) propose d'utiliser une métaheuristique de type « Island Model Genetic Algorithm » qui consiste à partitionner la population d'un algorithme génétique en sous-populations (les îlots – *islands*). Les îlots sont tous indépendants, ce qui permet de paralléliser facilement l'exploration de l'espace de recherche. Cependant, une communication existe entre les îlots, qui partagent un ensemble d'informations, et notamment certains individus qui vont migrer d'un îlot à un autre. Un algorithme génétique est appliqué sur chaque îlot, avec des procédures de croisement et mutation. La procédure de croisement est le croisement en deux points qui consiste à choisir deux points sur les gènes des parents. Chaque descendant reprend une partie des gènes de chaque parent entre les deux points, puis les places vacantes sont remplies à partir des informations de l'autre parent. Trois opérateurs de mutations, appliqués aléatoirement, sont utilisés : inversion (les éléments d'une séquence sont échangés terme à terme entre deux points) ; insertion (un élément est inséré avant un autre dans la séquence) ; swap (deux éléments sont simplement échangés). Une recherche locale basée sur une recherche tabou utilisant le voisinage de (Nowicki and Smutnicki, 1996) est proposée. Leurs résultats sont de très bonne qualité.

(Gao et al., 2015) proposent une méthode reposant sur une optimisation par essaim de particules (PSO – Particle Swarm Optimization), couplée à une recherche par voisinage variable (VNS). La PSO permet de gérer la population (l'essaim) et de faire évoluer ses membres en considérant la meilleure solution de l'essaim, le passé de la particule considérée, et sa trajectoire. L'algorithme intègre également des éléments originaires des algorithmes génétiques en utilisant un opérateur de croisement (POX – un ensemble de job est généré, chaque Job est réinséré à la même place dans le descendant que dans chaque parent, puis les places vacantes sont remplies en respectant l'ordre des jobs provenant de l'autre parent) et de mutation (insertion d'un élément avant un autre). Le VNS est utilisé en tant que recherche locale et utilise deux types de voisinages. Le premier correspond à celui proposé par (Nowicki and Smutnicki, 1996), et le second consiste à permuter un ensemble d'éléments de la séquence (5 éléments). Les auteurs ont obtenu de bons résultats avec cette approche.

(Mirshekarian and Šormaz, 2016) s'intéressent à un problème de JSP dans lequel ils cherchent des corrélations entre les caractéristiques des instances et l'estimation d'un critère de performance. Comme noté par les auteurs, le critère de performance à prendre en compte a une importance

cruciale. Si les caractéristiques considérées sont les temps de traitement des opérations et le makespan, il est probable que ce dernier sera très impacté par la taille des instances. Pour traiter ce problème les auteurs utilisent plutôt l'efficience de l'ordonnancement. Cette valeur calculée (appelée label dans leur article) correspond à la productivité dans l'usine dans le sens où une valeur de 1 signifie qu'il n'y a aucune attente. Ainsi, plus il y a des durées d'inactivité plus le critère est important. Les auteurs comparent ces caractéristiques avec des résultats obtenus à l'aide d'heuristiques de construction basées sur des règles de gestion, de type FIFO, LPT, etc… Ce travail est donc à la frontière des méthodes d'optimisation couramment mises en œuvre, mais présente des éléments intéressants pour obtenir rapidement des valeurs cibles du critère de performance.

Comme le montre cette vue d'ensemble des méthodes actuelles, la conception des voisinages et la réalisation d'heuristiques de construction sont au cœur de la réalisation de métaheuristiques efficaces. Pour démontrer l'efficacité de ces approches, il est proposé d'adapter une heuristique de construction ainsi qu'une méthode de recherche locale et de les utiliser au sein d'une métaheuristique de type GRASP×ELS. Les résultats obtenus sur des instances classiques de la littérature sont comparés aux travaux présentés dans cette section.

5. Proposition d'un GRASP×ELS pour le JSP

Récemment, le GRASP×ELS (Prins, 2009) a été utilisé pour fournir des solutions au problème de Job-shop (Chassaing et al., 2014). L'algorithme de principe de cette métaheuristique a été présenté dans le chapitre I (§5.2.4.5). Les algorithmes de principe correspondant aux différentes phases du GRASP×ELS, à savoir l'algorithme permettant de construire une solution initiale et celui de la recherche locale sont décrits. Les résultats obtenus à l'aide de la méthode développée sont présentés dans cette section.

5.1. Composants du GRASP×ELS

5.1.1. Phase de construction

La phase de construction du GRASP vise à créer une solution réalisable, en affectant et en ordonnançant une opération à chaque étape jusqu'à ce que toutes les opérations soient planifiées. Cet algorithme de construction est basé sur la proposition de (Binato et al., 2000) pour le JSP où l'objectif est de construire à chaque étape une liste de candidats restreints (RCL). À partir de l'ensemble contenant les jobs avec toutes les opérations antérieures déjà ordonnancées, un sous-ensemble de jobs est sélectionné en respectant un critère visant à limiter l'augmentation du makespan. Soit O_i une opération qui doit être planifiée. À une itération donnée de la phase de construction, au plus $|J|$ opérations peuvent être planifiées, où J est l'ensemble des jobs car chaque opération d'un job dépend des opérations antérieures déjà planifiées de ce job. Cet ensemble d'opérations est noté O_a, alors que les opérations déjà planifiées sont définies dans l'ensemble O_p. À chaque itération de la phase de construction, une liste CL est construite. Cette liste est le sous-ensemble des opérations qui peuvent être traitées à cette itération. Le plus petit makespan parmi toutes les opérations dans CL est noté $\underline{C_{max}}$; Le plus élevé est noté $\overline{C_{max}}$. La liste de candidats restreints (RCL) est alors définie de la manière suivante :

$$RCL = \left\{ O_i \in CL | \underline{C_{max}} \leq C_{max}(O_i) \leq \underline{C_{max}} + \beta(\underline{C_{max}} + \overline{C_{max}}) \right\},\ \beta \in [0,1]$$

L'opération suivante entrant dans le sous-ensemble O_p est ensuite choisie de manière aléatoire parmi les opérations de RCL. Comme l'ont souligné les travaux de (Binato et al., 2000), la sélection d'un élément dans RCL pourrait suivre n'importe quelle distribution de probabilité. Dans ce travail, une distribution uniforme est choisie afin de donner à chaque élément la même probabilité d'être sélectionné. L'Algorithme 2 fournit la procédure de l'heuristique de construction.

Algorithme 2 : Generer_Solution

Entrée

$data$	Structure stockant les données du problème
β	Nombre réel pour définir la valeur maximale h conduisant à la liste RCL

Sortie

S	Solution vide au début, contient la solution construite à la fin

Variables

$minMkpn, maxMkpn$	Plus petit ou plus grand makespan à chaque étape
h	Makespan autorisé pour insérer une opération dans la séquence (calculé)
RCL	Liste restreinte de candidats formée des jobs retenus après calcul de h
$infinite$	Nombre entier très grand
$mkpn$	Makespan temporaire pour la recherche d'une machine pour une opération

Début

1. $S \coloneqq \emptyset$;
2. **POUR** $i \coloneqq 1$ **A** $data.tailleProbleme$ **FAIRE**
3. $minMkpn \coloneqq infinite$; $maxMkpn \coloneqq 0$;
4. **POUR CHAQUE** jobs dont une opération reste à planifier **FAIRE**
5. $mkpn \coloneqq infinite$;
6. Calculer chaque valeur importante pour l'opération (makespan, date de début, …) ;
7. Mettre à jour $minMkpn$ si le makespan est plus petit ;
8. Mettre à jour $maxMkpn$ si le makespan est plus grand ;
9. **FIN POUR**
10. Calculer $h \coloneqq minMkpn + \beta * (maxMkpn - minMkpn)$;
11. Construire RCL avec les jobs dont le makespan est plus petit que h ;
12. Choisir aléatoirement un job dans RCL ;
13. Insérer le job sélectionné dans le vecteur par répétition de S ;
14. Insérer la machine dans le vecteur d'affectation de S ;
15. Sauvegarder toute valeur définitive (date de début, date de fin, …) ;
16. **FIN POUR**
17. Retourner S ;

Fin

5.1.2. Phase de recherche locale

Pour améliorer la qualité des solutions, il est nécessaire d'échanger l'ordre de deux opérations consécutives partageant la même machine sur le chemin critique. Pour le problème actuel, la recherche locale explore le chemin critique. Pour chaque opération sur le chemin, l'algorithme procède à une permutation des opérations si elles sont en fin ou en début de bloc ; le voisinage de (Nowicki and Smutnicki, 1996) est donc utilisé dans cette recherche locale. La recherche locale commence à explorer le chemin critique (lignes 2-13). Lorsque la recherche locale détecte deux opérations en disjonction en tête ou en queue d'un bloc (ligne 5), elle modifie l'ordre de traitement des opérations sur la machine (ligne 6). Si la solution obtenue après permutation est de meilleure qualité, elle est conservée, et la recherche locale est réappliquée à partir de la fin du nouveau chemin critique. Sinon, le parcours du chemin critique continue en modifiant la valeur de l'opération courante. Comme la recherche locale pour l'ordonnancement est une procédure clé de la recherche locale globale, elle est présentée dans l'Algorithme 3.

Algorithme 3 : Recherche_Locale

Entrée
$data$ Structure stockant les données du problème
Entrée/Sortie
S Solution à améliorer
Variables
op, $prec$ Opération et son prédécesseur sur le chemin critique
$mkpn$, Makespan temporaire après estimation
$savOp$, $savPrec$ Sauvegarde d'une opération et de son prédécesseur
$newS$ Solution temporaire

Début
1. $op \coloneqq$ prédécesseur de $*$; //on initialise *op* à la fin du chemin critique
2. **TANT QUE** $op \neq 0$ **FAIRE**
3. **SI** (*op* a un prédécesseur disjonctif) **ALORS**
4. $prec \coloneqq$ prédécesseur disjonctif de op ;
5 **SI** op et $prec$ sont en fin ou en début de bloc **FAIRE**
6 Appliquer la permutation de op et $prec$ dans $newS$; // on fait la permutation dans une copie de *S*
7 **Evaluer**$(data, newS)$;
8. **SI** $f(newS)$ est meilleur que $f(S)$ **ALORS**
9. $S \coloneqq newS$; $op \coloneqq *$; // on repart de la fin du chemin critique
10. **FIN SI**
11. **FIN SI**
12 $op \coloneqq$ prédécesseur de op sur le chemin critique ;
13. **FIN TQ**
14. Retourner S ;
Fin

5.1.3. Génération de voisins

Dans le cadre de l'ELS, utilisé en association du GRASP, il est nécessaire de générer des voisins de la solution courante. Pour la métaheuristique mise en œuvre, il est choisi de générer un voisin en appliquant une permutation sur deux valeurs du vecteur par répétition dès lors qu'elles appartiennent à deux jobs différents. Cette procédure peut donc générer un voisin représentant le vecteur en bas de la Figure 11 à partir du vecteur situé en haut.

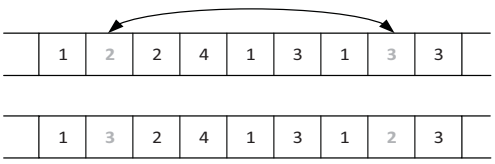

Figure 11 Exemple d'une permutation dans le vecteur par répétition

5.2. Résultats expérimentaux

La métaheuristique a été appliquée sur des instances classiques de la littérature issues des instances de (Lawrence, 1984). Ces instances sont notées laXX dans la suite. Leur taille varie de 50 à 300 opérations, de 10 à 30 jobs, et de 5 à 15 machines. Ces instances sont souvent utilisées telles quelles pour les problèmes de Job-shop, mais également en les modifiant afin de traiter des extensions du problème.

Avant d'appliquer la métaheuristique aux instances proposées dans le jeu de données, un plan d'expérience a été mené pour choisir les paramètres. Pour ce plan d'expérience, plusieurs configurations du GRASP×ELS ont été testées et sont résumées ci-dessous :

- Le nombre d'exécutions du GRASP, noté nb_g, va de 200 à 300 par pas de 50 ;
- Le nombre d'exécutions de l'ELS, noté nb_els, va de 100 à 200 par pas de 20 ;
- Le nombre de voisins générés, noté nb_n, va de 20 à 50 par pas de 1.

Le plan d'expérience a conduit à la sélection des paramètres suivants : nb_g est fixé à 300, nb_els est fixé à 200 et nb_n est fixé à 33. Pour chaque instance, 10 réplications sont réalisées.

Les algorithmes ont été implémentés en C ++ et ont été exécutés sur un ordinateur avec un processeur i7-4800MQ, exécutant Windows 7. Afin d'avoir une comparaison équitable entre les méthodes, les temps de calcul sont normalisés selon le processeur utilisé. Le facteur de vitesse est calculé en utilisant les données de l'université de Berkeley qui a appliqué le benchmark de Whetstone sur de nombreux ordinateurs. Une comparaison avec l'étude de (Dongarra, 2014) montre que les GFLOPS obtenus avec Whetstone sont comparables (Intel Core 2 Q6600 donné à 2.4GFLOPS dans (Dongarra, 2014), 2.6GFLOPS avec Whetstone), mais il existe plus d'informations disponibles sur le site web de Berkeley ce qui justifie l'utilisation de Whetstone. En outre, comme chaque valeur des GFLOPS rapportée dans le Tableau 4 est obtenue avec le même benchmark, les temps de calcul peuvent être facilement comparés et le facteur de vitesse devrait être comparable à d'autres benchmarks. Le facteur de vitesse est obtenu en divisant la valeur de GFLOPS des ordinateurs utilisés pour la comparaison par la valeur de GFLOPS de l'ordinateur utilisé pour tester le GRASP×ELS. Comme il existe parfois plusieurs valeurs dans le fichier pour une même configuration, deux données sont présentées pour les GFLOPS : $GFLOPS^{+}$, qui correspond à la meilleure valeur pour la configuration de l'ordinateur, et $GFLOPS^{-}$ qui correspond à la valeur la plus faible. Lorsque la dénomination du processeur utilisé dans un article n'est pas assez précise, le GFLOPS le plus bas est sélectionné afin de ne pas désavantager les travaux correspondants. Le facteur de vitesse est calculé à partir de la valeur $GFLOPS^{-}$. Cependant, il convient de noter, comme l'ont déclaré (Yuan and Xu, 2013a) que l'information sur les temps de calcul est purement indicative car elle dépend de divers facteurs tels que les compétences en programmation, le langage de codage, ou même le système d'exploitation.

Tableau 4 Performances des ordinateurs utilisés pour comparaison

	Ce travail	Zhang et al., 2008	Quing et al., 2012	Gao et al., 2015	Kurdi 2015	Cheng et al., 2016
Ordinateur	i7 4800MQ 2,7GHz	Pentium IV 3Ghz	Intel Core 2 duo 2,9GHz	Xeon 2,83Ghz	i7 3770 3,4Ghz	AMD Opteron 2,8Ghz
$GFLOPS^{+}$	3,51	1,67	3,23	2,16	4,41	2,68
$GFLOPS^{-}$		1,00	2,18	1,67	3,36	2,04
Facteur vitesse	1,00	0,28	0,62	0,48	0,96	0,58

*https://setiathome.berkeley.edu/cpu_list.php

Dans le tableau suivant, une synthèse des résultats sur le jeu de données est proposé. Les résultats sont comparés aux articles présentés dans la section état de l'art de ce chapitre. Les instances sont regroupées en lot en fonction de la taille de problème. **INS** renvoie à l'ensemble d'instances testées. $\overline{\mathbf{D^{+}}}$ correspond à la moyenne des déviations entre la meilleure solution trouvée et la borne inférieure pour chaque instance d'un lot. $\overline{\mathbf{D^{-}}}$ correspond à la moyenne des déviations

entre le makespan moyen et la borne inférieure pour chaque instance d'un lot. $\overline{\mathbf{CPU}}$ est le temps de calcul moyen en secondes pour chaque lot d'instances. Pour chaque article consulté pour ce travail, $\overline{\mathbf{CPU}}$ est obtenu en appliquant le facteur de vitesse au temps moyen de calcul indiqué par les auteurs. Lorsqu'une valeur est manquante, un tiret (-) est inséré dans le tableau.

Tableau 5 Synthèse des résultats sur le Job-shop

	Zhang et al., 2008			Quing et al., 2012			Cheng et al., 2016			Gao et al., 2015			Kurdi, 2015			Ce travail		
	TSSA			HGA1			HEA			ISCE			ISCE			GRASP×ELS		
INS	$\overline{D^+}$	$\overline{D^-}$	$\overline{CPU}$	$\overline{D^+}$	$\overline{D^-}$	$\overline{CPU}$	$\overline{D^+}$	$\overline{D^-}$	$\overline{CPU}$	$\overline{D^+}$	$\overline{D^-}$	$\overline{CPU}$	$\overline{D^+}$	$\overline{D^-}$	$\overline{CPU}$	$\overline{D^+}$	$\overline{D^-}$	$\overline{CPU}$
la01-05	0	0	0	0	-	-	0	0	0	0	-	0	0	0,01	3	0	0	0
la06-10	-	-	-	0	-	-	0	0	0	0	-	0	0	0	6	0	0	0
la11-15	-	-	-	0	-	-	0	0	0	0	-	0	0	0	13	0	0	0
la16-20	0	0	0	0,16	-	-	0	0	0	0	-	0	0	0,17	23	0	0,01	2
la21-25	0	0,03	4	0,64	-	-	0	0	7	0	-	2	0	0,44	212	0	0,24	20
la26-30	0,02	0,02	4	0,16	-	-	0,02	0,02	11	0,28	-	3	0,02	0,40	344	0,78	1,09	25
la31-35	-	-	-	0	-	-	0	0	1	0	-	2	0	0	282	0	0	0
la36-40	0,03	0,19	10	0,56	-	-	0	0,01	126	0,03	-	4	0,03	0,65	1002	0,33	1,03	40

Dans le Tableau 5, il est possible de voir que les résultats obtenus à l'aide du GRASP×ELS sont de bonne qualité. Au moins une solution optimale est trouvée pour chaque instance allant de la01 à la25, et de la31 à la35, dans des temps raisonnables. Sur les instances la21-25, la métaheuristique est très proche des autres travaux, bien que plus lente en général, hormis par rapport aux travaux de (Kurdi, 2015), qui est près de 10 fois plus lent. Le GRASP×ELS semble cependant moins adapté sur les instances la26-30 et la36-40, mais les résultats observés restent de bonne qualité et sont obtenus en des temps de calcul raisonnables, voire meilleurs que (Cheng et al., 2016 ; Kurdi, 2015). En effet, il est important lors de la définition d'une métaheuristique de trouver un compromis entre temps de calcul et qualité des solutions. Il ressort de ce constat que les travaux de (Zhang et al., 2008) semblent les meilleurs parmi les articles utilisés pour comparaison compte tenu de la date de publication.

6. Conclusion

Dans ce chapitre le problème théorique du Job-shop a été étudié car c'est un problème classique de la littérature sur l'ordonnancement. D'une part ce problème est un sous-problème du Job-shop Flexible et il généralise les problèmes à une machine ou encore le Flow-shop, d'autre part il est à l'origine des problèmes tels que le Job-shop avec transport (Lacomme et al., 2013a), le Job-shop avec Time-lags (Caumond et al., 2008) et le Job-shop avec flux financiers (Kemmoe et al., 2015a). Dans ce chapitre différents outils et méthodes ont été mis en évidence et utilisés pour résoudre le problème de Job-shop. Plus spécifiquement, il a été montré que les voisinages reposant sur le chemin critique sont des outils performants dans l'exploration de l'espace des solutions. Associés à des métaheuristiques ou des recherches locales, ils permettent d'obtenir de bonnes solutions.

Une métaheuristique a également été proposée pour traiter le problème du Job-shop. Elle repose sur un schéma de type GRASP×ELS. La procédure de construction des solutions, la recherche locale et la génération de voisins dans la phase d'ELS sont détaillées. La métaheuristique ainsi définie fournit des résultats de bonne qualité à la fois en termes de temps de calcul et de performance des solutions, ce qui montre la pertinence du GRASP×ELS dans la résolution de problèmes d'ordonnancement. Au cours de cette thèse il a également été proposé de paralléliser le GRASP×ELS à l'aide de la plateforme STORM pour la résolution du Job-shop avec Time-lags

(Avez et al., 2016). STORM est un outil implémentant le principe de MapReduce en temps-réel et permettant le traitement d'importantes quantités de données. L'outil permet, de par une approche maitre/esclave, de définir une topologie à partir d'ordinateurs reliés à un réseau (interne ou externe). Il est alors possible d'effectuer du calcul distribué et de paralléliser certaines procédures afin d'améliorer les performances d'un modèle d'optimisation. La parallélisation des métaheuristiques figure parmi les perspectives de cette thèse car cela permet de réduire considérablement les temps de calcul ressentis.

Les outils présentés dans ce chapitre ont donc été considérablement utilisés dans cette thèse et sont réutilisés par la suite, notamment le graphe conjonctif, la notion de recherche locale ainsi que le GRASP×ELS.

Chapitre III : Problème de Job-shop avec contraintes énergétiques

Dans ce chapitre une problématique importante du point de vue sociétale est abordée. Il s'agit de l'optimisation des systèmes de production en prenant en compte la réduction de leurs consommations énergétiques. Le choix s'est porté sur la réduction de la puissance injectée au système de production. Il s'agit en effet du problème de Job-shop avec des contraintes énergétiques. Un modèle mathématique et deux métaheuristiques sont proposés. Une première (GRASP×ELS) est appliquée pour le problème avec seuil de puissance variable, et une autre métaheuristique (Hybrid NSGA-II) pour le problème biobjectif associé.

Sommaire

Liste des figures

Liste des tableaux

Liste des algorithmes

1. Introduction

De nos jours, le secteur industriel dans le monde entier est responsable de la consommation de près de 50% de l'énergie distribuée à l'échelle mondiale par an (Giret et al., 2015). Cette consommation devrait augmenter de plus de 50% d'ici 2040. Les ressources énergétiques représentent donc un poste de dépense important pour les entreprises qui ont tendance à modifier leurs pratiques pour réduire leur consommation. Par exemple, (Gutowski et al., 2005) ont mené une étude sur près de 50 interlocuteurs, entreprises ou laboratoires, au Japon, en Europe et aux Etats-Unis. Leur étude a révélé que la plupart des entreprises étaient relativement impliquées dans l'impact qu'elles avaient sur l'environnement (en partie pour des raisons fiscales). De plus, certaines compagnies tentent de généraliser leurs bonnes pratiques à leurs différents sites et leurs sous-traitants. Ces pratiques peuvent par exemple consister à exclure l'usage de certains matériaux. Ce type de régulation volontaire conduit à l'obtention de certifications. Cependant les entreprises conçoivent leurs propres systèmes de management environnemental qui peuvent être plus stricts que ce que préconise la norme ISO14000, c'est le cas du groupe Chrysler par exemple (Gutowski et al., 2005). Il est également noté par les auteurs que ce type de certification est perçu comme un obstacle pour certaines compagnies, principalement américaines et japonaises. En Europe, la présence de partis politiques impliqués dans la protection de l'environnement a permis d'informer tôt sur les enjeux écologiques. Le développement de l'UE conduit également les entreprises à revoir leurs infrastructures. De plus l'introduction de taxes environnementales implique des prises de décision mesurées. Les Etats-Unis s'attachent particulièrement à améliorer les outils et les processus utilisés, bien que le management des chaînes de production soit encore plus important en UE. Cependant, la réponse des entreprises à ces problèmes est encore limitée, en partie à cause de l'écart entre la recherche et les attentes industrielles (Giret et al., 2015). Pourtant, de nombreuses recherches sont menées sur ces problèmes comme en attestent les nombreux journaux scientifiques ayant pour thématique les énergies renouvelables tels que « Journal of Cleaner Production » ou « Applied Energy ».

Parmi les travaux conduits sur la thématique de l'optimisation énergétique dans le milieu industriel, il ressort que deux décisions sont possibles : technologique et/ou organisationnelle (Duflou et al., 2012). Les mesures technologiques conduisent à de fortes réductions mais sont très coûteuses car elles se concentrent sur de nouvelles machines ou sur la modification des processus de production, tandis que les mesures organisationnelles mettent l'accent sur l'amélioration du système actuel, ce qui conduit à une meilleure efficience énergétique. Comme souligné dans (Zhang and Chiong, 2016), des améliorations significatives peuvent être réalisées avec les méthodes de planification prenant en compte l'utilisation de l'énergie pendant la fabrication. Dans la littérature, les approches d'optimisation de la planification avec pour objectif la réduction des coûts liés à la consommation énergétique concernent principalement la consommation totale d'énergie (Liu et al., 2014 ; May et al., 2015b ; Salido et al., 2016b). Cependant, une autre façon de réduire les coûts énergétiques dans les systèmes de fabrication est de gérer la production sans dépasser un seuil de puissance contractualisé (Nolde and Morari, 2010). Cette approche est très différente de celle qui vise à minimiser le coût énergétique total, car elle considère de manière plus approfondie la puissance d'entrée dont un système de production a vraiment besoin. Cela a notamment un impact direct sur la production d'électricité pour les fournisseurs, comme l'ont souligné (Merkert et al., 2015). Les auteurs notent également que l'émergence de sources renouvelables d'énergie (énergie solaire, énergie éolienne, etc.) implique un changement dans la façon dont les ressources énergétiques sont consommées par les entreprises. En effet, la production d'énergie basée sur ces ressources peut fluctuer avec le temps et le stockage de

l'énergie est encore difficile ; par conséquent, les fournisseurs pourront être amenés dans le futur à demander à leurs clients de réduire leur consommation pendant une période donnée. Une telle approche peut être vue comme une application de règles d'interdiction d'utilisation de ressources énergétiques (*rolling blackout policy*) telles que présentées dans (Liu et al., 2015) où les auteurs mentionnent qu'en Chine, les entreprises «[...] souffrent d'une politique de restrictions quant à l'utilisation de l'électricité, ce qui signifie que l'électricité peut être coupée plusieurs jours dans chaque semaine [...] ".

Compte tenu de cela, une planification efficace des opérations en prenant en compte les périodes où la puissance disponible est insuffisante est un objectif important pour l'industrie grâce à la possibilité d'avoir non seulement une interruption complète, mais aussi des périodes avec une puissance disponible réduite (ce qui modélise les contrats avec les fournisseurs d'électricité, où l'électricité disponible peut varier avec le temps, comme c'est le cas avec EDF en France par exemple). Le problème revient à gérer, du côté de la demande - *demand-side management* - la façon dont les ressources sont consommées afin de réduire la production d'énergie du côté des fournisseurs (Gupta and Venkataraman, 2013). Ainsi, il revient à l'entreprise de gérer et d'adapter l'ordonnancement des opérations à réaliser en considérant une limitation des capacités énergétiques. Avec une telle approche, il est possible pour une entreprise de réduire les dépenses supplémentaires liées à un dépassement de la puissance autorisée en respectant rigoureusement son contrat avec le fournisseur.

À cette fin, il est proposé d'aborder l'impact de la puissance consommée (PC) sur les ordonnancements dans le cadre d'un atelier de fabrication de type Job-shop. Dans le premier chapitre, il a été montré que ce problème était la forme basique des problèmes d'ordonnancement qu'il était possible d'obtenir à partir d'un problème impliquant un atelier d'élastomère. Une méthode de résolution a été proposée pour le problème de Job-shop (cf. chapitre II, §5). Dans ce chapitre les méthodes existantes pour le problème de Job-shop sont adaptées afin de prendre en compte les contraintes énergétiques.

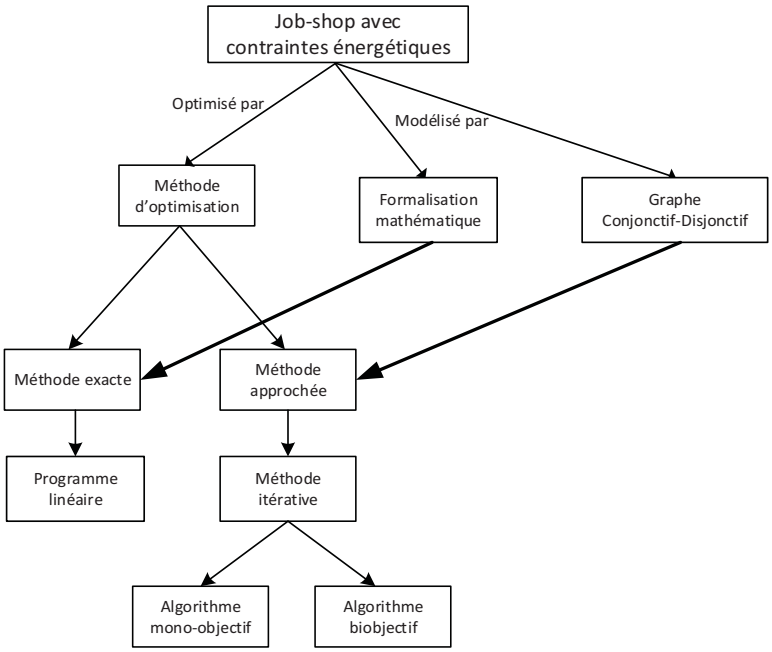

Figure 1 Approches pour le Job-shop avec contraintes énergétiques

La Figure 1 est une présentation graphique des travaux réalisés pour le Job-shop avec contraintes énergétiques. Ces travaux sont organisés en deux parties : une approche de modélisation du problème et une approche de résolution à l'aide de méthodes d'optimisation. Les modèles proposés sont une formalisation mathématique ainsi qu'une modélisation sous forme de graphe conjonctif-disjonctif. Ces deux modèles sont utilisés pour définir des méthodes d'optimisation : la formalisation mathématique permet de réaliser un programme linéaire en nombres entiers alors que le graphe conjonctif-disjonctif est utilisé pour concevoir des méthodes approchées, la première optimisant un seul critère, tandis que la deuxième permet la résolution de problèmes biobjectifs.

Deux méthodes d'optimisation complémentaires sont donc proposées : le programme linéaire permet de déterminer les solutions optimales sur des instances de petite taille. En partant des résultats obtenus, il est possible de vérifier la pertinence des méthodes approchées, qui peuvent être employées sur des instances de taille plus importante lorsque le programme linéaire est dans l'incapacité de proposer des solutions optimales. S'il n'est pas garanti que ces méthodes convergent vers la solution optimale, les paramétrer par rapport aux connaissances acquises sur des instances de plus petite taille permet d'estimer leur performance lors du changement d'échelle. D'après le synoptique, deux types de métaheuristiques sont proposés. Leur fonctionnement repose sur différentes briques logicielles qui seront présentées dans la suite.

Dans la section suivante une revue des travaux réalisés dans le domaine de l'ordonnancement avec prise en compte des contraintes énergétiques est présentée. Dans la troisième section, un problème d'ordonnancement avec seuil de puissance variable est introduit, suivi dans la quatrième section par un problème biobjectif. Le chapitre s'achève sur une conclusion ainsi que sur les perspectives liées aux travaux réalisés jusqu'à ce jour.

2. Etat de l'art

L'optimisation de la consommation énergétique fait aujourd'hui partie des préoccupations importantes des entreprises, qui s'efforcent d'améliorer leur efficacité énergétique (Giret et al., 2015 ; Gutowski et al., 2005), et aussi pour la société qui est directement impactée par tout type de processus de fabrication. Ainsi, les consommations d'énergie et les émissions des systèmes de production sont maintenant prises en considération dans les processus de décision. Selon l'administration américaine sur l'information énergétique (EIA), une hausse de 25% de la consommation énergétique dans le secteur industriel est à prévoir d'ici 2040 (*Annual Energy Outlook*, 2017). En outre, les solutions proposées aux entreprises confrontées à ces problèmes sont toujours limitées.

Pour améliorer la consommation d'énergie des systèmes de production, (Duflou et al., 2012 ; Rager et al., 2015) suggèrent que deux types de mesures peuvent être prises : des mesures technologiques et/ou organisationnelles. Les mesures technologiques consistent à remplacer les anciennes machines par de nouvelles moins gourmandes en énergie ou à modifier les procédés de fabrication, alors que les mesures organisationnelles se concentrent sur l'amélioration du système existant en s'intéressant à la planification et à l'ordonnancement, ce qui entraîne une réduction de la consommation énergétique tout en nécessitant des apports financiers restreints car aucun actif supplémentaire n'est à acquérir.

La plupart des problèmes d'ordonnancement dans la littérature consistent à minimiser le makespan (le temps de traitement total de toutes les opérations), le retard total ou d'autres fonctions objectif temporelles. Jusqu'à récemment, seuls quelques travaux traitaient de l'optimisation de l'énergie comme un critère essentiel dans l'ordonnancement. Cependant, la minimisation de la consommation

énergétique dans les systèmes de production est un sujet de recherche de plus en plus présent dans la littérature, comme l'ont souligné (Giret et al., 2015).

Pour les industriels, les plans de production qui ne prennent pas en compte les aspects énergétiques risquent de produire des ordonnancements aux performances réduites entraînant une perte financière et une empreinte environnementale négative (Biel and Glock, 2016). Pour améliorer la performance énergétique des entreprises plusieurs approches peuvent être trouvées dans la littérature, comme la minimisation des coûts dans le contexte d'une tarification de l'énergie consommée dépendant de l'instant où l'énergie est consommée (Luo et al., 2013 ; Shrouf et al., 2014), la minimisation de la consommation totale d'énergie (Dai et al., 2013 ; Liu and Tiwari, 2015), la prise en compte d'un seuil de puissance qui ne doit pas être dépassé (Bruzzone et al., 2012 ; Liu and Huang, 2014) ou encore la réduction des émissions de carbone (Fang et al., 2011). Ces problèmes interviennent sur différents systèmes de production tels que les systèmes à une machine (Mouzon and Yildirim, 2008 ; Shrouf et al., 2014), les machines parallèles (Moon et al., 2013 ; Tonelli et al., 2016), les problèmes de Flow-shop et Flow-shop Hybride (Bruzzone et al., 2012 ; Ding et al., 2016 ; Fang et al., 2011) et les problèmes de Job-shop et de Job-shop Flexible (Lei et al., 2016 ; May et al., 2015b). Enfin, les méthodes de résolution mises en œuvre peuvent être des programmes linéaires (Bruzzone et al., 2012 ; Liu, 2016 ; Mouzon et al., 2007) et des heuristiques (Che et al., 2016 ; Wang et al., 2016).

Certains de ces travaux sont abordés dans la suite et une vue d'ensemble des problèmes traités par rapport à la thématique énergétique et les objectifs de ces études est proposée dans le Tableau 1 où les termes *ass*, *ind*, et *c* signifient respectivement *assimilables*, *induit* et *contrainte*. Ces termes sont utilisés pour alléger le tableau, lorsque les éléments ne sont pas suffisamment précis dans les articles, ou qu'ils ne justifient pas l'ajout d'une colonne. Seul le travail de (Salido et al., 2016a) bénéficie d'une colonne concernant la robustesse car ce critère semble intéressant et très peu traité dans la littérature.

Trois états de l'art récents ont été proposés par (Biel and Glock, 2016), (Giret et al., 2015) et (Gahm et al. 2016). Les premiers se sont intéressés aux modèles d'aide à la décision intégrant l'aspect énergétique à court et moyen terme dans les problèmes d'ordonnancement des systèmes de production. Les auteurs considèrent aussi les approches faisant intervenir lot-sizing et planification à capacité finie, et pas seulement les aspects orientés sur l'ordonnancement des systèmes de production. Il est également noté que même si le nombre de travaux prenant en compte des contraintes énergétiques a augmenté de manière significative, de nombreuses études ouvrent des perspectives sans qu'elles soient suivies d'autres travaux allant dans la même direction. Parmi les travaux recensés, la réduction des consommations via des approches managériales des processus de fabrication, appelées « energy-efficient production planning » (EEPP) sont mises en avant. Ces dernières sont devenues populaires car elles ne nécessitent pas de forts investissements (contrairement à l'acquisition de machines). Deux tendances se distinguent dans leur étude : *Load Management* et *Load tracking*. Elles sont spécifiquement conçues pour favoriser l'efficience énergétique. L'objectif est double : pousser les clients des fournisseurs d'énergie (EDF, …) à réduire leur consommation et pénaliser ceux qui ne respecteraient pas la courbe de consommation qui leur est accordée après contractualisation. (Giret et al., 2015) proposent un état de l'art sur les méthodes de résolution de problèmes d'ordonnancement faisant intervenir des considérations énergétiques. Les articles sont classés selon plusieurs groupes et sous-groupes. Les auteurs notent que peu d'approches combinent *entrées* (énergie consommée, puissance injectée, …) et *sorties* (empreinte carbone, …) dans l'ordonnancement des systèmes de production. Un autre point concerne les approches réactives qui devraient être plus représentées dans la littérature selon les auteurs. Un manque de benchmarks est aussi relevé, et notamment l'absence de données sur des systèmes de production qui permettraient de valider les approches. (Gahm et al. 2016) proposent également un cadre de recherche et une classification des problèmes d'ordonnancement

énergétiquement efficace. La littérature est ordonnée selon plusieurs catégories qui font apparaître les problématiques de tarification en temps réel, par tranche d'heures, ou les pénalités liées aux pics de consommations. La façon dont l'énergie est utilisée est également mentionnée, avec les phénomènes de pertes liés à l'inactivité des machines, ou aux vitesses d'usinage.

Tableau 1 Vue d'ensemble des problèmes d'ordonnancement avec contraintes énergétiques

		Objectifs énergétiques					Objectifs temporels	
		Energie totale	Coûts	Puissance	CO_2	Robustesse	Makespan	Retard
Une machine	(Mouzon et al., 2007)	x					x	
	(Mouzon and Yildirim, 2008)	x						x
	(Wang et al., 2016)		x				x	
	(Shrouf et al., 2014)		x					
	(Liu, 2016)				x		x (ass)	
	(Che et al., 2016)		x				x	
Machines parallèles	(Moon et al., 2013)		x				x (ind)	
	(Grimes et al., 2014)		x					x
	(Rager et al., 2015)			x (ass)				
	(Tonelli et al., 2016)	x						x
	(Ding et al., 2016)		x					x (ass)
Flow-shop	(Fang et al., 2011)			x	x		x	
	(Zhang et al., 2014)		x	x			x (c)	
	(Lin et al., 2015)				x		x	
	(Liu et al., 2016)	x						
	(Ding et al., 2016)				x		x	
	(Lu et al., 2017)	x					x	
Flow-shop Hybride/Flexible	(Nolde and Morari, 2010)			x (ass)			x (c)	
	(Bruzzone et al., 2012)			x (c)			x	x
	(Dai et al., 2013)	x					x	
	(Luo et al., 2013)		x				x	
	(Sharma et al., 2015)		x		x		x	
	(Tang et al., 2016)	x					x	
Job-shop	(Liu et al., 2014)	x						x
	(Liu et al., 2015)	x	x					x
	(May et al., 2015b)	x					x	
	(Tang and Dai, 2015)	x					x	
	(Zhang and Chiong, 2016)	x						x
	(Salido et al., 2016b)	x					x	
	(Salido et al., 2016a)	x				x	x	
Job-shop Flexible	(Pach et al., 2014)	x					x	
	(Moon and Park, 2014)		x				x (ind)	
	(He et al., 2015)	x					x	
	(L. Zhang et al., 2015)	x					x	x
	(Garcia-Santiago et al., 2015)	x						x (c)
	(Stock and Seliger, 2015)	x					x	
	(Li and Cao, 2015)				x			
	(Liu and Tiwari, 2015)	x					x	
	(Lei et al., 2016)	x					x (ass)	
	(Zhang et al., 2016)	x					x	x
	(Yin et al., 2016)	x					x	
	(Piroozfard et al., n.d.)				x			x

Il ressort de ces états de l'art que plusieurs approches existent : la minimisation de la consommation énergétique, du coût de l'énergie, ou la réduction des émissions de gaz à effet de serre. La prise en compte de contraintes telles que le non dépassement d'une puissance contractualisée ou le respect d'un seuil d'émission de dioxyde de carbone à ne pas dépasser font également partie des approches recensées dans la littérature. Dans la suite, l'ordre appliqué dans le Tableau 1 est suivi pour la description de certains articles de la littérature.

2.1. Les problèmes à une machine

Les articles de (Mouzon et al., 2007 ; Mouzon and Yildirim, 2008) sont précurseurs. Les auteurs s'y intéressent à l'optimisation de la consommation énergétique dans le cadre d'un atelier à une machine où l'objectif est de minimiser la durée totale de traitement des jobs et la consommation énergétique totale. Le premier article présente différentes approches avec notamment des règles de gestion de l'état des machines dans le contexte d'un traitement par batch des produits, ou sans batch. Ils remarquent que la plupart du temps les machines sont inactives et que d'importantes économies peuvent être réalisées en les arrêtant. Il est par exemple noté qu'il est commun de trouver des machines n'étant pas des goulots d'étranglement, souvent laissées inactives ; il est alors possible via une règle de gestion de définir quand la machine doit s'arrêter. Alors que, selon leurs observations, une pratique courante vise à garder les machines inactives en moyenne 16% du temps, les éteindre peut faire diminuer la consommation en électricité de 13%. Ils proposent des règles de gestion pour minimiser l'énergie consommée. Une optimisation par réseau de neurones est ensuite proposée afin de prévoir les durées inter-arrivées des ordres de fabrication. Grâce à cet outil d'aide à la décision, couplé aux règles de gestion, un responsable de la production peut décider d'éteindre une machine ou de la laisser inactive en vue des prochaines pièces à usiner. Une approche pour le problème biobjectif est également proposée. Dans le second article, les auteurs proposent un modèle linéaire du problème étudié ainsi qu'un GRASP dont le rôle est de minimiser l'énergie totale consommée ainsi que le retard total pour un problème constitué d'une seule machine.

(Wang et al., 2016) traitent d'un problème faisant état d'une machine unique dans un contexte d'ordonnancement par lots. La taille des lots est variable d'un job à un autre. Leur objectif est de minimiser le makespan et le coût total en énergie, impliquant à la fois l'utilisation des machines et le coût économique lié à la production. Une approche exacte ayant pour objectif de construire le front de Pareto optimal, et deux heuristiques sont développées pour traiter les instances de taille importante.

(Shrouf et al., 2014) proposent l'étude d'un processus impliquant une machine où il est question des coûts énergétiques en fonction de plages horaires ainsi que des transitions entre les états de la machine (active, éteinte, en attente). Pour résoudre le problème un algorithme génétique est suggéré. Cet algorithme prend en entrée une séquence de passage prédéterminée des opérations sur la machine. Les résultats sont obtenus en faisant varier les coûts appliqués à la consommation énergétique. Une fois encore la métaheuristique démontre son intérêt vis-à-vis d'approches exactes lorsque la taille du problème augmente. Elle permet l'obtention de solutions optimales dans la plupart des cas et quasi-optimale dans les autres, avec des temps de calcul compétitifs dans tous les cas. Ils relèvent le fait que leur algorithme est particulièrement intéressant dans les cas suivants : (i) lorsque la variabilité des prix est grande et que la demande est faible dans les plages horaires où les prix sont élevés ; (ii) lorsque l'énergie consommée est élevée entre les transitions ; (iii) lorsque les périodes d'attente sont longues. Ils pointent également le fait que leur algorithme peut être utilisé en extension d'un MRPII (*Manufacturing Ressource Planning*) ce qui corrobore l'étude menée par (Bruzzone et al., 2012)

visant à mettre en place des solutions qui viennent en support de celles déjà employées pour l'ordonnancement.

(Che et al., 2016) présentent un programme linéaire d'un système de production à une machine avec tarification énergétique dépendant du temps. Leur modèle n'a pour objectif que la minimisation des coûts associés à la consommation en électricité, en considérant le makespan de la production totale connu à l'avance (égal à la somme de traitement de tous les jobs). L'objectif est d'ordonnancer l'ensemble des jobs en réduisant le coût requis pour la production totale. Une heuristique d'insertion des jobs en fonction des tarifs énergétiques en cours est proposée. Cette heuristique est capable de proposer des ordonnancements pertinents pour des problèmes de taille industrielle en moins d'une seconde. Une réduction de 30% des coûts liés à la consommation en électricité est envisageable selon leur étude.

(Liu, 2016) traite d'un problème faisant intervenir une seule machine soumise à une ressource énergétique issue des énergies renouvelables. L'objectif est de réduire la consommation en dioxyde de carbone et d'améliorer la performance d'une entreprise. Du fait du caractère intermittent de la disponibilité d'énergie caractéristique des ressources renouvelables, la production peut être impactée. Le principal problème étudié cherche à minimiser un critère de performance temporel ainsi que les émissions en dioxyde de carbone. Un front de Pareto est obtenu par application successive d'un modèle mathématique.

2.2. Les problèmes à machines parallèles

Dans (Moon et al., 2013), les auteurs travaillent sur un système de type machines parallèles indépendantes. Les jobs ont tous la même date d'échéance prévisionnelle. Les auteurs cherchent alors à optimiser le coût total de la production en intégrant le coût lié à la durée de la production (makespan) en y ajoutant le coût de la production du point de vue de la consommation en électricité. Cette dernière dépend de la période de la journée mais aussi des machines utilisées. Du fait de la complexité du problème, un algorithme génétique hybride est développé. Pour réduire les coûts en électricité les auteurs intègrent des jobs fictifs dans l'ordonnancement afin de décaler les dates de début des opérations dans des zones dont les tarifs sont plus profitables. L'insertion de ces jobs se fait aléatoirement par une procédure dédiée qui repose sur les coûts en électricité des périodes, l'objectif étant de ne pas dépasser la date d'échéance tout en réduisant les coûts. Sur de petites instances, ils montrent que des gains significatifs peuvent être obtenus, tels que 13% en moyenne sur des tarifs pratiqués durant l'été, et 22% en moyenne sur les tarifs hivernaux. Sur des instances de taille plus importantes (40 à 60 jobs), ces valeurs sont respectivement de 16% et 12% ce qui suggère que l'approche retourne de bons résultats pour un environnement à machines parallèles.

(Rager et al., 2015) proposent une approche de planification orientée énergie dans le contexte d'un environnement à machines parallèles identiques. Leur objectif est de minimiser la demande en énergie. Ils appliquent un jeu de données issue de l'industrie textile. Dans le problème une fois qu'un job est commencé, ses opérations doivent s'enchaîner sans interruption sur la même machine. En d'autres termes chaque opération modélise le profil de puissance du processus complet. Dans leur cas ils doivent chauffer des chaudières qui transmettent la chaleur à des unités terminales. Ils maximisent l'utilisation de ces unités principales afin d'éviter de relancer les chaudières à n'importe quel moment. La méthode approchée fonctionne en deux temps et repose sur des algorithmes évolutionnaires. Dans un premier temps les jobs sont affectés aux machines en équilibrant au mieux la charge. La deuxième étape permet de déterminer les dates de début des opérations. L'optimisation de la consommation énergétique est gérée par un processus de recherche locale visant à décaler des opérations vers la droite.

Ils estiment également le coût et les émissions de dioxyde de carbone associés aux solutions retournées par leur approche. Selon eux 20% d'amélioration est observée sur ce dernier critère.

Dans (Tonelli et al., 2016) les auteurs s'intéressent à l'ordonnancement *off-line* d'un système de production impliquant des machines parallèles indépendantes et des temps de préparation des machines. Ils appliquent leur approche à un atelier d'injection. Un modèle mathématique pour un problème multiobjectif est considéré, visant à optimiser la consommation d'énergie totale, le retard total pondéré des jobs et la durée totale de préparation des machines. Ils notent que l'industrie d'injection de pièces plastiques, pour laquelle l'étude a été menée, est l'une des plus consommatrices en énergie. Chaque ordre de fabrication peut être produit sur un ensemble fini de machines. La durée et l'énergie consommée dépendent de la machine choisie, tandis que les temps de préparation dépendent des séquences des jobs sur les machines. Le critère d'optimisation est agrégé via une somme de déviations par rapport à des fonctions objectifs cibles. Une résolution à base d'un système multiagent est ensuite abordée, où chaque agent correspond à une machine. Chaque sous-problème est constitué d'un ensemble de jobs en fonction de la machine/agent qui leur est alloué. Chaque agent applique alors le modèle mathématique et retourne au nœud maître le résultat trouvé, qui recompile les données en un ordonnancement final. L'approche est validée sur un ensemble d'instances générées.

(Grimes et al., 2014) s'intéressent à des ordonnancements adaptatifs en fonction de la volatilité des tarifs énergétiques. Ils notent que la plupart des consommateurs paient l'électricité le même tarif tout au long de la journée, ou en mode heures pleines/heures creuses ou encore avec des tarifs qui dépendent des tranches horaires sur la journée ou la saison. Cependant ces tarifs sont loin de la réalité du marché car ils sont précalculés et absorbés par le fournisseur d'électricité. Comme les prix varient beaucoup au cours d'une journée, il est dans l'intérêt du consommateur et du fournisseur d'utiliser les prix en temps réel du marché. Ces derniers offrent un avantage important, particulièrement pour les gros consommateurs qui peuvent exploiter la volatilité des prix, mais les risques d'une telle approche sont évidemment en conséquence. Le but de tels ordonnancements est de placer les tâches gourmandes en énergie à des moments où le coût de l'électricité est le plus faible. Dans cette étude, l'objectif est de construire un planning pour le jour suivant en se basant sur une tarification supposée. Le planning sera ensuite évalué sur la véritable valeur appliquée le jour suivant, car les prix fluctuent énormément d'un jour à l'autre et les prévisions sont délicates à effectuer. Deux approches sont étudiées : une approche industrielle et une approche à destination des particuliers. Le modèle mathématique concernant les particuliers est relativement novateur car il inclut la possibilité d'avoir des véhicules électriques reliés au réseau de la maison, ce qui présente deux avantage : être rechargé en heure creuse, et participer à l'alimentation du domicile lors des pics de consommation.

2.3. Les problèmes de Flow-shop

(Fang et al., 2011) exposent le fait que la consommation énergétique des machines est relativement faible en comparaison de la consommation totale de l'entreprise. Leur modèle inclut des contraintes telles que la minimisation de l'empreinte carbone, des pics de consommation énergétique ainsi que la minimisation de la durée totale de traitement. La particularité de l'environnement de production modélisé est d'intégrer des machines à vitesses variables. Les auteurs étudient l'optimisation d'un problème contenant 36 pièces dans un Flow-shop à 2 machines et 5 vitesses variables. L'outil du marché qu'ils utilisent pour la résolution n'a pas permis d'obtenir des résultats viables en une journée de calcul. Cependant, en considérant des relaxations sur certaines contraintes, il est possible de résoudre le problème. Ils soulignent le fait que dans un milieu industriel, avec de nombreux ordres de fabrication, il est judicieux de construire une solution répondant à une contrainte

(par exemple le makespan) et de modifier cette solution par la suite pour prendre en compte les autres contraintes. Les approches hiérarchisées sont donc des alternatives envisageables et même préconisées pour optimiser des problèmes de taille importante. L'utilisation de méthodes de résolution approchées est également préconisée (exploration par voisinages) dans un tel contexte industriel.

(Zhang et al., 2014) proposent un programme linéaire indexé sur le temps afin de minimiser les dépenses en électricité et l'empreinte carbone dans le cas d'une tarification différente selon les horaires. Le but est de ne pas compromettre la productivité de l'atelier. Ils s'intéressent plus particulièrement à un Flow-shop à huit étages. Comme noté par les auteurs, le coût de la production lié à la consommation en électricité peut être réduit en déplaçant les opérations vers les heures creuses (off) ou moyenne (middle), dans le cadre d'une tarification à trois niveaux. Cependant, dans les régions où l'électricité est produite par combustion de gaz, ou par utilisation de charbon, le fait de produire pendant les heures creuses/middle peut avoir un impact sur les émissions en CO_2. Ceci complique évidemment la problématique visant à fournir des ordonnancements énergétiquement et environnementalement responsables.

(Lin et al., 2015) étudient un problème de Flow-shop bloquant, c'est-à-dire qu'un job reste bloqué sur une machine tant que la machine suivante n'est pas disponible. Les objectifs consistent à définir les paramètres des processus d'usinage et à résoudre le problème d'ordonnancement. Les objectifs sont alors la minimisation du makespan et de l'empreinte carbone. Trois stratégies sont utilisées : des décalages à droite des opérations sans altérer la date de fin de la dernière opération d'un job ; l'extinction/allumage des machines, et l'optimisation préliminaire des paramètres d'usinage des machines. Ce sont principalement les décalages et la définition en amont des paramètres d'usinage qui conduisent aux meilleurs résultats. Les auteurs utilisent un algorithme évolutionnaire afin de résoudre le problème étudié.

(Lu et al., 2017) proposent une modélisation mathématique d'un problème de Flow-shop avec temps de préparation des machines dépendant des séquences, temps de transport contrôlable, et énergie. Ils ajoutent la possibilité d'allumer/éteindre les machines en incluant la durée de vie des matériaux afin d'éviter des redémarrages trop nombreux. Les temps de transports ne sont généralement pas étudiés dans l'ordonnancement avec contraintes énergétiques. Les transports sont gérés par des convoyeurs dont la vitesse est variable, permettant à un job de se déplacer plus vite d'une machine à une autre. L'effort mis pour déplacer une pièce dépend de son poids ce qui se traduit par une dépense énergétique additionnelle. Les auteurs proposent un algorithme multiobjectif évolutionnaire pour résoudre le problème visant à minimiser le makespan et la consommation énergétique.

2.4. Les problèmes de Flow-shop Flexible/Hybride

(Nolde and Morari, 2010) s'intéressent à la planification dans une aciérie, le problème est proche d'un Flow-shop Flexible. La problématique étudiée dans leur travail est de trouver un ordonnancement qui réponde au mieux à une courbe de charge qui est définie conjointement entre le client (manufacture) et le fournisseur d'électricité. Cette courbe est acceptée au moins un jour avant l'exécution de l'ordonnancement. L'avantage pour le fournisseur est double : premièrement, cela permet de limiter la consommation durant certaines périodes durant lesquelles l'énergie est chère à produire ; deuxièmement, en connaissant à l'avance les besoins d'un industriel (surtout ceux présentant des processus intensifs) il peut équilibrer différemment la puissance injectée, ce qui permet une meilleure gestion de la grille d'énergie. Cependant, même si cela peut permettre à l'industriel des économies importantes, ce dernier doit respecter précisément la courbe acceptée et toute

consommation en excès ou en dessous de la courbe est sujette à des pénalités financières. L'objectif pour l'industriel est que sa consommation moyenne, par tranche de 15 min, respecte la courbe. Les auteurs conçoivent un programme linéaire pour leur problème. Leur objectif est la minimisation de la somme des erreurs sur les intervalles considérés. Ils utilisent également une heuristique de construction qui vise à fournir une bonne solution de départ au solveur. Chaque solution est soumise à l'acceptation du fournisseur d'électricité. S'il accepte alors le processus s'arrête, sinon une nouvelle solution est cherchée. Les auteurs proposent comme perspective la prise en compte de la robustesse des solutions face à des pannes ou des délais qui pourraient avoir un fort impact sur l'ordonnancement et par conséquent une incidence sur la consommation négociée avec le fournisseur. Une alternative dans la modélisation mathématique a été proposée par (Haït and Artigues, 2011), en réduisant presque par cinq le nombre de variables binaires, ce qui a permis aux auteurs d'atteindre l'optimalité sur le cas d'étude initial en réduisant drastiquement les temps de calcul.

(Bruzzone et al., 2012) présentent une méthode de résolution visant, à partir d'un plan de production proposé par un APS (Advanced Planning and Scheduling), à minimiser les pics de consommation en puissance sur une chaîne de production modélisée par un Flow-shop Flexible (FFS). L'optimisation se fait via un module EAS (Energy Aware Scheduling). Ici, l'APS va minimiser le makespan et le retard total ; l'EAS, qui ne modifie en rien la séquence retournée pas l'APS, tente alors d'optimiser l'ordonnancement proposé du point de vue de la contrainte énergétique, en modifiant les dates de début des opérations. Il est donc probable que le makespan et le retard total soient impactés par l'EAS. Deux approches sont proposées : une résolution à l'aide d'un programme linéaire et une approche par exploration de voisinage : Randomized Neighbourhood Search (RANS). Il s'avère que le programme linéaire n'est applicable que si des pics de puissance élevés sont autorisés, et qu'il devient vite impossible de trouver une solution au fur et à mesure que le seuil de puissance devient plus faible. Le modèle ne prend pas en compte la variabilité de la puissance requise par les machines qui est ici constante. Cependant un des avantages non négligeables de leur approche est la possibilité de coupler un EAS avec les outils d'APS existants sur le marché sans avoir besoin de se procurer un outil qui effectuerait les deux actions en parallèles (EAS+APS). Si les approches intégrées sont connues pour retourner de meilleures solutions, les investissements financiers requis sont souvent à l'origine du rejet par les industriels des solutions logicielles proposées afin de réduire la consommation énergétique des systèmes de production (O'Rielly and Jeswiet, 2014). Le fait de ne proposer qu'un module supplémentaire peut donc avoir un intérêt dans la mise en place de systèmes de production efficients.

(Dai et al., 2013) s'intéressent à un problème de Flow-shop Flexible. Ils proposent un Algorithme Génétique (GA) associé à un Recuit Simulé (AS). Ils prennent en compte dans leur modèle les différentes puissances requises par les machines en fonction de leur état afin de calculer l'énergie dépensée. Ils proposent d'éteindre ou d'allumer des machines en fonction des besoins de la chaîne de production et en respectant certaines conditions. Par exemple, lorsque l'énergie utilisée dans une phase de veille dépasse celle nécessaire à l'extinction et au rallumage de la machine durant cette même période, alors la machine peut être éteinte. Ils incluent également la durée de vie des composants afin de ne pas éteindre une machine trop souvent ; leur résultat est représenté par un front de Pareto.

(Luo et al., 2013) prennent en compte la variabilité du coût de l'électricité en fonction de différentes périodes en journée et intègrent le fait qu'il est fréquent de trouver des machines plus ou moins anciennes et récentes à une même étape de production (volonté de l'entreprise de maintenir d'anciens actifs) et qu'il faut donc répartir la charge entre ces machines en fonction de leur performance et de leur consommation énergétique. Leur travail se dirige vers les systèmes de type Flow-shop Hybrides (HFS). Les objectifs pris en compte sont la minimisation du makespan et la minimisation du coût total de la production. Une approche par métaheuristique est employée avec

l'utilisation d'un système d'optimisation multiobjectif par colonie de fourmis (MOACO), pour arriver à des résultats prenant en compte les deux objectifs. Aucune recherche locale n'est appliquée pour modifier l'ordre de traitement des opérations contrairement à de nombreux travaux, mais des modifications sont effectuées sur les dates de début (Right-Shift) de sorte à réduire le coût associé à chaque ordonnancement. Les résultats obtenus par ce type d'approche sont de bonne qualité et sont comparés avec d'autres algorithmes multiobjectifs. Une conclusion intéressante et générale de leurs travaux est qu'il vaut mieux coupler une machine rapide et énergétiquement coûteuse avec une machine lente et économique plutôt que de mettre deux machines moyennes.

(Sharma et al., 2015) proposent de coupler les aspects économiques et écologiques dans le cadre d'un système de production de type Flow-shop Hybride à deux étages avec assemblage. La vitesse des machines peut varier afin d'adapter leur consommation énergétique aux besoins de la production. Des contraintes de tarification par tranche horaire sont également considérées. Les auteurs visent à minimiser l'impact financier et écologique, en réduisant les émissions de carbone, d'un atelier de fabrication tout en préservant un niveau de production cible (makespan). Comme il n'est pas possible de réduire le coût de la production en profitant de tarifs favorables sans émettre plus de dioxyde de carbone, une métaheuristique multiobjectif est proposée en considérant une fonction objectif agrégée. Les auteurs étudient alors trois approches : l'approche économique, écologique et « éconologique » (combinant les deux approches précédentes). D'après les résultats obtenus, l'approche éconologique propose globalement un meilleur comportement que les approches écologique/économique.

2.5. Les problèmes de Job-shop

(May et al., 2015b) considèrent un Job-shop avec consommation énergétique. Trois indicateurs de performance sont proposés : le makespan, la consommation énergétique totale, et la consommation énergétique perdue. Leur algorithme évolutionnaire multiobjectif développé pour le problème est basé sur NSGA-II et SPEA-II (Zitzler et al., 2001). Quatre règles de gestion sont étudiées. La première concerne le makespan uniquement en considérant que toutes les machines sont allumées tout le temps, la deuxième permet d'allumer les machines en début et de les éteindre lorsqu'elles ont fini de traiter les tâches qui leurs sont dévolues, la troisième permet d'éteindre les machines lorsqu'elles sont inactives, et la quatrième ajoute un mode "stand-by" qui dépend de la consommation énergétique. Dans ce dernier mode, il est possible d'éteindre la machine, de la laisser inactive ou de passer en mode standby en fonction des avantages qu'apporte chaque état. Les auteurs démontrent la validité de leur algorithme via une application reposant sur trois instances de (Fisher and Thompson, 1963) en appliquant les différentes règles de gestion.

(Salido et al., 2016b) étudient un problème de Job-shop où les machines ont des consommations énergétiques qui dépendent des vitesses auxquelles elles fonctionnent. La vitesse de base est celle des instances sur lesquelles reposent leur étude. L'énergie est calculée de manière aléatoire en fonction des durées des opérations. Ils optimisent une combinaison linéaire des deux objectifs : la consommation énergétique totale et le makespan. Deux approches sont employées. La première repose sur l'utilisation de CP optimizer qui intègre un système de recherche par voisinage approprié aux problèmes d'ordonnancement, la seconde sur un algorithme génétique. Les auteurs remarquent que cette dernière approche est plus performante lorsque l'objectif se concentre principalement sur la minimisation de la durée de traitement de l'ensemble des opérations. Par la suite, (Salido et al., 2016a) s'intéressent à une autre problématique liée à l'ordonnancement sous contrainte énergétique en prenant en compte la robustesse des ordonnancements. Les auteurs montrent qu'un système moins

énergivore est également plus robuste. En effet, un ordonnancement énergétiquement efficace suggère que les machines tournent à vitesse réduite et consomment donc moins. Lorsqu'une panne apparaît, il est possible de reprendre la production en augmentant les vitesses pour rattraper le temps perdu à cause de la panne.

(Liu et al., 2014) s'intéressent à un Job-shop où sont minimisés le retard total et la consommation énergétique. Leurs travaux prennent place dans un contexte industriel où les machines ne peuvent être éteintes. Ils notent qu'optimiser la consommation énergétique revient au même que de minimiser les durées d'inactivité des machines, puisqu'il n'est pas possible d'intervenir sur la consommation des machines lors des phases d'usinage. Ils développent un algorithme issu de la classe des algorithmes génétiques multiobjectifs. Une étude a été faite en se basant sur l'instance de taille 100 de Fisher et Thompson, modifiée pour intégrer des dates d'échéances et des profils de consommation pour chaque opération, afin d'évaluer la métaheuristique proposée. Leurs résultats sont concluants et montrent la capacité de l'algorithme à fournir des solutions de bonne qualité. (Liu et al., 2015) traitent d'un problème où la quantité d'énergie disponible à certains moments est réduite à cause de la réglementation gouvernementale consistant à réduire la production d'électricité à certaines périodes. Pour certaines compagnies, un tiers de la capacité de production peut être perdue à cause de ces réductions ce qui les pousse à utiliser illégalement des générateurs qui polluent davantage pour répondre aux demandes des clients. Dans ce contexte, les auteurs cherchent à optimiser le retard total pondéré, ainsi que la consommation en électricité et les coûts liés à l'électricité dans le cadre d'un problème de Job-shop. Un modèle mathématique est proposé et un algorithme de type NSGA-II est utilisé pour résoudre le problème d'optimisation multiobjectif. Ils proposent une heuristique visant à modifier les ordonnancements déjà calculés en fonction des restrictions. Cette heuristique applique des décalages dans les dates de début des opérations initialement ordonnancées dans les périodes de chute d'énergie. La procédure applique ensuite un décalage vers la gauche des opérations qui peuvent être réinsérées plus tôt en fonction de leur priorité liée aux dates d'échéances.

(Zhang and Chiong, 2016) proposent de résoudre un problème visant à minimiser la consommation énergétique et le retard total pondéré sur un système de production de type Job-shop. Les machines du problème ont des vitesses variables. Un algorithme génétique multiobjectif, intégrant deux recherches locales en fonction de l'objectif optimisé, est proposé. Pour appliquer ces recherches locales, les solutions de la population sont réparties en quatre zones en fonction de leur qualité. Pour les solutions présentant une bonne qualité du critère énergétique, la recherche locale temporelle est appliquée ; pour les solutions présentant une bonne qualité temporelle, la recherche locale énergétique est appliquée. L'espace étant divisé en quatre, les solutions de la zone correspondant aux individus avec une bonne qualité sur les deux critères se voient appliquer les deux recherches locales, tandis que la quatrième zone est réservée aux « mauvaises » solutions qui ne sont pas améliorées. Ce principe de répartition permet de réduire les temps de calcul alloués aux phases de recherches locales, et les résultats sont de meilleure qualité qu'avec une métaheuristique sans recherche locale.

(Tang and Dai, 2015) étudient un problème de Job-shop où les vitesses des machines peuvent varier afin de réduire la consommation totale en énergie du système de production. L'objectif principal est l'optimisation de cette consommation tout en préservant l'ordre de traitement des opérations établi par un outil de planification en amont. Une formalisation mathématique ainsi qu'un algorithme génétique sont proposés afin de résoudre le problème.

2.6. Les problèmes de Job-shop Flexible

(Zhang et al., 2016) s'intéressent au réordonnancement dynamique dans le cadre d'un atelier flexible (assimilable à un Job-shop Flexible) car ce type d'approche est plus proche des réalités du milieu industriel. Les jobs arrivent de manière aléatoire et des pannes sont également considérées de manière simultanée sachant que chaque opération peut être effectuée sur des machines différentes. Les critères de productivité optimisés sont le makespan et le retard, ces deux critères étant agrégés au sein d'une fonction objectif combinant les deux en donnant plus de poids au makespan. La fonction objectif liée à l'énergie consiste à minimiser l'énergie perdue au sein du système. Ils utilisent un algorithme génétique pour la résolution, ainsi qu'une modélisation sous forme de programme linéaire.

(Pach et al., 2014) s'intéressent à une approche faisant intervenir une planification réactive dans un environnement flexible de production, tout en prenant en compte la consommation totale de la production. Ils notent que la résolution exacte sur des problèmes de type Job-shop ou Flow-shop étant déjà très longue, rajouter une contrainte supplémentaire ou un autre objectif rend la résolution plus difficile encore. Ceci rend ce type de résolution incompatible avec un contexte dynamique. Au-delà des métaheuristiques couramment utilisées, les auteurs s'intéressent à des approches alternatives. Plus précisément, ils mettent en œuvre un champ de potentiel, qui peut être vu comme une heuristique en temps réel. Cette méthode vise à influencer le comportement des entités dans un environnement incertain. Par exemple, si une entité peut être traitée par deux ressources, elle se dirigera plus probablement vers la ressource ayant le potentiel attracteur le plus élevé. Les auteurs ont montré l'efficacité d'une telle approche à la fois en terme de facilité d'implémentation et de qualité des décisions prises pour les problèmes de routage et d'affectation. Dans leur étude les produits ont également leur propre champ de potentiel afin d'apporter un niveau de décision supplémentaire. Ainsi, lorsqu'un produit s'approche d'une ressource sur laquelle il s'apprête à subir un traitement, cette ressource est mise en marche si elle était préalablement éteinte. La consommation de chaque ressource ainsi que de l'ensemble du système est prise en compte ; les auteurs monitorent également le nombre d'allumage/extinction des machines car ce procédé peut endommager les microcontrôleurs. Un modèle de simulation est également développé pour évaluer les capacités de l'approche à traiter des problèmes de taille plus conséquente. L'approche proposée peut conduire à une économie de 12% au niveau de la consommation d'énergie totale, pour une augmentation du makespan de moins de 1% seulement.

(Garcia-Santiago et al., 2015) abordent un Job-shop Flexible avec contrainte énergétique dont la résolution est gérée par une métaheuristique. La priorité donnée à leur modèle est la minimisation de la consommation énergétique tout en respectant les échéances prévisionnelles. Ils notent que si le bon sens suggère de passer les machines dans des modes de consommation réduite, de telles pratiques dépendent beaucoup du système de production et de la possibilité de les mettre en œuvre car changer l'état d'une machine peut avoir des impacts sur d'autres objectifs (capacité de production, inventaire prévisionnel, priorité de fabrications, mais aussi sur la disponibilité des opérateurs, des phases de maintenance, des pannes potentielles etc.). De plus, les responsables de production ne sont généralement pas formés au potentiel d'amélioration de la consommation énergétique du système de production et laissent les machines en phase d'attente afin qu'elles puissent traiter n'importe quelle opération à n'importe quel moment. Afin de réduire la consommation énergétique liée à la production, les auteurs maximisent la durée passée en mode économique des machines.

(Lei et al., 2016) proposent une métaheuristique dans le but de réduire la consommation énergétique totale dans un système de production modélisé par un Job-shop Flexible. Leur deuxième objectif est d'équilibrer la charge sur les machines. Les auteurs ajoutent un degré supplémentaire de

difficulté au problème en prenant également en compte des vitesses variables d'usinage pour les machines.

(He et al., 2015) s'intéressent à un Job-shop Flexible également où la consommation globale du système doit être minimisée en allouant les machines les plus adaptées et en réduisant les phases d'inactivité. Un modèle linéaire est proposé ainsi qu'une métaheuristique. Deux scenarii sont appliqués. Dans le premier, la consommation énergétique totale est minimisée. Dans le deuxième, l'énergie est minimisée en même temps que le makespan.

(Stock and Seliger, 2015) notent que les énergies renouvelables produites sont intermittentes et qu'il n'est pas possible d'assurer l'alimentation continue d'une usine par ces moyens. Sans utiliser de systèmes de stockage (batteries) il est alors important de pouvoir adapter rapidement les besoins énergétiques d'une usine pour répondre efficacement aux fluctuations énergétiques. Les auteurs traitent d'un problème de Job-shop Flexible en agrégeant deux critères : la consommation énergétique totale et le makespan. Une métaheuristique évolutionnaire est proposée.

(Li and Cao, 2015) proposent d'optimiser les émissions en dioxyde de carbone au sein d'un atelier de type Job-shop Flexible. Ils considèrent que les émissions varient d'une machine à une autre. Les émissions produites durant les phases d'inactivité sont également prises en compte. Une approche en deux étapes est proposée pour la résolution du problème. Chaque étape est basée sur l'optimisation d'une fonction objectif en utilisant un algorithme génétique. La première vise à réduire les émissions liées aux phases d'usinage, en proposant une affectation et un ordonnancement adapté des opérations. La deuxième étape vise à optimiser les émissions en réduisant les phases d'inactivité des machines.

(Moon and Park, 2014) traitent de deux problèmes de Job-shop Flexible. Dans le premier l'objectif est la minimisation conjointe du coût lié à la production et à la consommation en électricité. Dans le second, les auteurs intègrent également des sources d'électricité provenant d'énergies renouvelables (solaire) et de batteries qui sont chargées aux moments les plus opportuns et déchargées pour soutenir la production durant les périodes onéreuses.

(Liu and Tiwari, 2015) s'intéressent à un problème de Job-shop Flexible avec lot-sizing. Ils proposent une approche biobjectif dans le but d'optimiser le makespan et la consommation énergétique totale. Cette approche est basée sur un algorithme de type NSGA-II qui fonctionne en deux temps. Premièrement, la taille des lots est définie en appliquant des opérateurs issus des algorithmes génétiques dédiés au problème de lot-sizing pour définir une première population ; deuxièmement, cette population est transformée à l'aide d'opérateurs dédiés au problème d'ordonnancement. L'algorithme est réitéré jusqu'à atteindre une population acceptable.

Dans (May et al., 2015a) les auteurs développent plus en détail la notion d'indicateurs de performance pour l'optimisation énergétique (e-KPI). Ces indicateurs sont construits sur la base de questionnaires aux entreprises. Plus précisément, leurs e-KPI font le rapprochement entre les états sur la ligne de production (observations temporelles) et les états énergétiques au travers de relations cause/effet. Ces e-KPI sont obtenus après avoir suivi un certain nombre d'étapes dont : la définition du système de référence, l'identification des puissances requises par ressource, l'analyse des causes responsables de l'inefficacité énergétique des ressources, etc... Ils notent que les événements pouvant affecter le comportement énergétique d'une machine-outil sont de cinq ordres : le management de la production, la qualité, les ressources humaines, la maintenance, et d'éventuelles causes externes.

(H. Zhang et al., 2015) s'intéressent à la consommation électrique d'un ensemble d'usines. Une approche reposant sur une tarification en temps réel est utilisée car est la plus adaptée à un environnement de ce type, en permettant de mieux représenter la dynamique du système dans son

ensemble. Dans leur problème chaque usine consiste en un ensemble de Flow-shops et le but est d'optimiser le coût de la consommation en électricité en prenant en compte la quantité de produits fabriqués pour chaque usine. Deux types d'approches sont étudiées : une approche collaborative (les usines partagent entre elles l'ordonnancement appliqué) et sans-collaboration (l'ordonnancement est fait indépendamment des autres). S'ils notent l'aspect utopique d'une telle approche à cause des entreprises, et des règles limitant les comportements anti-concurrentiels, une réduction de presque 10% est observée lorsque les entreprises échangent des informations. Une amélioration notable est déjà observée lorsque les entreprises considèrent qu'elles sont dans un environnement où d'autres usines similaires du point de vue de la demande en énergie sont aussi présentes (8 %). Prendre en compte l'environnement dans lequel évolue une entreprise semble donc présenter un intérêt financier non négligeable.

Le nombre d'articles ayant pour objectif de réduire l'impact environnemental et la consommation énergétique des systèmes de production a subi une forte progression depuis 2010. Il n'est donc pas possible d'être exhaustif. Afin de faciliter la lecture de cette revue de la littérature, le Tableau 1 proposé au début de cette section permet de récapituler les approches traitées par les articles qui ont servi à cet état de l'art.

De manière générale, (Gutowski et al., 2006) ont montré que l'énergie requise pour l'usinage ne correspond qu'à 15% de l'énergie totale consommée, les 85% restants correspondent au maintien de la machine dans un état prêt à produire. Dans le cadre d'une machine à commande numérique, ces taux passent à 65.8% d'énergie utilisée durant l'usinage, et 34.2% pour maintenir la machine en état stationnaire. L'optimisation des processus à l'échelle d'un site de production est donc d'une importance primordiale. Comme noté par (Biel and Glock, 2016), la plupart des articles se concentrent principalement sur l'optimisation de la consommation énergétique totale et ils ne traitent pas des besoins en puissance instantanée des systèmes de production alors que ce point constitue un poste de dépense important pour une entreprise car les puissances requises par les machines montrent qu'il est particulièrement intéressant de lisser les pics de puissance qui peuvent être engendrés par l'usinage simultané de différents produits. Les auteurs ont également observé que seulement treize articles du panel exposé traitent de problématiques impliquant des Job-shop alors que ces derniers représentent fidèlement de nombreux systèmes de production. Il paraît donc important de proposer des méthodes de résolution pour les problèmes de Job-shop tout en prenant en compte les besoins en puissance du système de production. A cette fin, deux problèmes complémentaires sont étudiés dans ce chapitre. Le premier est un problème mono-objectif qui consiste à minimiser le makespan de l'ordonnancement lorsqu'un seuil de puissance variable est considéré ; le deuxième traite d'un problème biobjectif visant à proposer un ensemble de solutions ayant chacune une puissance maximale autorisée.

3. Job-shop avec seuil de puissance variable (Kemmoe et al., 2017)

Dans cette section il est proposé d'aborder l'impact de la puissance consommée (PC) sur les ordonnancements dans le cadre d'un atelier de fabrication de type Job-shop. Une formalisation mathématique basée sur les travaux proposés par (Kemmoe et al., 2015b; Kemmoé-Tchomté et al., 2015a) et modélisant le problème est introduite ; son objectif est de minimiser le makespan en prenant en compte le profil de puissance requis par les opérations et un seuil de puissance variable à ne pas dépasser. La considération d'un seuil de puissance est liée à la problématique ECOTHER qui vise à améliorer la production soumise à ce type de contrainte. Le fait d'intégrer un seuil de puissance variable à respecter permet également de prendre en compte un environnement où la puissance peut fluctuer

dans la réalisation des ordonnancements (sources d'énergies renouvelables). Trois métaheuristiques sont également proposées pour la résolution de ce problème.

3.1. Présentation du problème

L'objectif de cette étude est de prendre en compte les profils de puissance des opérations lors de la résolution des problèmes d'ordonnancement des systèmes de production lorsque ces derniers sont soumis à un seuil de puissance variable. Après analyse de différents profils de puissance présents dans la littérature, il a été choisi de considérer que les opérations ont des profils de puissances comme montré sur la Figure 2.

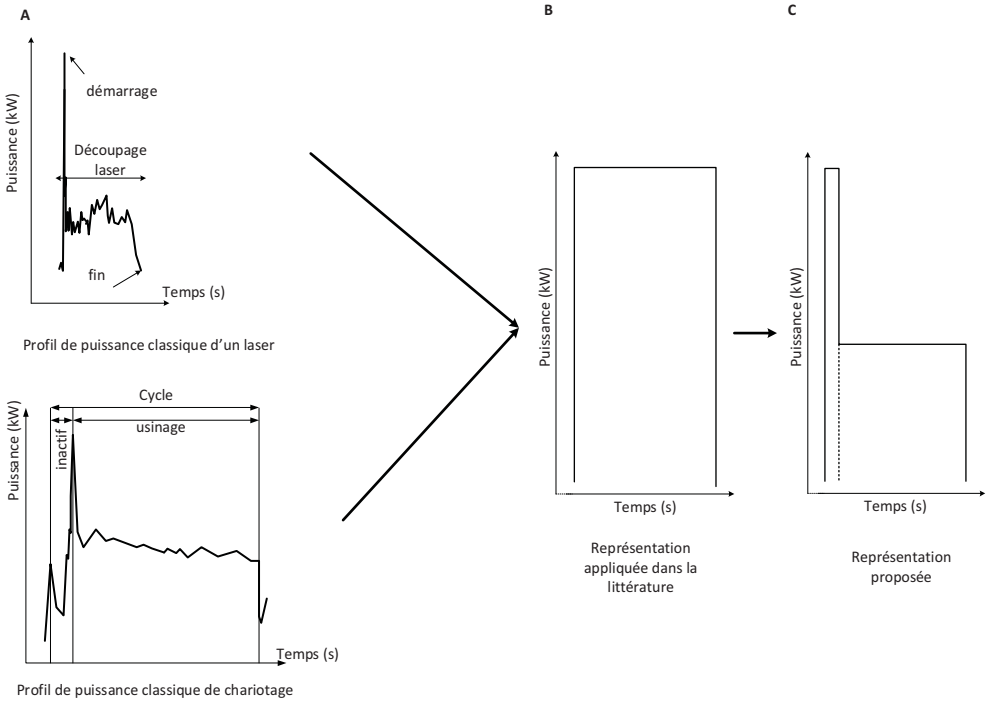

Figure 2 Représentation graphique d'un profil de puissance pour l'usinage (Kemmoe et al., 2017)

Dans la Figure 2, il est facile de comprendre que considérer une puissance constante n'est pas suffisant pour modéliser les profils de puissance issus du monde réel pour les opérations d'usinage : cela entraîne une perte d'informations précieuses. Par exemple, dans la Figure 2(A), le profil de puissance d'une machine de découpe laser pourrait être simplifié comme étant une opération nécessitant 40kW tout au long du processus (Figure 2(B)). Si un seuil de puissance est considéré comme fixé à 40kW, cela implique qu'une autre opération consommant 15kW ne sera pas planifiable avant la fin de l'opération précédente. Cependant, si l'on considère que l'opération a une puissance maximale en début d'usinage et une puissance nominale durant le processus de découpe, il est possible de programmer la deuxième opération (15kW) plus tôt, et ce juste après la fin du pic de puissance observé en début d'usinage. Pour intégrer cette information dans cette étude, il a été décidé de diviser les opérations en fonction de leurs besoins en puissance comme dans la Figure 2(C). Ce découpage repose sur la notion de blocs énergétique proposée par (Weinert et al., 2011). La Figure 3 présente un tel découpage.

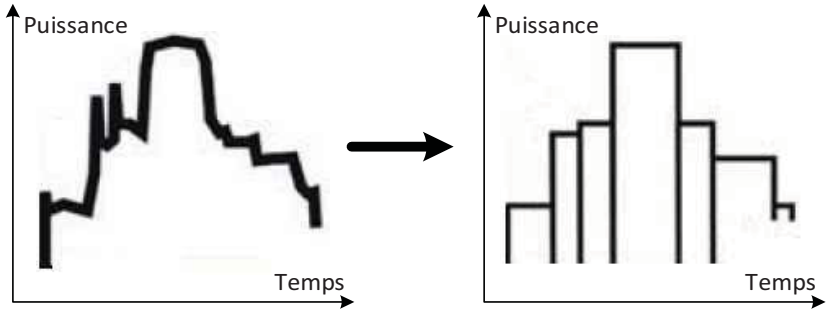

Figure 3 Découpage d'un profil de puissance en blocs énergétiques

On peut constater que le profil de puissance peut être divisé en de nombreux blocs. Cependant, comme noté par (Merkert et al., 2015), la finesse de modélisation conduit à une augmentation de la difficulté des problèmes. Il faut donc trouver un compromis entre une modélisation intégrant suffisamment d'information sans pénaliser la résolution du problème. Sans perte de généralité, si le choix de ne diviser les opérations qu'en deux blocs (sous-opérations) dans les instances traitées est fait, la formalisation mathématique proposée dans la suite peut gérer des profils de puissance nécessitant une décomposition plus fine. Afin de faciliter la compréhension, l'énoncé d'un problème où les opérations ont des puissances requises est proposé dans le Tableau 2.

Tableau 2 Instance exemple d'un problème avec puissance pour les opérations

J_1	$O_1 = (M_1 ; 8 ; 38) (M_1 ; 81 ; 17)$	$O_2 = (M_2 ; 10 ; 37) (M_2 ; 25 ; 21)$
J_2	$O_3 = (M_2 ; 17 ; 44) (M_2 ; 13 ; 20)$	$O_4 = (M_1 ; 18 ; 47) (M_1 ; 36 ; 15)$

Dans le Tableau 2, la première opération du job J_1, qui doit être exécutée sur la machine M_1, est constituée de deux sous-opérations. La première, qui correspond au pic de consommation, a une durée de 8 unités de temps (TU), et requiert une puissance de 38 unités de puissance (PU1) ; la seconde a une durée de 81 unités de temps et une puissance requise de 17 et elle représente la consommation nominale.

Un autre aspect de cette étude concerne le seuil de puissance autorisé, car ce dernier affecte profondément les ordonnancements proposés. Ce seuil peut être considéré comme constant dans le temps, mais aussi variable. C'est cette approche qui est privilégiée dans cette étude (voir la Figure 4), ce qui permet d'aborder de nombreux problèmes. Par exemple, sur la Figure 4, quatre opérations sont planifiées sur deux machines (M_1, M_2) et les lignes pointillées représentent la séparation entre les pics de puissance en début d'usinage et la puissance nominale qui suit. Dans cette figure, le seuil de puissance diminue, atteignant une valeur de 5 pendant une période où aucune opération n'a pu être planifiée. Par conséquent, il faut attendre une augmentation du seuil de puissance autorisé pour continuer le processus de fabrication. Ensuite, une panne de courant pour une maintenance préventive du côté du fournisseur se produit entre la deuxième opération du job J_1 et la deuxième opération du job J_2.

1 Power Unit PU

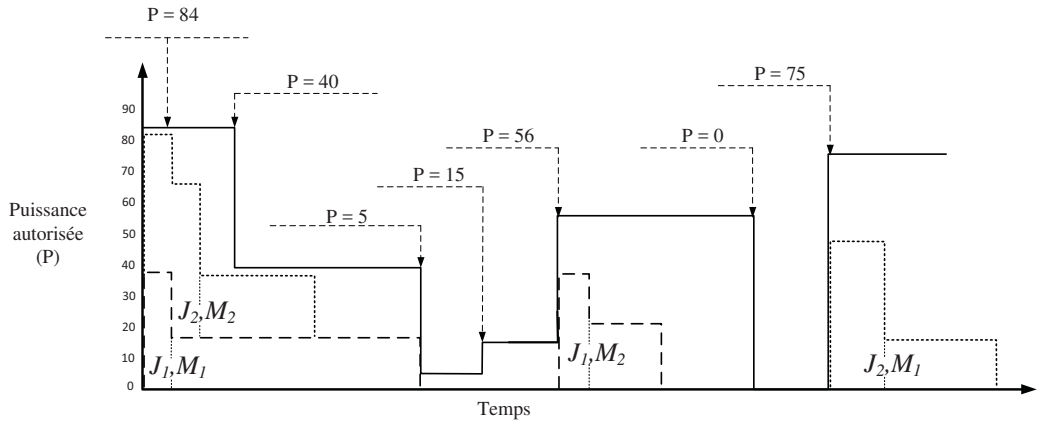

Figure 4 Ordonnancement d'opérations contraintes par un seuil de puissance variable

Le but de cette étude est de fournir un ordonnancement efficace pour un problème de Job-shop en répondant à la question suivante : « Quel est le meilleur ordonnancement, compte tenu du profil de puissance variable dont on dispose ? ». Étant donné que ce seuil variable est une contrainte donnée par le fournisseur en électricité, il ne peut pas être modifié. Par conséquent, lorsque de nouveaux ordres de fabrication arrivent, l'entreprise doit planifier sa production en fonction de la puissance disponible. Un modèle mathématique visant à définir complètement le problème est proposé dans la suite.

Beaucoup de systèmes de fabrication peuvent être modélisés en tant que Job-shop (Salido et al., 2016b). Dans le cadre d'un tel système de fabrication, un ensemble J de jobs doit être traité. Il a été vu dans le chapitre précédent que chaque job $j \in J$ est constitué d'un ensemble d'opérations. Pour exécuter ces opérations, le job va visiter un ensemble M de machines dans un ordre prédéfini. Dans cette étude le cas déterministe d'un Job-shop où le nombre de jobs à traiter est connu avant la planification est considéré. Aucune préemption n'est autorisée et aucun événement aléatoire n'est considéré. Ces éléments sont évoqués dans le chapitre II (§2) et un tel problème peut être modélisé par un graphe conjonctif-disjonctif. Une solution réalisable du problème correspond à un graphe acyclique qui prend en compte tous les arcs conjonctifs et qui ne contient qu'un arc disjonctif pour chaque paire d'opérations en disjonction (devant être traitées par la même machine). Une solution optimale correspond au graphe faisable présentant la plus petite valeur de makespan possible, obtenue après avoir fixé la date de début de chaque opération. A partir de l'exemple de la Figure 4, et en ne considérant que les précédences (les arcs conjonctifs et disjonctifs) entre les opérations, le graphe présenté sur la Figure 5 est construit.

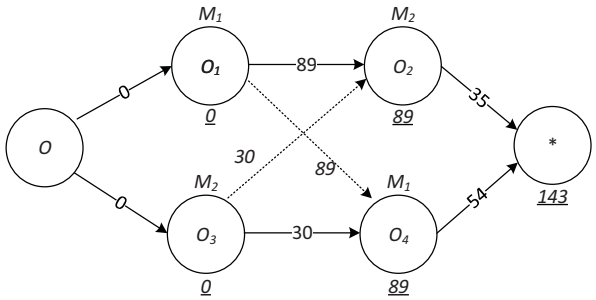

Figure 5 Représentation sous forme de graphe du problème sans considération de puissance

Dans la Figure 5, considérant les arcs disjonctifs issus du pseudo-diagramme de Gantt de la Figure 4, la date de début des opérations est obtenue avec un calcul de plus long chemin et ajoutée sous chaque opération. La Figure 5 ne présente cependant pas les contraintes énergétiques qui sont appliquées dans cette étude. De ce fait, la durée de l'opération O_1 est agrégée et correspond à la somme des durées des sous-opérations qui la constituent. Dans le problème, chaque opération est censée avoir un profil de puissance donné qui varie en fonction de l'avancement du processus. Ainsi, une opération peut être considérée comme la succession de sous-opérations correspondant à la puissance spécifique requise à différentes étapes du processus de production (Weinert et al., 2011). Deux types de puissances pour les opérations sont considérés à savoir : (i) la première concerne la puissance maximale qui apparaît au début de l'opération, (ii) la seconde correspond à la puissance nominale requise pendant la phase d'usinage. La distinction entre ces deux valeurs de puissances différentes s'effectue en divisant les opérations en sous-opérations comme mentionné dans le Tableau 2. Chaque opération O_i se compose de deux sous-opérations $O_{i,1}$ et $O_{i,2}$ ayant chacune pour durée $P_{i,1}$ et $P_{i,2}$ et les consommations $PC_{i,1}$ et $PC_{i,2}$ respectivement. Par conséquent, chaque opération modélisée par un sommet dans la Figure 5 est alors représentée par deux sommets comme le montre la Figure 6. Sur cette dernière, l'opération O_1 de la Figure 5 est divisée en deux sous-opérations ($O_{1,1}$ et $O_{1,2}$) en fonction de la puissance demandée pendant le processus. La puissance nécessaire à la réalisation de chaque sous-opération est donnée en exposant du numéro de la sous-opération dans chaque sommet ($O_{i,1}^{PC_{i,1}}$).

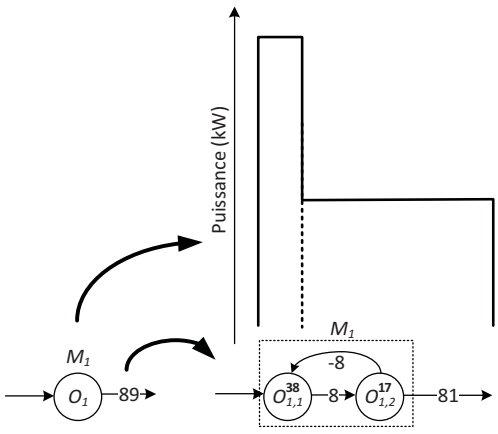

Figure 6 Passage d'une opération à deux sous-opérations

En fonction de la précision souhaitée, il est possible d'envisager différents profils de puissance en divisant les opérations en k sous-opérations pour que la modélisation soit aussi générale que possible, comme c'est d'ailleurs le cas dans la formulation mathématique. Puisque, une fois démarrées, les opérations ne peuvent pas être interrompues, des contraintes de délai nul sont envisagées entre ces sous-opérations en utilisant la notion de Time-lags max (Caumond et al., 2008). Les Time-lags max consistent en des retards de temps maximaux. Ils sont représentés dans un graphe à l'aide d'arcs pondérés négativement. Lorsque les contraintes de Time-lags ont pour objet la représentation d'un délai nul entre les sous-opérations, les arcs négatifs entre les sous-opérations dans le graphe ont des valeurs absolues égales aux temps de traitement. Dans la Figure 6, l'arc négatif entre $O_{1,2}$ et $O_{1,1}$ signifie que la date de début de l'opération $O_{1,2}$ est égale à la date de fin de l'opération $O_{1,1}$. Les deux arcs (négatifs et positifs) garantissent que les opérations sont exécutées sans délai. De cette façon, les contraintes spécifiques aux Time-lags sont toujours vérifiées et exprimées comme suit : $s_{i,2} - s_{i,1} \leq P_{i,1} + t_{i,1,2}^{max}, \forall i \in V$, où $t_{i,1,2}^{max}$ représente le délai maximal autorisé entre la date de fin de $O_{i,1}$ et la date de début de $O_{i,2}$ (Caumond et al., 2008). Compte tenu d'une opération spécifique O_i, si ce délai est nul, alors l'équation est identique à $s_{i,2} - s_{i,1} \leq P_{i,1}$, ce qui correspond à un arc négatif de valeur $-P_{i,1}$ allant de $O_{i,2}$ à $O_{i,1}$. Par conséquent, dans la Figure 6 l'arc entre $O_{1,2}$ et $O_{1,1}$ vaut -8. Pour respecter la contrainte de précédence entre $O_{i,1}$ et $O_{i,2}$, la contrainte $s_{i,2} - s_{i,1} \geq P_{i,1}$ est également considérée. Pour assurer la faisabilité de ces contraintes, nécessairement, $O_{i,2}$ commence directement après la fin de la sous-opération $O_{i,1}$. L'application de cette modélisation à chaque opération du graphe (Figure 5) entraîne le nouveau graphe présenté dans la Figure 7. La considération des contraintes classiques de Job-shop et des nouvelles contraintes de puissance implique la résolution d'un nouveau problème appelé problème de Job-shop avec puissances requises (JSPPR – Job-shop Problem with Power Requirements).

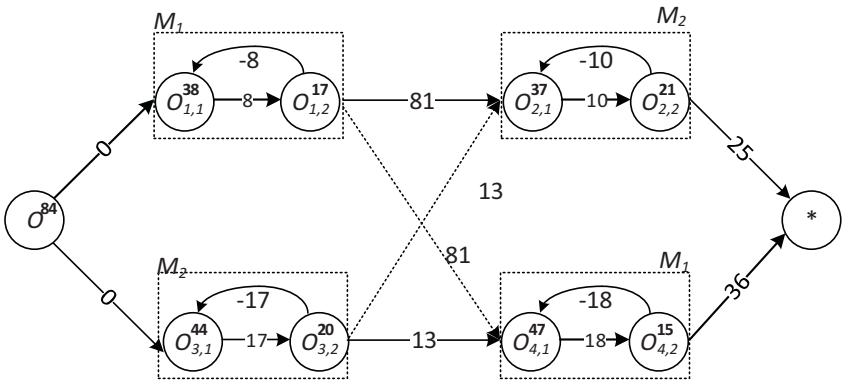

Figure 7 Représentation sous forme de graphe avec des sous-opérations

Le but de cette étude est de planifier l'exécution de toutes les opérations en considérant leur profil de puissance. Si l'on considère un seuil de puissance variable, il est possible d'obtenir la plus grande puissance autorisée pendant l'horizon de planification. Cette valeur la plus élevée sera considérée comme le seuil de puissance initial et est exprimée comme étant la puissance du sommet de départ O (valant 84PU d'après la Figure 4). Pour tenir compte des variations du seuil de puissance, des opérations fictives supplémentaires ayant elles-mêmes des besoins en puissance et des dates de départ sont considérées comme dans la Figure 8 (à savoir: opérations $O_{5,1}$, $O_{6,1}$... $O_{11,1}$). Ces opérations ne sont pas des opérations d'usinage et ne demandent aucune machine. Elles fonctionnent comme des puits temporaires. Les dates de début de ces opérations fictives sont connues avant

l'ordonnancement et font partie des données du problème ($O_{5,1}$ à partir de 0 - le début de l'ordonnancement, $O_{6,1}$ à partir de 29, etc.). Ensuite, l'objectif est de proposer un ordonnancement minimisant le makespan tout en respectant les dates de début et les exigences en puissance des opérations fictives.

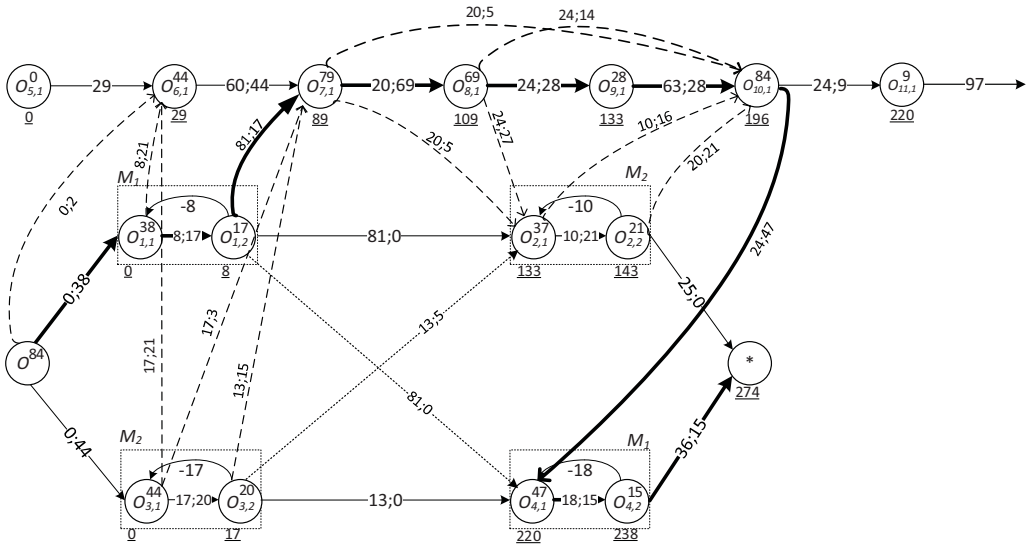

Figure 8 Graphe incluant un seuil de puissance variable

La puissance qui ne doit pas être dépassée dans le temps sera prise en compte en considérant des flots dans le graphe. Ces flots sont représentés avec de nouveaux arcs en tirets sur la Figure 8. Ces arcs supportent plus d'information qu'auparavant : ils sont désormais pondérés par la durée de l'opération et par la quantité représentant la puissance qui transite dans le système de production (i.e. « durée », « puissance transférée »). Cette puissance est considérée comme une ressource qui passe d'une opération à une autre. Concernant l'ensemble d'arcs conjonctifs ou disjonctifs déjà existants, la puissance qui va d'une opération à une autre est ajoutée sur l'arc plutôt que d'ajouter un nouvel arc. Pour modéliser les contraintes de puissance, la somme des arcs allant de la source O à d'autres opérations ne doit pas excéder le seuil autorisé. Par conséquent, la somme des puissances transitant du sommet fictif O aux autres est inférieure ou égale au seuil de puissance le plus élevé, valant donc 84PU. À la date 89, le seuil de puissance diminue à 5PU. Ceci est modélisé à l'aide de l'opération fictive $O_{7,1}$ qui consomme 79PU. De cette façon, seulement 5PU sont disponibles pour le traitement d'autres opérations. Ainsi, les opérations futures doivent attendre une augmentation du seuil de puissance pour pouvoir être exécutées.

Le graphe de la Figure 8 modélise le diagramme de Gantt de la Figure 4. Les dates de début sont obtenues avec un calcul de plus long chemin. Il est également vérifié lors de ce calcul que la puissance utilisée à chaque instant ne dépasse jamais le seuil de consommation autorisé. Par exemple, en raison des besoins en puissance, il n'est pas possible de démarrer l'opération $O_{2,1}$ en fin d'opération $O_{1,2}$, car $O_{7,1}$ nécessite une grande quantité de PU. Par conséquent, $O_{2,1}$ doit attendre jusqu'à ce que suffisamment de puissance entre dans le sommet correspondant via des arcs de flots (c'est-à-dire les arcs $O_{7,1} \rightarrow O_{2,1}$, $O_{8,1} \rightarrow O_{2,1}$ et $O_{3,2} \rightarrow O_{2,1}$). Parce qu'une opération ne peut recevoir de la puissance que d'opérations terminées, la date de début de $O_{2,1}$ est obtenue à partir de la date de fin maximale entre les opérations $O_{7,1}, O_{8,1}, O_{3,2}$ et $O_{1,2}$. Les arcs disjonctifs liés aux exigences en puissance auront

un rôle important dans la recherche locale définie dans la suite. Avant de présenter la méthode de résolution approchée employée, le modèle mathématique complet du problème est présenté dans la section suivante.

3.2. Formalisation mathématique

Comme indiqué précédemment, la puissance est attribuée aux opérations selon les principes de flots dans un graphe. Ainsi, le modèle est basé sur le problème théorique de Job-shop et sur une approche par flots. La formalisation mathématique du problème commence par les notations utilisées.

Notations

Paramètres :

H : un grand nombre positif ;

M : ensemble de machines ;

J : ensemble de jobs ;

V : ensemble de toutes les opérations ($|V| = |M|.|J|$) ;

T : ensemble d'événements pour le changement de seuil de puissance ;

V^+ : représente l'ensemble V adjoint avec des opérations fictives ($|V^+| = |V|+|T|$) ;

i, j : indices pour les différentes opérations ($i = O_i$) ;

J_i : job associé à l'opération i ;

SO_i : ensemble de sous-opérations de l'opération i ;

k, l : indexes représentant les différentes sous-opérations des opérations "globales" ;

μ_i : machine requise pour traiter l'opération i, $\mu_i \in M$;

$P_{i,k}$: durée de la $k^{ième}$ sous-opération de l'opération i ;

$PC_{i,k}$: puissance requise pour traiter la $k^{ième}$ sous-opération de l'opération i ;

T_k : temps de changement de seuil de puissance ($k = 0$ correspond au premier seuil) ;

PT^{max} : puissance maximale à ne pas dépasser (valeur la plus élevée du seuil de puissance) ;

Variables :

c_{max} : date d'achèvement de toutes les opérations, également appelée makespan ;

$s_{i,k}$: date de début de la $k^{ième}$ sous-opération de l'opération i ;

$x_{i,k,j,l}$: variable binaire égale à 0 si la $k^{ième}$ sous-opération de l'opération i n'est pas réalisée avant la $l^{ième}$ sous-opération de l'opération j et égale à 1 autrement ;

$y_{i,k,j,l}$: variable binaire égale à 0 si aucun flot de puissance ne passe de l'opération i à l'opération j et égale à 1 autrement ;

$\varphi_{i,k,j,l}$: nombre d'unités de puissance transférées entre les entités du système (pourraient être des opérations réelles ou fausses) ;

Formalisation mathématique :

La première ligne (1) du modèle mathématique concerne l'objectif du problème : minimiser le makespan.

$$Min\ c_{max} \tag{1}$$

Les contraintes (2) donnent l'expression du makespan qui doit être supérieur à la date de fin de la dernière sous-opération et ce pour chaque opération i.

$$s_{i,|SO_i|} - c_{max} \leq -P_{i,|SO_i|}, \forall i \in V \tag{2}$$

Les contraintes (3) représentent les contraintes disjonctives pour les sous-opérations traitées par les mêmes machines (à savoir : la dernière sous-opération d'une opération i est prévue avant la première sous-opération d'une autre opération j ou vice versa).

$$x_{i,|SO_i|,j,1} + x_{j,|SO_j|,i,1} = 1, \forall (i,j) \in V, J_i \neq J_j, \mu_i = \mu_j \tag{3}$$

Par exemple, si les opérations assignées à la machine M_2 de la Figure 9 sont considérées, la contrainte disjonctive consiste à considérer l'arc $O_{2,2} \rightarrow O_{4,1}$ ou l'arc $O_{4,2} \rightarrow O_{2,1}$ pour modéliser complètement la disjonction entre ces opérations sur la machine.

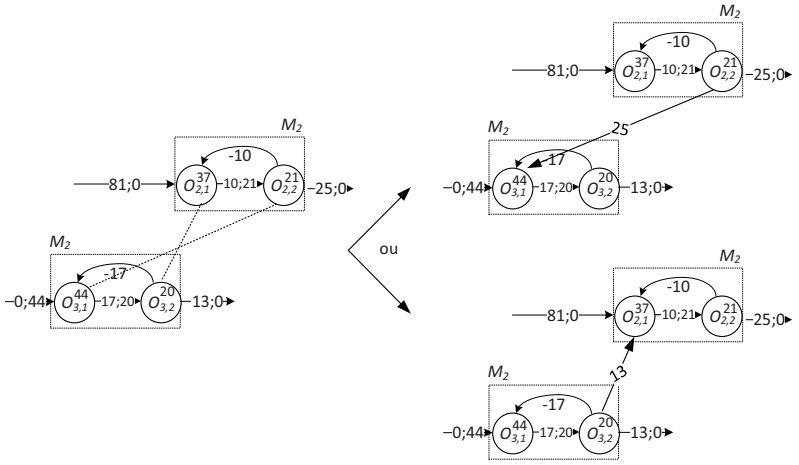

Figure 9 Précédences possibles entre deux sous-opérations (disjonctions)

Les contraintes (4) définissent les dates de début des sous-opérations en fonction de l'ordre de passage des opérations sur les machines pour chaque job. Par conséquent, la date de début de chaque sous-opération dépend des sous-opérations planifiées précédemment.

$$s_{j,1} - s_{i,|SO_i|} \geq P_{i,|SO_i|}, \forall (i,j) \in V, i < j, J_i = J_j \tag{4}$$

Les contraintes (5) assurent que les sous-opérations de l'opération i sont planifiées sans délai (la préemption n'est pas autorisée).

$$s_{i,k} - s_{i,k-1} = P_{i,k-1}, \forall i \in V, \forall k \in SO_i, k > 1 \tag{5}$$

Les contraintes (6) déterminent les dates de début pour les opérations de démarrage (première sous-opération) qui appartiennent à des jobs différents si elles doivent être traitées sur la même

machine. Il s'agit de contraintes adaptées à la décomposition en sous-opérations par rapport au problème classique (voir équation ii'' – chapitre II).

$$s_{j,1} - s_{i,|SO_i|} - Hx_{i,|SO_i|,j,1} \geq P_{i,|SO_i|} - H, \forall (i,j) \in V, J_i \neq J_j, \mu_i = \mu_j \quad (6)$$

Les contraintes (7), (8) et (9) sont des contraintes de conservation de flots. La contrainte (7) évite de dépasser le seuil de puissance maximal lors de la détermination des flots sortant du sommet d'origine vers toute sous-opération qu'elle soit fictive ou non. On notera que $|SO_j| = 1$ pour les opérations fictives modélisant la variation de puissance.

$$\sum_{j \in V^+} \sum_{k \in SO_j} \varphi_{0,0,j,k} \leq PT_{max} \quad (7)$$

Les contraintes (8) garantissent que la somme des flots sortant du sommet d'origine, plus ceux provenant des opérations fictives (qui modélisent les variations de puissance autorisée) et ceux venant des sous-opérations déjà effectuées pour alimenter les sous-opérations qui doivent toujours être planifiées sont égales à la puissance nécessaire à cette dernière sous-opération. Notons que, même si la date de début des opérations fictives est connue, leur besoin en puissance implique la présence d'arcs de flots vers ces opérations également. Par exemple, les flots de O, $O_{1,1}$ et $O_{3,1}$ à $O_{6,1}$ sont égaux à 44PU qui correspond à la puissance de l'opération $O_{6,1}$ (voir la Figure 8). En considérant une sous-opération comme la $l^{ième}$ sous-opération d'une opération j, cette contrainte peut être écrite comme suit :

$$\varphi_{0,0,j,l} + \sum_{i \in V^+ \setminus j} \sum_{k \in SO_i} \varphi_{i,k,j,l} + \sum_{k=1}^{l-1} \varphi_{j,k,j,l} = PC_{j,l}, \forall j \in V^+, \forall l \in SO_j \quad (8)$$

Les contraintes (9) assurent que la somme des flots sortant d'une sous-opération de l'opération i et allant vers les autres sous-opérations ou les opérations fictives n'excède jamais la puissance utilisée pour son traitement.

$$\sum_{j \in V^+ \setminus i} \sum_{l \in SO_j} \varphi_{i,k,j,l} + \sum_{l=k+1}^{|SO_i|} \varphi_{i,k,i,l} \leq PC_{i,k}, \forall i \in V^+, \forall k \in SO_i \quad (9)$$

Les contraintes (10) garantissent que les dates de début des opérations fictives qui représentent la variation du seuil de puissance dans le temps sont égales à une constante donnée. Ces variations sont connues avant de commencer le processus de planification.

$$s_{i,1} = T_{i-|V|}, \forall i \in V^+ \setminus V \quad (10)$$

Les contraintes (11), consistent à détecter la présence d'un arc de flot allant d'une sous-opération de l'opération i à une autre sous-opération de l'opération j. Si un arc de flot existe alors $y_{i,k,j,l} = 1$, sinon $y_{i,k,j,l} = 0$, c'est-à-dire qu'aucun flot ne peut transiter de i vers j.

$$\varphi_{i,k,j,l} - Hy_{i,k,j,l} \leq 0, \forall (i,j) \in V^+, \forall k \in SO_i, \forall l \in SO_j \quad (11)$$

Les contraintes (12) précisent que s'il n'y a pas besoin d'un flot allant de i à j ($\varphi_{i,k,j,l} = 0$), alors $y_{i,k,j,l} = 0$; si $y_{i,k,j,l} = 1$ alors $\varphi_{i,k,j,l} > 0$.

$$y_{i,k,j,l} - \varphi_{i,k,j,l} \leq 0, \forall (i,j) \in V^+, \forall k \in SO_i, \forall l \in SO_j \quad (12)$$

Les contraintes (13), permettent d'ajuster la date de début de chaque sous-opération de l'opération j en fonction des arcs provenant de toute sous-opération d'une opération i. Ces arcs sont modélisés par $y_{i,k,j,l}$ dans les contraintes 11-12. Si aucun arc n'existe entre deux sous-opérations, alors $y_{i,k,j,l} = 0$ et ces contraintes sont toujours vérifiées grâce à la constante H.

$$s_{j,l} - s_{i,k} - Hy_{i,k,j,l} \geq P_{i,k} - H, \forall (i,j) \in V^+, \forall (k,l) \in \left(SO_i, SO_j\right), J_i \neq J_j \quad (13)$$

Les contraintes (14) précisent qu'aucun flot n'est possible d'une opération d'un job vers une opération qui lui est antérieure dans ce même job.

$$\varphi_{i,k,j,l} = 0, \forall (i,j) \in V, \forall (k,l) \in (SO_i, SO_j), (j < i, J_i = J_j) \vee (i = j, l < k) \quad (14)$$

Comme le problème est une extension du problème de Job-shop qui est connu pour être NP - difficile (Garey et al., 1976), c'est aussi un problème NP-difficile. Ainsi, on s'attend à ce que l'obtention de solutions exactes soit difficile et prenne du temps, surtout lorsque la taille des instances augmente. Par conséquent, un ensemble de métaheuristiques efficaces pour ce problème est proposé. Les spécificités de ces métaheuristiques sont présentées dans la section suivante, avant d'être directement comparées à la résolution exacte du problème.

3.3. Métaheuristiques

Avant d'approfondir la structure des métaheuristiques mises en œuvre, il est important de détailler la manière dont les solutions sont représentées pendant le processus d'optimisation. Par conséquent, la sous-section suivante est consacrée au codage et au décodage d'une solution, qui permet d'obtenir le makespan d'un ordonnancement contraint par un seuil de puissance variable.

3.3.1. Encodage et décodage d'une solution

Il a été montré dans le chapitre précèdent que la définition d'un élément de codage est une étape importante dans la réalisation d'une métaheuristique efficace. Comme le problème étudié repose sur un problème de Job-shop, il est possible d'étendre le vecteur par répétition au problème incluant des sous-opérations. Du fait de la structure du problème, avec des sous-opérations qui doivent être exécutées consécutivement, la génération d'un vecteur valide λ basé sur une représentation par répétition basée sur les sous-opérations est délicate. Si l'exemple de la Figure 8 est considéré, un vecteur par répétition valide peut-être écrit comme suit : 2 2 1 1 1 1 2 2. En d'autres termes, un vecteur où les sous-opérations relatives à une même opération sont consécutives est un vecteur valide. Cela ne signifie pas qu'il n'existe pas de vecteur basé sur les sous-opérations qui puisse conduire à une solution valide, mais il est beaucoup plus simple d'utiliser un vecteur tel qu'énoncé précédemment. Partant de ce constat, un tel vecteur permettrait d'ordonnancer toutes les sous-opérations d'une opération en une seule fois, ce qui revient à ordonnancer l'opération complète et donc il est possible de se ramener à un vecteur par répétition contenant uniquement les opérations globales. Les calculs visant à s'assurer de l'ordonnancement de toutes les sous-opérations en respectant le critère de seuil de puissance sont effectués dans la procédure d'évaluation. La taille du vecteur est donc réduite au nombre d'opérations à ordonnancer. Le vecteur résultant est codé 2 1 1 2. Pour le problème considéré, un autre vecteur est pris en compte en plus de λ. Si le seuil de puissance variable est considéré, il est possible de détecter la puissance maximale autorisée tout au long de l'horizon de planification. A partir de cette puissance maximale, le seuil de puissance est converti en une succession d'opérations qui requièrent des puissances précises. Ces opérations sont utilisées comme des puits temporaires dans le graphe. Cette succession d'opérations est un vecteur noté Γ qui contient autant d'occurrences qu'il y a de changements de puissances. Dans la Figure 8, ce vecteur est écrit 3 3 3 3 3 3 3, car ce seuil de puissance peut être vu comme étant un 3ème Job avec sept opérations distinctes. La fonction de

décodage d'un tel couple (λ, Γ) est découpée en deux algorithmes dont le premier est donné dans l'Algorithme 1.

Algorithme 1 : Evaluation

Entrée/Sortie

S	Solution après décodage de λ (vecteur par répétition)

Entrée

$data$	Données du problème
Γ	Vecteur représentant le seuil de puissance variable

Variables

$op\_M[]$	Tableau stockant la dernière opération traitée par chaque machine
$t\_Job[]$	Tableau stockant la dernière opération traitée pour chaque job
$t\_E[]$	Tableau stockant la puissance disponible à chaque instant
$t\_S[]$	Tableau stockant la date de variation de puissance au cours de l'horizon de temps
job, op	Job traité et opération à ordonnancer
$machine$	Machine requise pour effectuer l'opération
$father, fatherD$	Prédécesseur de l'opération et opération précédente sur la machine
d, dPD	Date de début de l'opération op et date de fin de l'opération précédente sur la machine

Début

1. Initialiser les valeurs de op_M à -1 ; //initialiser les structures de données
2. Initialiser les valeurs de t_Job à 0 ;
3. Initialiser t_E avec la valeur la plus élevée du seuil ;
4. **POUR** $i \coloneqq 0$ **A** $\Gamma.size$ **FAIRE**
5. Mettre à jour la puissance disponible pour chaque période ; // initialiser t_E et t_S
6. **FIN POUR**
7. **POUR** $i \coloneqq 0$ **A** $S.\lambda.size$ **FAIRE**
8. $job \coloneqq S.\lambda[i]$; // numéro du job dont une opération doit être ordonnancée
9. $op \coloneqq$ job opération ; // récupération de l'opération à ordonnancer
10. $machine \coloneqq$ machine for op ; // machine qui doit exécuter l'opération
11. $dPD \coloneqq 0$; $d \coloneqq 0$; // initialisation des dates
12. $fatherD \coloneqq -1$; $father \coloneqq -1$; // initialisation des prédécesseurs
13. **SI** $t\_Job[job] <> 0$ **ALORS** // si ce n'est pas la première fois qu'on voit le job
14. $father \coloneqq op - 1$;
15. $d \coloneqq S.End[father]$;
16. **FIN SI**
17. **SI** $(op\_M[machine] <> -1)$ **ALORS** // si ce n'est pas la première opération sur la machine
18. $fatherD \coloneqq op\_M[machine]$; // on récupère l'opération précédente sur la machine
19. $dPD \coloneqq S.End[fatherD]$;
20. **FIN SI**
21. **SI** $(dPD > d)$ **ALORS** $father \coloneqq fatherD$ et $d \coloneqq dPD$; **FIN SI** // la date la plus grande est gardée
22. **Calcul_Date_P**$(data, op, d, father, t\_E, t\_S)$; // mise à jour de d en fonction du seuil de puissance
23. $S.date[op] \coloneqq d$; $S.end[op] \coloneqq d$ + durée de op ; $S.father[op] \coloneqq father$; // stocker informations
24. Mise à jour de $t\_Job[job]$ et de $op\_M[machine]$;
25. **FIN POUR**
26. $S.makespan \coloneqq$ makespan final ;

Fin

La première partie de l'Algorithme 1 (lignes 4 à 6) consiste à planifier les opérations de Γ, ce qui signifie que le seuil de puissance disponible au long de l'horizon de temps est définitivement défini.

Chaque itération de la boucle POUR (lignes 7 à 25) est divisée en deux parties. La première partie (lignes 8 à 21) est utilisée pour déterminer la date de début de l'opération en fonction de ses prédécesseurs : à ce stade un prédécesseur peut être une opération antérieure du job ou une autre opération se produisant sur la même machine. En d'autres termes, l'algorithme construit un ordonnancement semi-actif comparable au Job-shop classique si une quantité suffisante de puissance est disponible à tout moment. Cependant, il est possible que la quantité de puissance ne soit pas suffisante et que la date de début de l'opération doive être modifiée. La procédure **Calcul_Date_P** (ligne 22) met à jour les dates de début en prenant en compte les contraintes de puissance. Les principales étapes de cette procédure qui prend en compte la puissance maximale et nominale sont détaillées dans l'Algorithme 2.

Algorithme 2 : Calcul_Date_P

Entrée

$data$	Données du problème
op	Opération à ordonnancer

Entrée/Sortie

d	Date de début de op qui doit être mise à jour si la puissance disponible est insuffisante
$father$	Prédécesseur actuel de l'opération
$t\_E[]$	Tableau stockant la puissance disponible à chaque instant
$t\_S[]$	Tableau stockant la date de variation de puissance au cours de l'horizon de temps

Variables

s_i	Indices utilisés pour vérifier que la période respecte les besoins en puissance de op
op_isOK	Variable booléenne valant vrai si l'opération est peut être ordonnancée, 0 sinon

Début

1. Initialiser s_1, s_2 et s_3 à 0 ;
2. $op\_isOK \coloneqq$ false ;
3. **TANT QUE** non op_isOK **FAIRE**
4. $s_1 \coloneqq$ indice dans t_E à partir duquel assez de puissance est disponible pour la première sous-opération;
5. Vérifier si la première sous-op peut être exécutée sur la durée nécessaire ;
6. **SI** la puissance disponible est suffisante pour la première sous-opération **ALORS**
7. Vérifier si la seconde sous-opération peut être ordonnancée également ;
8. **SI** la puissance disponible est suffisante pour la seconde sous-opération **ALORS**
9. $op\_isOK \coloneqq$ true ;
10. **SINON**
11. $s_1 \coloneqq$ indice à partir duquel la seconde sous-opération peut être placée entièrement dans t_E ;
12. **FIN SI**
13. **SINON**
14. $s_1 \coloneqq$ indice à partir duquel la première sous-opération peut être placée entièrement dans t_E ;
15. **FIN SI**
16. **FIN TQ**
17. **SI** $t\_S[s_1]$ est différent de d **ALORS**
18. Mettre à jour *father* en considérant les arcs disjonctifs liés à la puissance ;
19. **FIN SI**
20. $d \coloneqq t\_S[s_1]$;
21. Insérer d, ainsi que la date de fin de la 1ère et 2ème sous-opérations dans t_S ;
22. Déduire la puissance utilisée entre s_1 et s_3 dans t_E pour pouvoir ordonnancer les opérations suivantes ;

Fin

Dans l'Algorithme 2, s'il y a assez de puissance disponible une opération est exécutée au plus tôt en considérant la date de fin de traitement du prédécesseur considéré (arcs conjonctifs et arcs disjonctifs de la machine), sinon l'opération sera exécutée plus tard. La nouvelle date de début est donc liée aux arcs modélisant les flots ; une sous-opération commence donc toujours après la fin d'une autre sous-opération. Par conséquent, l'ordonnancement obtenu à la fin de l'algorithme d'évaluation peut être considéré comme un ordonnancement semi-actif puisque tout décalage d'une opération vers la gauche entraîne une modification dans l'ordre de traitement des opérations. En outre, si la date de début d'une opération est mise à jour, le prédécesseur de l'opération change. Le nouveau prédécesseur est une opération planifiée directement avant l'opération courante et qui consomme la plus grande quantité de puissance ayant conduit l'opération en cours à être planifiée plus tard. Par exemple, dans la Figure 8, l'opération $O_{10,1}$ est le prédécesseur de l'opération $O_{4,1}$.

3.3.2. GRASP×ELS

Comme il a été montré précédemment (chapitre II, §5.1), le GRASP×ELS est composé d'un certain nombre de phases. L'application de cette métaheuristique au problème étudié implique donc des adaptations, non pas dans le schéma de la métaheuristique, mais dans les algorithmes utilisés aux différentes étapes clés de la métaheuristique à savoir :

- La phase de construction qui permet d'obtenir un vecteur par répétition en tenant compte de la contrainte du seuil de puissance.
- La phase de recherche locale qui permet une amélioration des solutions.
- La phase de mutation qui transforme un vecteur par répétition en un nouveau, et qui permet de mieux explorer l'espace des solutions.

3.3.2.1. Phase de construction

L'approche suggérée dans le chapitre II visant à construire une séquence d'opérations en insérant une opération à chaque itération est adaptée au JSPPR. La particularité de l'heuristique de construction employée pour le problème de Job-shop avec seuil de puissance réside dans l'intégration du seuil de puissance dans le calcul des valeurs des dates de début des opérations. Dans la procédure, une liste de candidats (CL) des opérations éligibles est considérée à chaque itération (opérations pour lesquelles toutes les opérations précédentes dans la séquence du job ont déjà été ordonnancées). Pour chacune de ces opérations, le makespan temporaire après l'insertion de l'opération dans la séquence est calculé, sans tenir compte des autres opérations dans la liste CL. Toutes les contraintes sont donc vérifiées : puissance disponible et précédences vis-à-vis des prédécesseurs conjonctifs et disjonctifs. Étant donné que chaque opération a son propre profil de puissance, il est probable que certaines d'entre elles ne soient pas prises en compte immédiatement pour l'insertion, car la date à laquelle une opération peut être insérée dépend fortement du profil de puissance. Par conséquent une opération, avec un profil de puissance présentant de fortes consommations, ordonnancée trop tôt entraînerait une augmentation prématurée du makespan. Pour éviter ce cas, une liste de candidats restreints (RCL) est construite à partir de la liste CL en fonction de l'augmentation du makespan. Ceci est effectué en observant le makespan temporaire obtenu après chaque insertion potentielle et en évitant de le laisser augmenter plus qu'un ratio donné. Enfin, une opération est sélectionnée au hasard à partir de la liste

RCL et insérée dans l'ordonnancement. Le processus est répété jusqu'à ce que toutes les opérations soient insérées dans la séquence. L'Algorithme 3 détaille la procédure.

Algorithme 3 : Phase_de_Construction

Entrée

$data$	Données du problème
a	Ratio permettant d'éjecter certaines opérations de la liste restreinte
Γ	Seuil de puissance variable

Sortie

S	Solution contenant la séquence construite avec les dates de début des opérations

Variables

$minMkpn, maxMkpn$	Makespan minimal et maximal à chaque étape
p	Makespan autorisé pour choisir les jobs à insérer dans la séquence
RCL	Liste de candidats restreinte après éjection des opérations non conformes
H	Constante représentant l'infini
$mkpn$	Makespan temporaire après insertion de chaque opération au plus tôt

Début

1. $S \coloneqq \emptyset$; // initialisation de la solution
2. **POUR** i:=1 **A** $data.nbOperations$ **FAIRE**
3. $minMkpn = H$; $maxMkpn = 0$;
4. **POUR CHAQUE** opération à ordonnancer **FAIRE**
5. $mkpn \coloneqq H$;
6. Calculer les valeurs à insérer (date de début, fin,…) en prenant en compte le seuil de puissance ;
7. Stocker les informations ;
8. Mettre à jour $minMkpn$ si la date de fin est plus basse ;
9. Mettre à jour $maxMkpn$ si la date de fin est plus haute ;
10. **FIN POUR**
11. $p \coloneqq minMkpn + a * (maxMkpn - minMkpn)$; // calcul du makespan maximal autorisé
12. Construire la liste RCL avec les opérations ayant une date de fin inférieure à p ;
13. Choisir aléatoirement une opération dans la liste RCL ;
14. Insérer l'opération dans la séquence d'opérations $S.\lambda$; // encodage de la solution
15. Sauvegarder chaque valeur définitive (dates, prédécesseur, …) ;
16. **FIN POUR**
17. Retourner S ;

Fin

La valeur d'entrée a a un impact direct sur la qualité des solutions générées par l'algorithme. Si a est égal à 1, alors l'algorithme est dominant, c'est-à-dire qu'il est possible de produire une solution optimale car il est identique à une procédure de génération totalement aléatoire qui, par sa nature, peut permettre d'obtenir la solution optimale. Cependant, cela peut conduire à la génération de nombreuses séquences qui ne sont pas profitables en tant que points de départ pour la métaheuristique. Par conséquent, un compromis entre la qualité de la solution initiale et la possibilité de générer réellement un ordonnancement optimal doit être fait. Pour le problème, il a été choisi une valeur de a qui permet la génération de solutions initiales de meilleure qualité en moyenne qu'une génération aléatoire.

3.3.2.2. Phase de recherche locale

Comme la phase de construction produit rarement un optimum local, le rôle de la recherche locale est d'augmenter la qualité d'une solution donnée en explorant le voisinage de cette solution.

Dans la recherche locale proposée pour le JSPPR le voisinage de (Van Laarhoven et al., 1992) est étendu en incluant des arcs entre des sommets spécifiques à la modélisation du seuil de puissance. Dans le JSPPR, le chemin critique exploré appartient à un graphe G' obtenu en modifiant le graphe initial G. En fait, une spécificité du JSPPR est la présence d'opérations modélisant le seuil de puissance. Un chemin critique dans G peut passer par ces opérations, comme le montre la Figure 8 ($O_{7,1}$ et $O_{8,1}$ sont sur un chemin critique). Étant donné que les dates de début des opérations modélisant le seuil sont connues au début de l'ordonnancement, aucune modification ne peut être apportée à l'ordre de traitement de ces opérations ni à leurs dates de début. Or, ignorer le renversement d'un arc impliquant une de ces opérations peut conduire à manquer un mouvement qui pourrait potentiellement améliorer la qualité de la solution. Pour résoudre ce problème, de nouveaux arcs disjonctifs sont ajoutés dans le graphe en appliquant une fermeture transitive sur des sommets spécifiques suivant la propriété proposée :

Soit ρ un chemin critique d'un graphe G. Supposons que ρ passe par un ensemble K d'opérations consécutives modélisant le seuil de puissance, $K \in \Gamma$. Soient O_i et O_j des opérations qui appartiennent à des jobs différents, et Γ l'ensemble des opérations modélisant le seuil de puissance. Supposons que ces opérations soient dans $G \backslash \Gamma$ et telles qu'un arc disjonctif existe entre O_i et les opérations de K, et entre les opérations de K et O_j (i.e. les arcs disjonctifs liés à la contrainte de puissance). Un arc disjonctif peut être ajouté dans le graphique de sorte que O_i et O_j soient reliés. Ces arcs sont appelés arcs disjonctifs induits dans la suite. Un exemple est donné dans la Figure 10 sur une partie du graphique de la Figure 8.

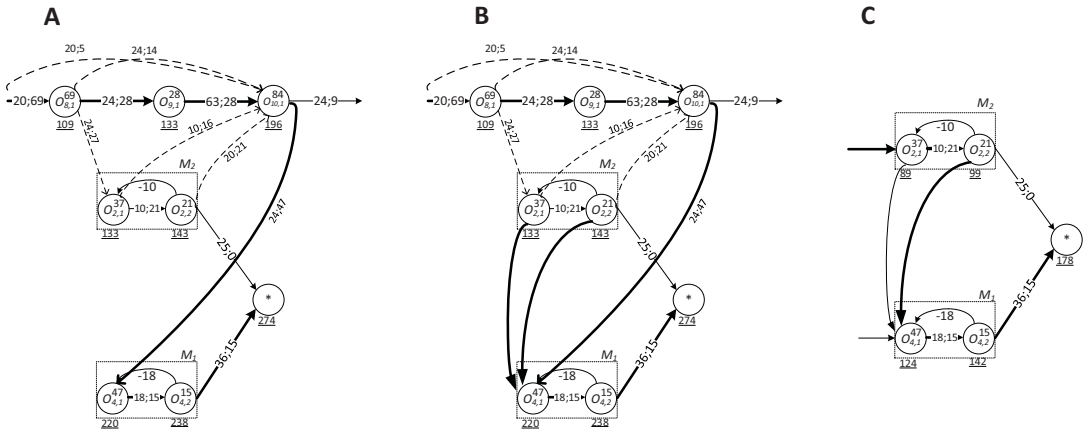

Figure 10 Principes d'ajout des arcs induits en vue de l'application de la recherche locale

Dans ce graphe partiel (Figure 10(A)), l'échange dans l'ordre de traitement des opérations $O_{10,1}$ et $O_{4,1}$ n'améliorerait pas la qualité de la solution car l'opération $O_{10,1}$ a un fort besoin en puissance qui limite l'ordonnancement des opérations, et sa date est définitive. Cependant, les arcs induits $O_{2,1} \rightarrow O_{4,1}$ et $O_{2,2} \rightarrow O_{4,1}$ peuvent être ajoutés dans le graphe. Une fois que tous les arcs disjonctifs induits sont ajoutés au graphe, il est possible de considérer un sous-ensemble G'du graphe G en supprimant toute opération du seuil de puissance et en gardant tous les arcs disjonctifs des machines ainsi que les arcs disjonctifs induits qui viennent d'être ajoutés. En appliquant un calcul de plus long chemin sur G', une partie du chemin critique de la Figure 10(B) devient comme dans la Figure 10(C). Cette propriété spécifique liée au problème étudié peut être intégrée efficacement dans les voisinages pour la définition d'une recherche locale efficace basée sur le chemin critique du graphe. Dans (Grabowski et al., 1986), l'approche par blocs basée sur la définition de bloc-machine est proposée. Comme il a

été montré dans le chapitre II (§4.1.4.1), un bloc-machine est défini comme la succession d'opérations traitées par la même machine dans la séquence et sur le chemin critique. Ici, l'approche par blocs est étendue à toute opération consécutive sur le chemin critique qui appartient à des jobs différents et liées par des arcs disjonctifs. Un mouvement est alors défini comme étant la permutation de deux opérations. Étant donné qu'une solution consiste en un graphe acyclique, la propriété principale du voisinage de (Van Laarhoven et al., 1992) sur la faisabilité d'un mouvement reste valide. Après la permutation, les arcs induits sont retirés du graphe pendant le calcul de plus long chemin. La réévaluation du graphe avec toutes les contraintes de puissance conduit à la solution exprimée sur la Figure 11, où le nouveau graphe après permutation présentant l'ordonnancement avec le seuil de puissance variable est donné. Rappelons que la solution exprimée sur la Figure 8 est encodée 2 1 1 2. La permutation de O_2 et O_4 dans le graphe, c'est-à-dire la permutation effectuée sur l'espace des solutions est équivalent à la permutation de la deuxième occurrence du job 1 avec la deuxième occurrence du job 2 dans la séquence λ (Espace de codage). Le nouveau vecteur est alors 2 1 2 1. Lors du décodage de la séquence λ, un arc lié aux consommations de puissance est présent entre $O_{4,1}$ et $O_{2,1}$, ce qui est le comportement attendu par la permutation appliquée. La Figure 11(A) montre que l'opération O_4 débute son traitement avant l'opération O_2 ce qui conduit à un meilleur makespan (187 vs 274), alors que ces deux opérations étaient ordonnancées de sorte que la chute de puissance allant de la date 196 à 220 conduisait à des dates de traitement très éloignées. La Figure 11(B) montre le diagramme de Gantt associé à cet ordonnancement.

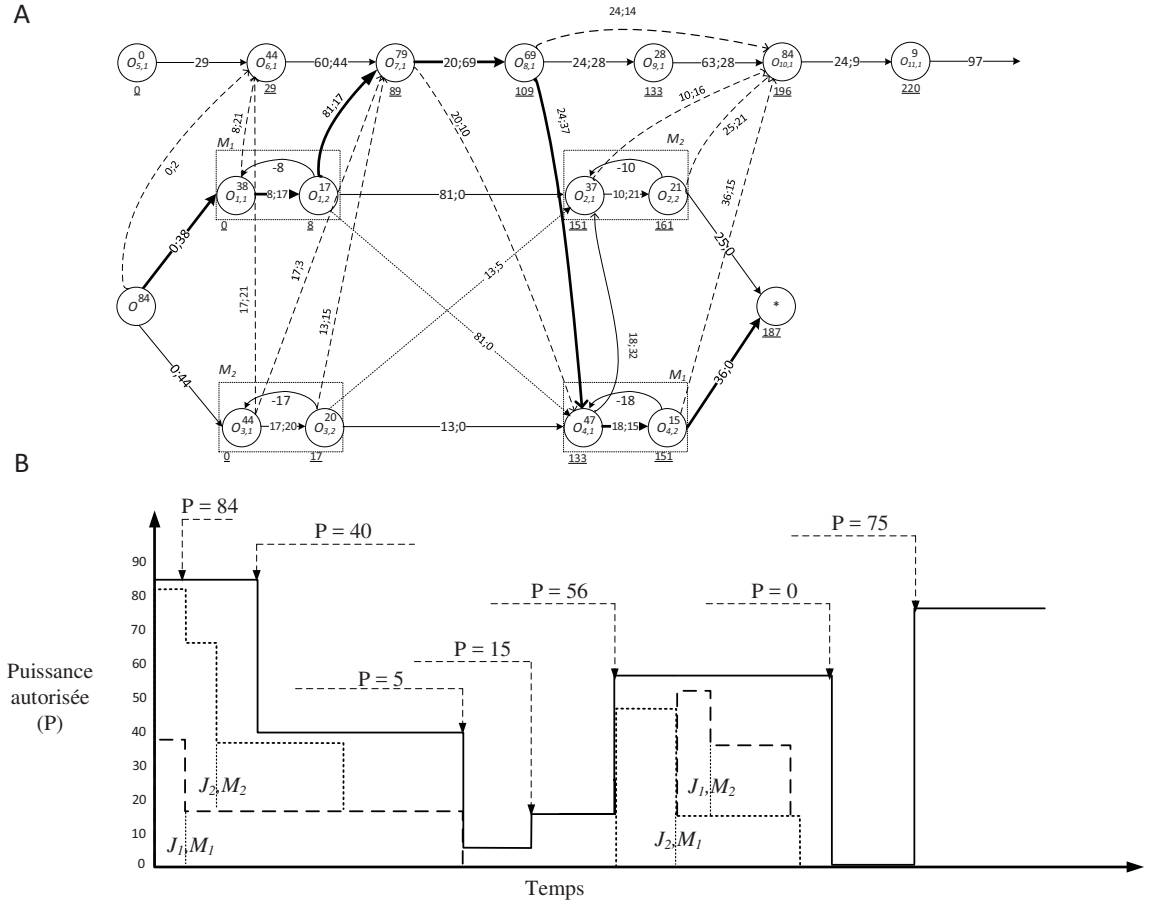

Figure 11 Nouveau graphe après permutation (A) et ordonnancement associé (B)

La procédure de recherche locale complète est donnée par l'Algorithme 4. Cet algorithme diffère de la recherche locale employée dans le chapitre II. En effet, toutes les opérations sur le chemin critique sont considérées car la présence du seuil de puissance peut conduire à des décalages à l'intérieur d'un bloc qu'il est important de vérifier. Les arcs disjonctifs induits sont ajoutés à deux endroits : à la ligne 2 et à la ligne 13. A la ligne 13, l'ajout se fait sur la nouvelle solution si la permutation conduit à une meilleure solution. Le changement de solution courante implique de reconstruire ces arcs pour pouvoir appliquer les permutations futures. L'étape consistant à reconstruire le chemin critique après l'ajout des arcs disjonctifs se fait en appliquant un calcul de plus long chemin classique pour le Job-shop, en considérant tout arc allant d'une opération à une autre (arcs conjonctifs, arcs disjonctifs, arcs induits).

Algorithme 4 : Phase_de_Recherche_Locale

Entrée

$data$	Données du problème
Γ	Seuil de puissance variable

Entrée/Sortie

S	Solution à améliorer

Variables

$tmpS$	Solution temporaire
$op, father$	Opérations consécutives sur le chemin critique de S

Début

1. **Evaluation**(S, Γ) ; // décodage de la séquence vers l'espace des solutions
2. Ajouter les arcs disjonctifs induits dans S ; // ajout d'arcs dans l'espace des solutions
3. Construire le chemin critique de S en retirant les opérations de Γ ;
4. $op \coloneqq$ dernière opération du chemin critique de S ;
5. $father \coloneqq S.father(op)$; //prédécesseur de *op* sur le chemin critique ;
6. **TANT QUE** *father* est différent du nœud source **FAIRE**
7. **SI** $(data.job(op) != data.job(father))$ **ALORS** // jobs différents ⇒ arc disjonctif
8. $tmpS \coloneqq S$;
9. Echanger op et $father$ dans $tmpS.\lambda$; // permutation dans l'espace de codage
10. **Evaluation**$(tmpS, \Gamma)$; // décodage de la séquence vers l'espace des solutions
11. **SI** $tmpS.makespan < S.makespan$ **ALORS**
12. $S \coloneqq tmpS$;
13. Ajouter les arcs disjonctifs induits dans S ; // ajout d'arcs dans l'espace des solutions
14. Construire le chemin critique de S en retirant les opérations de Γ ;
15. $op \coloneqq$ dernière opération du chemin critique de S ;
16. **SINON**
17. $op \coloneqq S.father(op)$; //on remonte le chemin critique
18. **FIN SI**
19. **FIN SI**
20. $father \coloneqq S.father(op)$;
21. **FIN SI**
22. **FIN TQ**
23. Retourner S **;**

Fin

3.3.2.3. Phase de mutation

Dans la procédure ELS pour le problème de Job-shop, les solutions voisines sont obtenues à l'aide de mutations issues des Algorithmes évolutionnaires. La mutation consiste à permuter deux opérations liées à différents jobs dans la séquence par répétition. Cette procédure de mutation est identique pour le JSPPR à celle utilisée dans le chapitre II (§5.1.3) car une amélioration du makespan peut être obtenue en modifiant les disjonctions liées aux machines mais aussi en modifiant les disjonctions liées aux arcs supportant les flots dans le graphe. Dans les deux cas ces disjonctions peuvent être modifiées en inversant l'ordre de traitement des opérations appartenant à des jobs différents dans la séquence.

3.3.3. Recherche par Voisinage Variable (VNS)

La recherche par voisinage variable a été présentée dans le chapitre I (§5.2.4.1). Elle repose sur l'utilisation de plusieurs voisinages afin d'explorer au mieux l'espace des solutions. Afin d'optimiser la recherche, ces voisinages doivent être choisis pertinemment. Il a par exemple été évoqué que deux voisinages reposant sur des transformations élémentaires proches seront moins intéressants que deux voisinages avec des transformations élémentaires très différentes.

Une version adaptée du VNS est proposée dans cette partie. La première étape de cette métaheuristique est de générer une solution initiale (à l'aide d'une heuristique de construction ou de manière aléatoire). Dans le VNS proposé, une solution est construite de manière aléatoire. Le VNS comporte également une procédure de recherche locale. L'algorithme réalisé à cet effet dans le GRASP×ELS est réutilisé sans modification dans le VNS.

La particularité du VNS réside dans l'exploration de différents voisinages, appliqués les uns après les autres. La recherche locale est appliquée à la suite de l'obtention d'un voisin et si la solution est meilleure, la solution courante est actualisée. Si aucune meilleure solution n'est trouvée à l'aide d'un voisinage, le voisinage suivant est utilisé. L'algorithme s'arrête après avoir appliqué itérativement ces voisinages un nombre de fois donné. Les différents voisinages utilisés sont présentés ci-dessous.

3.3.3.1. Permutation

Le premier voisinage appliqué est le même que celui utilisé au sein de l'ELS, ce voisinage étant facile à mettre en œuvre. Il n'est pas nécessaire de revenir sur son contenu ici.

3.3.3.2. Insertion

Le deuxième voisinage consiste à insérer une opération à un autre endroit dans la séquence d'opérations. Le nouvel emplacement est choisi de manière aléatoire. La procédure utilisée est illustrée graphiquement ci-dessous.

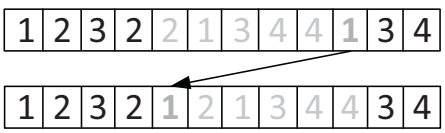

Figure 12 Exemple d'insertion dans le vecteur par répétition

Dans la Figure 12 la dernière opération du job 1 est insérée avant la dernière opération du job 2. Du fait de la renumérotation liée au vecteur par répétition, la dernière opération du job 1 est désormais en septième position dans le vecteur. Ceci n'est pas problématique, l'objectif étant de générer de nouvelles solutions.

3.3.3.3. Permutation terme à terme

Le troisième et dernier voisinage consiste à permuter terme à terme les éléments du vecteur entre deux opérations sélectionnées aléatoirement. La procédure utilisée est illustrée graphiquement sur la Figure 13.

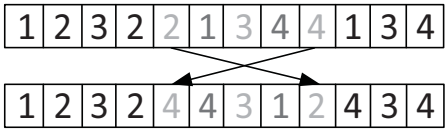

Figure 13 Exemple de permutation terme à terme dans le vecteur par répétition

Dans la Figure 13 la dernière opération du job J_2 est échangée avec la deuxième opération du job J_4. Puis la deuxième opération du job J_1 est échangée avec la première opération du job J_4. La deuxième opération du job J_3 n'est pas concernée.

3.3.4. Algorithme Mémétique (MA)

Les algorithmes mémétiques ont été évoqués dans le chapitre I (§5.2.4.7). Ces algorithmes évolutionnaires incorporent également une recherche locale afin d'améliorer la qualité des individus appartenant à la population. Une telle métaheuristique comporte différentes procédures, les principales étant : la génération d'une population initiale, un opérateur de croisements, un opérateur de mutation et une recherche locale. La taille de la population est maintenue par des algorithmes de sélection qui permettent également d'obtenir des individus plus performants au fil des générations. Ces points sont détaillés ci-dessous.

3.3.4.1. Population initiale

La population initiale est constituée de séquences générées aléatoirement. Ces séquences consistent en des vecteurs par répétition ; ils sont adaptés aux algorithmes génétiques puisqu'initialement utilisé dans ce contexte. La population initiale est générée en évitant l'apparition de doublons. La présence de doublons est gérée en calculant la signature d'une solution. Cette signature est obtenue en sommant les dates de début des opérations modulo un nombre très grand. Si une solution présentant la même signature est déjà présente dans la population, alors une nouvelle solution est générée. Si les collisions ne sont pas inévitables, l'utilisation de la signature permet de réduire fortement l'apparition de solutions identiques au sein de la population.

3.3.4.2. Croisement des individus

Les individus qui sont utilisés au sein de la population ont une structure identique à celle proposée par (Bierwirth, 1995). L'opérateur de croisement appelé GPMX est appliqué (Bierwirth et

al., 1996). L'objectif de cet opérateur est de préserver l'ordre relatif d'apparition des jobs dans les chromosomes des parents lorsqu'ils sont transférés aux fils générés.

A partir de deux parents sélectionnés par tournoi binaire, deux fils sont conçus. Dans un premier temps deux indices sont sélectionnés aléatoirement. Les valeurs contenues entre les deux indices du premier parent sont transmises au premier fils, tandis que les valeurs du deuxième parent sont transmises au deuxième fils. Ensuite, pour chaque fils, les places vacantes sont comblées en utilisant l'ordre relatif d'apparition des jobs de l'autre parent afin de conserver le même nombre d'occurrences de chaque job. La procédure est illustrée dans la Figure 14. A l'issue de la procédure de croisement, un fils est gardé de manière aléatoire (le fils présentant le meilleur makespan a une probabilité plus élevée d'être choisi).

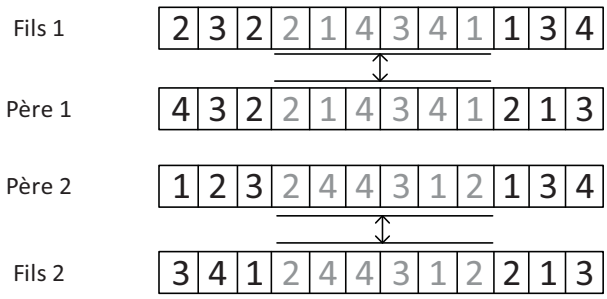

Figure 14 Exemple de croisement entre deux individus

3.3.4.3. Mutation

La mutation utilisée repose elle aussi sur celle utilisée dans le GRASP×ELS, à savoir la permutation de deux éléments d'un chromosome. Elle est appliquée avec une probabilité donnée. Après chaque mutation, la recherche locale est appliquée afin d'améliorer la qualité de l'individu obtenu. Afin de ne pas pénaliser la métaheuristique avec de nombreuses recherches locales, cette dernière est appliquée avec une probabilité fixée.

3.3.4.4. Remplacement des individus

Les algorithmes génétiques font souvent intervenir des opérateurs de sélection visant à insérer dans la population les meilleurs individus, ou à favoriser le croisement entre des individus prometteurs. L'opérateur appliqué est une sélection par tournoi. Lorsque la population est constituée des parents et des fils générés, la nouvelle population est obtenue en éliminant certains des individus non prometteurs. A chaque itération, une paire d'individus est choisie aléatoirement. L'individu préservé dans la population est celui présentant la meilleure fonction objectif avec une probabilité donnée.

3.4. Expérimentations numériques

3.4.1. Instances

Afin d'évaluer l'efficacité des algorithmes implémentés, de nouvelles instances ont été créées. Elles ont été conçues pour comparer les performances de la résolution exacte à l'aide du solveur

CPLEX. Ces instances sont composées de quatre à dix jobs et de quatre machines. Les données liées à la puissance pour ces instances ont été générées de manière aléatoire en considérant que les besoins en puissance nominale et la puissance maximale requise par une opération sont compris respectivement dans l'intervalle $[0, p/2]$ et $[0, p]$, où p représente la durée totale de l'opération. La durée du pic de puissance rencontré en début d'opération est prise dans l'intervalle $[0, p/3]$. Le seuil de puissance variable a également été généré de manière aléatoire, en commençant par la puissance minimale requise pour l'ensemble du système plus 20%. Ensuite, à un moment différent, la puissance augmente ou diminue avec un ratio donné (au début, la variation est comprise entre + 25% et -25%) ; si une augmentation se produit, la prochaine variation change et vaut + 20% en cas d'augmentation et -30% en cas de réduction. La probabilité de choisir une puissance décroissante ou croissante est également modifiée, ce qui permet de donner plus d'importance à la valeur de la variation la plus élevée en valeur absolue. Pour assurer la faisabilité des instances construites, le seuil minimal nécessaire à l'ordonnancement de l'opération présentant les besoins en puissance les plus élevés est sélectionné après un délai équivalent à la somme des durées de toutes les opérations. Toutes les instances sont disponibles en ligne (http://damienlamy.com/works/). Une comparaison des résultats obtenus sur ces instances est donnée ci-dessous. Le GRASP×ELS proposé est comparé à la métaheuristique de recherche par voisinage variable (VNS) et à l'algorithme mémétique (MA).

3.4.2. Paramétrage des métaheuristiques

Un aspect important des métaheuristiques est le réglage des paramètres. Pour le problème considéré, un plan d'expérience tel que celui pour le GRASP×ELS classique a été mené sur les nouvelles instances générées avec 9 et 10 jobs. Pour chaque ensemble de paramètres, le makespan moyen des solutions et le makespan moyen des meilleures solutions sont calculés. Le temps de calcul moyen pour atteindre les meilleures solutions est également sauvegardé.

- Ce plan d'expérience a permis de sélectionner le paramétrage suivant pour le GRASP×ELS : le nombre de redémarrages du GRASP, nb_g = 200, le nombre d'itérations de l'ELS, nb_els = 100 et le nombre de voisins générés, nb_n = 22.

- Les paramètres du VNS sont moins nombreux, car le nombre de voisinages est fixé en amont. Le seul vrai paramètre concerne le critère d'arrêt qui est fixé à 10000 itérations.

- Pour l'Algorithme mémétique, les paramètres sont les suivants : la taille de la population n = 100, la probabilité de croisement cp = 0,6, la probabilité de mutation mp = 0,1 et la probabilité de recherche locale lsp = 0,7. L'opérateur de sélection utilise la probabilité sp = 0,8 pour sélectionner les individus de la population et en générer une nouvelle.

3.4.3. Résultats

Dans le tableau présentant les résultats (Tableau 4), la partie CPLEX fait référence aux solutions exactes obtenues en utilisant le solveur linéaire. **UB**, **LB** et **GAP** renvoient respectivement à la borne supérieure (Upper Bound), à la borne inférieure (Lower Bound) et à l'écart entre ces valeurs. Un astérisque (*) est ajouté à la valeur **UB** lorsque la solution optimale est trouvée. Dans la colonne **GAP**, un tiret est utilisé pour l'instance JSPPR_4_10x4 car le solveur n'a pas réussi à trouver une solution. La colonne **CPU** réfère au temps d'exécution total en secondes ; l'exécution est arrêtée après 10800

(symbole ~). Concernant les métaheuristiques GRASP×ELS, MA et VNS, la colonne $\bar{\mathbf{S}}$ renvoie à la solution moyenne sur 10 exécutions. La colonne $\mathbf{S^+}$ correspond à la meilleure solution trouvée par les métaheuristiques. $\bar{\mathbf{S}}_{\boldsymbol{CPU}}$ correspond au temps de calcul total moyen tandis que $\mathbf{S^+_{CPU}}$ renvoie à la durée observée pour obtenir la valeur présentée dans la colonne $\mathbf{S^+}$. $\mathbf{S_{DEV}}$ renvoie à la déviation entre $\bar{\mathbf{S}}$ et $\mathbf{S^+}$. Enfin, $\mathbf{UB_{DEV}}$ et $\mathbf{LB_{DEV}}$ correspondent à la déviation entre les valeurs de la colonne $\mathbf{S^+}$ par rapport aux colonnes **UB** et **LB** de la partie CPLEX. Ces notations sont présentées dans le Tableau 3 pour gagner en lisibilité.

Tableau 3 Récapitulatif des notations utilisées

CPLEX	UB	Borne supérieure retournée par CPLEX
	LB	Borne inférieure retournée par CPLEX
	GAP	Déviation entre UB et LB
	CPU	Durée nécessaire pour démontrer l'optimalité
Métaheuristiques	$\bar{S}$	Valeur moyenne du makespan
	$\bar{S}_{CPU}$	Temps de calcul moyen
	S^+	Meilleure valeur du makespan
	S^+_{CPU}	Temps de calcul pour obtenir la meilleure solution
	S_{DEV}	Déviation entre $\bar{S}$ et S^+
	UB_{DEV}	Déviation entre S^+ et UB
	LB_{DEV}	Déviation entre S^+ et LB

Comme le montre le Tableau 4, le solveur linéaire éprouve des difficultés à trouver des solutions optimales lorsque la taille des problèmes augmente. Ceci est tout à fait normal, car le solveur doit faire face à beaucoup de contraintes telles que les contraintes de Time-lags modélisant la non-préemption entre les sous-opérations, alors que ces contraintes peuvent être simplifiées dans les métaheuristiques en raison du codage choisi pour les solutions. Bien que la déviation moyenne soit inférieure à 0,5% pour les instances 5x4 et 7x4, le temps de calcul diffère légèrement entre ces problèmes en raison de leur taille. Une raison pour laquelle les instances 6x4 présentent un GAP moyen plus grand pourrait être liée au seuil de puissance qui a un impact non négligeable sur la résolution de ce type de problèmes comme souligné par (Bruzzone et al., 2012). En considérant les instances 8x4, 9x4 et 10x4, les valeurs présentes dans la colonne GAP sont très importantes. Par ailleurs, les métaheuristiques semblent avoir un comportement intéressant dans un temps de calcul relativement court. Le GRASP×ELS est très efficace pour obtenir des solutions proches pour une même instance, ce qui entraîne une valeur de S_{DEV} moyenne de 0,41. Ce résultat n'est impacté qu'en raison de l'instance JSPPR_4_10x4, qui semble être l'instance la plus difficile dans l'ensemble de données.

Le VNS surpasse le GRASP×ELS du point de vu de la colonne S^+_{CPU} et trouve le même nombre de solutions optimales. Cependant, la valeur LB_{DEV} est fortement dégradée par rapport à celle du GRASP×ELS et du MA. Cette dernière métaheuristique est la plus proche des performances du GRASP×ELS sur l'ensemble de données, avec des résultats comparables à ceux du GRASP×ELS. Cependant, ce dernier offre un meilleur comportement quant à la convergence vers la solution optimale, comme le souligne la colonne S_{DEV}. Le tableau montre que les valeurs moyennes pour LB_{DEV} sont assez importantes pour toutes les métaheuristiques, et que les méthodes ont des performances similaires sur l'ensemble des instances. Par conséquent, il n'est pas possible de conclure sur la qualité des solutions. À cette fin, d'autres études peuvent être menées sur l'approche de résolution exacte en considérant une relaxation lagrangienne du problème pour améliorer les bornes inférieures et supérieures et avoir de meilleurs éléments de comparaison.

Tableau 4 Résultats du GRASP×ELS, VNS et MA sur les instances JSPPR

INSTANCE	CPLEX				GRASP×ELS							VNS							MA						
	UB	LB	GAP	CPU	$\bar{S}$	S^+	$\bar{S}_{CPU}$	S^+_{CPU}	S_{DEV}	UB_{DEV}	LB_{DEV}	$\bar{S}$	S^+	$\bar{S}_{CPU}$	S^+_{CPU}	S_{DEV}	UB_{DEV}	LB_{DEV}	$\bar{S}$	S^+	$\bar{S}_{CPU}$	S^+_{CPU}	S_{DEV}	UB_{DEV}	LB_{DEV}
JSPPR_1_4x4	300*	300	0	6	300	300*	6	0	0	0	0	300	300*	1	0	0	0	0	300	300*	4	0	0	0	0
JSPPR_2_4x4	475*	475	0	181	475	475*	5	0	0	0	0	475	475*	1	0	0	0	0	475	475*	5	0	0	0	0
JSPPR_3_4x4	298*	298	0	10	298	298*	6	0	0	0	0	298	298*	2	0	0	0	0	298	298*	5	0	0	0	0
JSPPR_4_4x4	297*	297	0	2	297	297*	4	0	0	0	0	297	297*	1	0	0	0	0	297	297*	4	0	0	0	0
JSPPR_5_4x4	318*	318	0	1	318	318*	4	0	0	0	0	318	318*	1	0	0	0	0	318	318*	3	0	0	0	0
JSPPR_1_5x4	344*	344	0	6	344	344*	9	0	0	0	0	344	344*	2	0	0	0	0	344	344*	7	0	0	0	0
JSPPR_2_5x4	447*	447	0	329	447	447*	7	0	0	0	0	447	447*	2	0	0	0	0	447	447*	7	0	0	0	0
JSPPR_3_5x4	374	373	0.27	~	374	374	9	0	0	0	0.27	374.6	374	3	1	0.16	0	0.27	374.6	374	9	1	0.16	0	0.27
JSPPR_4_5x4	332	331	0.3	~	332	332	5	0	0	0	0.30	332	332	1	0	0	0	0.30	332	332	6	0	0	0	0.30
JSPPR_5_5x4	350	349	0.29	~	350	350	7	0	0	0	0.29	350	350	2	0	0	0	0.29	350	350	5	0	0	0	0.29
JSPPR_1_6x4	436*	436	0	1976	436	436*	12	0	0	0	0	436	436*	4	0	0	0	0	436	436*	11	0	0	0	0
JSPPR_2_6x4	521	462	11.32	~	521	521	15	0	0	0	12.77	521	521	4	0	0	0	12.77	522.2	521	9	0	0.23	0	12.77
JSPPR_3_6x4	459*	459	0	74	459	459*	12	0	0	0	0	459	459*	5	1	0	0	0	459.4	459*	10	3	0.09	0	0
JSPPR_4_6x4	418	348	16.75	~	418	418	9	0	0	0	20.11	418	418	2	0	0	0	20.11	420.9	418	9	2	0.69	0	20.11
JSPPR_5_6x4	473	392	17.12	~	437	437	12	2	0	-7.61	11.48	453.2	443	3	2	2.30	-6.34	13.01	460.8	437	10	4	5.45	-7.61	11.48
JSPPR_1_7x4	456*	456	0	1412	456	456*	16	0	0	0	0	457.3	456*	5	2	0.29	0	0	456.7	456*	15	1	0.15	0	0
JSPPR_2_7x4	478*	478	0	8621	478	478*	18	0	0	0	0	478	478*	6	0	0	0	0	478.6	478*	12	0	0.13	0	0
JSPPR_3_7x4	495*	495	0	1018	495	495*	15	0	0	0	0	495	495*	4	0	0	0	0	495	495*	14	0	0	0	0
JSPPR_4_7x4	395	394	0.25	~	395	395	12	0	0	0	0.25	395	395	3	1	0	0	0.25	395	395	11	0	0	0	0.25
JSPPR_5_7x4	429	418	2.56	~	429	429	17	0	0	0	2.63	429	429	4	0	0	0	2.63	429	429	14	0	0	0	2.63
JSPPR_1_8x4	1754	437	75.09	~	485	485	24	0	0	-72.35	10.98	485.9	485	7	1	0.19	-72.35	10.98	485	485	19	4	0	-72.35	10.98
JSPPR_2_8x4	1881	507	73.05	~	579	579	28	2	0	-69.22	14.20	591.5	581	9	5	1.81	-69.11	14.60	588.7	579	20	1	1.68	-69.22	14.20
JSPPR_3_8x4	527	503	4.55	~	527	527	26	0	0	0	4.77	527	527	9	0	0	0	4.77	527	527	21	0	0	0	4.77
JSPPR_4_8x4	1588	395	75.13	~	487	487	21	2	0	-69.33	23.29	496.5	494	5	2	0.51	-68.89	25.06	491.1	487	20	1	0.84	-69.33	23.29
JSPPR_5_8x4	696	384	44.83	~	468	468	25	1	0	-32.76	21.88	480.3	476	7	4	0.90	-31.61	23.96	472.2	468	21	3	0.90	-32.76	21.88
JSPPR_1_9x4	1201	462	61.53	~	580.8	579	33	16	0.31	-51.79	25.32	619.9	597	11	7	3.84	-50.29	29.22	593.8	579	29	6	2.56	-51.79	25.32
JSPPR_2_9x4	2233	521	76.67	~	644	643	35	9	0.16	-71.20	23.42	682	665	13	5	2.56	-70.22	27.64	651.2	643	25	10	1.28	-71.20	23.42
JSPPR_3_9x4	2018	458	77.3	~	558.3	558	31	3	0.05	-72.35	21.83	574.9	567	12	6	1.39	-71.9	23.80	575	558	26	3	3.05	-72.35	21.83
JSPPR_4_9x4	2136	414	80.62	~	639	639	30	17	0	-70.08	54.35	649	649	9	4	0	-69.62	56.76	649	649	27	0	0	-69.62	56.76
JSPPR_5_9x4	1868	408	78.16	~	510.3	506	38	18	0.85	-72.91	24.02	531.2	529	13	8	0.42	-71.68	29.66	519.5	513	28	9	1.27	-72.54	25.74
JSPPR_1_10x4	2458	471	80.48	~	609	609	35	1	0	-75.22	29.30	635.5	629	6	4	1.03	-74.41	33.55	611	609	34	7	0.33	-75.22	29.30
JSPPR_2_10x4	2717	459	83.11	~	656.1	653	40	16	0.47	-75.97	42.27	727.4	701	9	6	3.77	-74.2	52.72	669.2	655	35	16	2.17	-75.89	42.70
JSPPR_3_10x4	2289	453	80.21	~	588	588	29	0	0	-74.31	29.80	589.8	588	7	0	0.31	-74.31	29.80	588	588	32	0	0	-74.31	29.80
JSPPR_4_10x4	-	448	∞	~	766.2	680	28	14	12.68	-	51.79	1309	1066	8	2	22.80	-	137.95	1011.1	680	33	10	48.69	-	51.79
JSPPR_5_10x4	2442	394	83.87	~	593	593	40	0	0	-75.72	50.51	593	593	7	0	0	-75.72	50.51	593	593	34	0	0	-75.72	50.51
Moyenne :							19	3	0.41	-26.20	13.60	-	-	5	2	1.21	-25.9	17.16			16	2	1.99	-26.17	13.73

3.5. Conclusion

Dans cette étude, un modèle mathématique et une approche pour résoudre un problème de Job-shop avec seuil de puissance variable sont proposés. L'objectif est de trouver des ordonnancements qui respectent un seuil de puissance soumis à des variations pendant l'horizon de planification. Ce seuil de puissance est une contrainte donnée dans le présent travail. Chaque opération à ordonnancer présente un profil de puissance. Pour qu'une opération soit exécutée, une puissance suffisante doit être disponible tout au long de la réalisation de l'opération. La puissance requise par chaque opération est appelée profil de puissance. Dans cette étude le profil de puissance des opérations est divisé en deux parties : un pic de puissance au début de l'usinage et une consommation nominale ensuite. Une métaheuristique basée sur un GRASP×ELS, ainsi qu'un VNS et un MA sont proposés. Le but de ces métaheuristiques est d'explorer l'espace des solutions afin de trouver des solutions de bonne qualité, c'est-à-dire minimisant le makespan tout en respectant le seuil de puissance. Les performances des métaheuristiques sont évaluées sur un premier ensemble d'instances générées. Les solutions des algorithmes sont comparées aux solutions fournies par le solveur CPLEX. Sur ces instances, le GRASP×ELS fournit des solutions égales ou meilleures que le solveur et les autres métaheuristiques.

4. Approches biobjectifs au problème de Job-shop avec puissance

Comme il a été montré dans le chapitre I (§5.1), l'optimisation multiobjectif a pour but de trouver les meilleures solutions en considérant plusieurs objectifs. Trois approches sont généralement appliquées pour ce type de problèmes : la première consiste à optimiser une combinaison linéaire des différents objectifs ce qui permet de revenir à un problème mono-objectif ; la seconde consiste à chercher la meilleure solution en fixant un des objectifs ; la dernière concerne l'optimisation de l'ensemble des objectifs en reposant sur une approche de type Pareto.

Dans cette section, deux métaheuristiques biobjectifs sont proposées afin de proposer des solutions à un cas particulier du problème précédent, celui avec seuil de puissance constant. Le but consiste à définir un ensemble de solutions réalisables ayant des seuils de puissances différents. Parmi les deux approches explorées, la première repose sur une utilisation itérée du GRASP×ELS, tandis que la deuxième est basée sur un algorithme de type NSGA II.

Si la première métaheuristique construit le front de Pareto un élément après l'autre, en fixant à chaque nouvelle application du GRASP×ELS un nouveau seuil de puissance, la seconde manipule une population d'individus. La notion de dominance qui permet de comparer deux solutions entre elles est au cœur de cette dernière approche. Cependant, deux solutions peuvent être incomparables, c'est-à-dire qu'aucune ne domine l'autre. C'est le cas lorsque deux solutions appartiennent au même front de Pareto. Il est cependant possible de définir des critères de comparaison différents comme il sera montré par la suite. Le fait qu'il existe des solutions non dominées complexifie les méthodes de résolution. En effet, il n'existe pas une ou plusieurs solutions minimisant un unique critère, mais un ensemble de solutions pour lesquelles les valeurs des critères de performance sont incomparables.

Le choix du NSGA-II est motivé par le fait que cette approche a montré son efficacité dans la résolution de problèmes réels jusqu'à trois objectifs (Bechikh et al., 2017 ; Zhou et al., 2011). Les éléments de cette métaheuristique sont détaillés dans cette section.

Dans un premier temps, le problème à l'étude est défini plus précisément en s'appuyant sur la définition faite pour le JSPPR. La formalisation mathématique diffère peu de celle présentée dans le cas du seuil variable ; elle peut être trouvée en Annexe 1. Les deux métaheuristiques développées spécifiquement pour le problème biobjectif sont ensuite présentées et évaluées sur des jeux de données générés à partir d'instances de la littérature. Enfin, la dernière partie de cette section conclut sur des perspectives de recherche.

4.1. Présentation du problème

Cette section traite du problème biobjectif visant à définir un front de Pareto des solutions ayant des seuils de puissance et des makespan différents. Les opérations des jobs à réaliser sont telles que présentées dans la section précédente, à savoir des opérations avec un pic de puissance en début d'usinage et une consommation nominale par la suite. Le problème est cependant légèrement différent car le seuil de puissance n'est pas considéré variable, mais constant. En d'autres termes l'objectif est de définir la puissance maximale requise pour obtenir chaque ordonnancement retourné par les métaheuristiques. Si l'on considère un exemple avec 3 jobs et 4 machines, il est possible d'obtenir deux diagrammes de Gantt tels que présentés dans les Figure 15 et Figure 16. La Figure 15 montre une solution d'un problème avec un seuil de puissance fixé à 53. Le makespan obtenu vaut 77. La courbe noire continue donnée en dessous du diagramme de Gantt correspond à la puissance injectée dans le système à chaque instant. Cette courbe ne dépasse jamais le seuil fixé (ligne noire horizontale). Un autre diagramme de Gantt et une courbe de puissance différente pour un seuil supérieur (94) sont présentés sur la Figure 16. Comme attendu, un seuil supérieur conduit à un meilleur makespan (69 dans ce cas). Sur ces deux figures, il est facile de remarquer que l'augmentation du seuil de puissance permet de planifier des opérations plus tôt, ce qui se traduit par un meilleur makespan. En outre, un responsable de production peut aller plus loin en utilisant une solution afin de négocier un seuil de puissance variable, ce qui peut conduire à une réduction des tarifs. En réalité, sur la Figure 16, il apparaît que le seuil de puissance fixé à 94 n'est pas nécessaire tout au long de l'horizon de temps et peut être réduit à 60 après 25 unités de temps. Il en va de même pour la Figure 15, où le seuil peut être réduit à 30, de 46 à 59 unités de temps. Cependant, comme il ne sera pas possible d'avoir un meilleur makespan pour le seuil de puissance variable et pour préserver un bon compromis entre le temps de calcul et la qualité de la solution du point de vue du makespan, le choix est fait de se concentrer uniquement sur le seuil de puissance maximale. Par conséquent, l'optimisation du seuil de puissance final est laissée à un algorithme appliqué en post-optimisation si nécessaire.

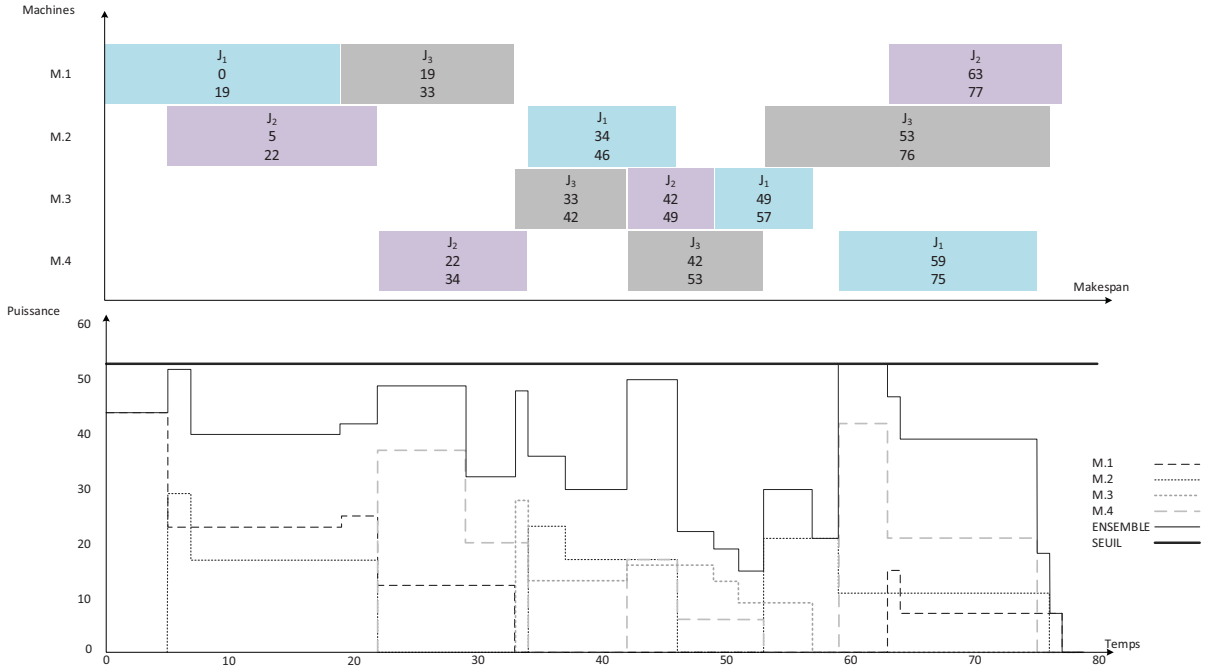

Figure 15 Diagramme de Gantt et puissances requises pour un problème avec seuil à 53

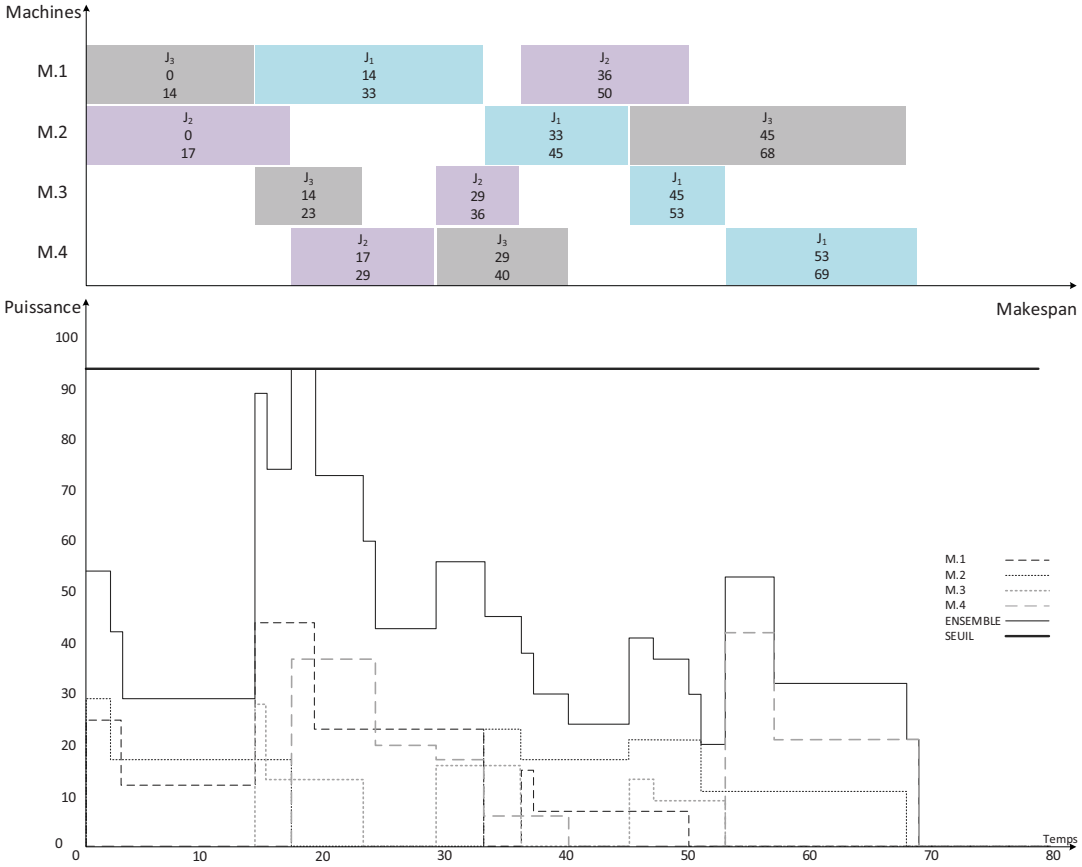

Figure 16 Diagramme de Gantt et puissances requises pour un problème avec seuil à 94

Deux difficultés se présentent. La première concerne la comparaison des solutions car il n'est pas possible de dire si la solution avec le seuil de puissance le plus élevé et un plus faible makespan est meilleure ou pire que la solution avec le seuil de puissance le plus bas et la valeur de makespan plus élevée. La deuxième est liée à la complexité du problème. Puisqu'il est NP-difficile (Garey et al., 1976), et parce qu'il faut un temps de calcul important pour obtenir une solution du front de Pareto, trouver toutes les solutions exactes constituant un tel front prendrait un temps de calcul important, comme il a d'ailleurs été souligné par le travail de (Fang et al., 2011). Par conséquent, l'utilisation des métaheuristiques est appropriée.

Par rapport au problème énoncé dans la section précédente, une fois qu'un seuil de puissance est considéré, il ne va pas varier dans le temps. Un tel problème exprimé sous forme de graphe ne nécessite pas la considération d'opérations fictives telles que présentées précédemment. Ainsi, la solution dont le diagramme de Gantt est montré sur la Figure 15 peut être modélisée par le graphe ci-dessous.

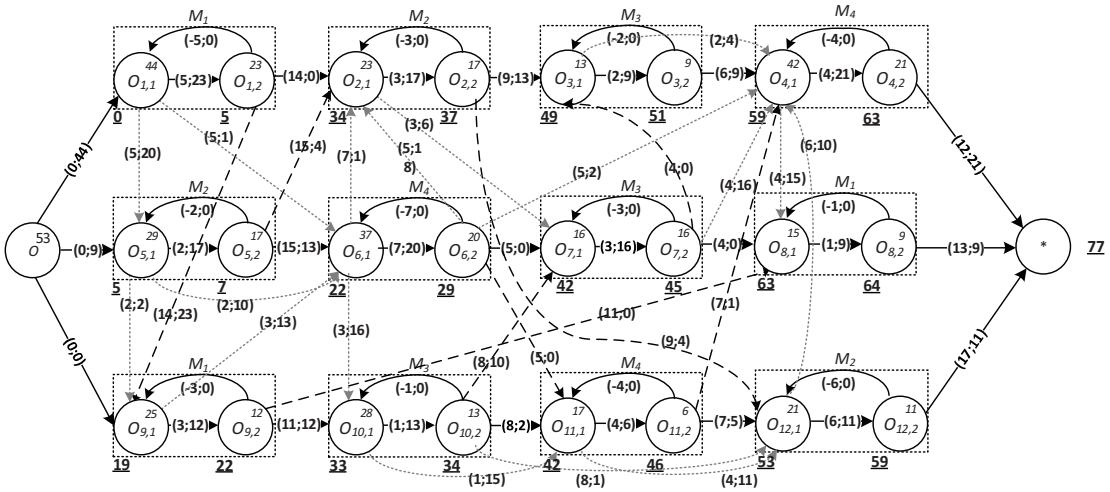

Figure 17 Graphe disjonctif orienté d'un problème avec seuil constant

Dans le graphe de la Figure 17, les différents outils appliqués dans la formalisation mathématique utilisée dans la section précédente interviennent. Par exemple, les arcs de Time-lags max entre les sous-opérations pour respecter la non-préemption sont présents. Des arcs en tirets sont utilisés pour modéliser les disjonctions machines, tandis que les arcs en pointillés représentent les flots. Comme le seuil est constant au cours de l'horizon d'ordonnancement, il n'y a cependant pas besoin d'opérations fictives comme celles utilisées dans la section précédente. Seule la puissance initialement stockée dans le sommet modélisant la source dans le graphe est utile, la résolution du problème de flot s'assurant du respect de cette contrainte (le seuil de puissance à respecter).

Les travaux réalisés ici visent donc à fixer la valeur du seuil, ici O^{53} et la valeur de makespan associée (77). Afin de différencier le problème de celui traité précédemment, il est noté Bi-JSPPR. Pour apporter des solutions sous forme de fronts de Pareto, deux métaheuristiques sont appliquées dont les structures sont présentées dans la suite.

4.2. Métaheuristiques

Comme l'ont noté (Lacomme et al., 2006), plusieurs algorithmes multiobjectifs sont disponibles dans la littérature et on en rencontre de plus en plus. Par conséquent, choisir la métaheuristique la plus appropriée à un problème d'optimisation multiobjectif donné n'est pas trivial. Dans ce travail, deux métaheuristiques sont développées afin de créer des fronts de Pareto pour le Bi-JSPPR ; ces métaheuristiques sont basées sur un GRASP×ELS, et sur un NSGA-II (Deb et al., 2002). La première métaheuristique consiste en un GRASP×ELS itéré, la seconde est un NSGA-II hybride. Par la suite, les métaheuristiques proposées sont nommées iGRASP×ELS et hNSGA-II. Le GRASP×ELS classique a été utilisé dans plusieurs problèmes à un seul objectif (Duhamel et al., 2010; Hu and Lim, 2014; Lacomme et al., 2013b), et il a été appliqué dans la section précédente, tandis que le NSGA-II est l'un des algorithmes multiobjectifs les plus utilisés dans la littérature. De manière générale, les métaheuristiques partagent certains éléments ou procédures. Dans le problème étudié, le codage des solutions et l'algorithme d'évaluation sont les mêmes pour le iGRASP×ELS et le hNSGA-II. Si la procédure d'évaluation n'intègre pas de seuil variable, et est donc simplifiée de ce point de vue, l'ensemble des instructions reste similaire. Afin de faciliter la lecture et la compréhension, l'évaluation d'une solution est rappelée.

4.2.1. Encodage et décodage d'une solution

Comme il a été vu dans le chapitre précèdent, la définition d'un élément de codage est une étape importante dans la réalisation d'une métaheuristique efficace. Le vecteur permettant le codage d'une solution partielle, noté λ, est repris de la section précédente, la structure du problème n'étant pas modifiée. La fonction de décodage ne requiert que la séquence d'opérations λ et la puissance maximale, cette dernière remplaçant le vecteur représentant le seuil variable. La procédure est globalement donnée dans l'Algorithme 5.

Dans cet algorithme, les lignes 4-8 correspondent au calcul de la date de début d'une opération en fonction de la séquence d'opérations du job (i.e. la deuxième opération du job J_1 doit être démarrée après l'achèvement de la première opération de J_1). Les lignes 9-13 permettent le calcul de la date de début en fonction des opérations planifiées précédemment sur la machine requise (sur la Figure 15 la première opération du job J_3 se produit après la première opération du job J_1 sur la machine M_1). Enfin, les lignes 14-18 consistent à définir les dates de début de l'opération en fonction de la puissance instantanée disponible. Cette procédure est identique à celle présentée dans l'Algorithme 2. Il revient alors aux métaheuristiques de définir les séquences à évaluer et le seuil de puissance appliqué. Ces métaheuristiques sont présentées dans la suite.

Algorithme 5 : Evaluer

Entrée/Sortie

S	Solution après décodage de λ (vecteur par répétition)

Entrée

$maxP$	Puissance maximale autorisée au cours de l'ordonnancement

Variables

job, op	Job traité et opération à ordonnancer
$machine$	Machine requise pour effectuer l'opération
j_date, m_date, p_date	Dates de début temporaires pour une opération
$start\_date$	Date de début final pour une opération

Début

1. **POUR** chaque job dans $S.\lambda$ **FAIRE**
2. $op \coloneqq$ opération correspondant à l'occurrence du job en cours de traitement ;
3. $machine \coloneqq$ machine requise pour exécuter l'opération ;
4. **SI** op est la première opération du job **ALORS** // première occurrence du job
5. $j\_date \coloneqq 0$;
6. **SINON**
7. $j\_date \coloneqq$ date de fin de la dernière opération du job ;
8. **FIN SI**
9. **SI** aucune opération n'a été traitée sur la $machine$ **ALORS** // premier job traité par la machine
10. $m\_date \coloneqq 0$;
11. **SINON**
12. $m\_date \coloneqq$ date de fin de la dernière opération exécutée sur la $machine$;
13. **FIN SI**
14. **SI** suffisamment de puissance est disponible pour exécuter l'opération **ALORS**
15. $p\_date \coloneqq 0$;
16. **SINON**
17. $p\_date \coloneqq$ date à partir de laquelle suffisamment de puissance est disponible ;
18. **FIN SI**
19. $start\_date \coloneqq max(j\_date\ ;\ m\_date\ ;\ p\_date)$
20. Mise à jour de toute information supplémentaire (date de fin, etc.)
21. **FIN POUR**
22. $S.makespan \coloneqq$ makespan final ;

Fin

4.2.2. GRASP×ELS itéré

Le GRASP×ELS est généralement utilisé pour des problèmes mono-objectifs pour lesquels il s'est révélé très performant (Duhamel et al., 2010; Lacomme et al., 2013b). Cette métaheuristique a également été appliquée au Job-shop avec seuil de puissance constant (Kemmoe et al., 2015c). Dans cette étude, un problème biobjectif est considéré. Pour pouvoir appliquer le GRASP×ELS, il est choisi d'appliquer cette métaheuristique itérativement avec plusieurs seuils de puissance (c'est-à-dire que le problème est réduit à un seul objectif). Par conséquent, le GRASP×ELS est transformé en un GRASP×ELS itéré (iGRASP×ELS) dans lequel un critère (le makespan) est optimisé tout en considérant l'autre critère fixé (le seuil de puissance). Cet iGRASP×ELS est présenté dans l'Algorithme 6. Une fois que le seuil de puissance est fixé, le GRASP×ELS repose sur les trois phases déjà évoquées dans la section précédente :

- La phase de construction : une première solution est construite avec une heuristique randomisée. À chaque itération de la construction, toutes les opérations qui peuvent être planifiées sont insérées dans une liste de candidats. Ensuite, un élément est choisi de manière aléatoire dans cette liste s'il suit certaines exigences, qui sont habituellement la non-augmentation du critère temporel. L'heuristique développée pour le problème du JSPPR en considérant un seuil de puissance constant est appliqué lors de cette phase.

- La phase de recherche locale : Cet algorithme est peu ou prou le même que dans la section précédente pour le GRASP×ELS. Cependant, il n'est pas nécessaire d'ajouter d'arcs disjonctifs induits du fait de l'absence d'opérations fictives modélisant un seuil variable. Ainsi, les seuls prédécesseurs potentiels sont directement les opérations du problème. La recherche locale consiste donc à parcourir le chemin critique et à permuter les opérations en disjonction, qu'elles soient liées par un arc disjonctif relatif aux machines, ou à un arc ajouté en cas de puissance insuffisante.

- La phase d'ELS : au cours de cette phase, le voisinage d'une solution est exploré en générant un ensemble de voisins. Pour le iGRASP×ELS, un voisin consiste à permuter deux opérations dans le vecteur qui représente une solution du problème, de la même manière que pour le JSPPR. La recherche locale est appliquée sur ce voisin en conservant toujours le même seuil de puissance. Le meilleur voisin de l'ensemble est sélectionné et il est ensuite utilisé comme point de départ de l'itération ELS suivante. La qualité de ce meilleur voisin est comparée à la meilleure solution trouvée et, si nécessaire, cette meilleure solution est mise à jour.

À la ligne 1 de l'Algorithme 6, le seuil de puissance initial est défini. Ensuite, à chaque itération de la boucle POUR (lignes 2-21), un élément du front de Pareto est recherché. Le GRASP×ELS est appliqué entre les lignes 4 et 18. À la ligne 19, la meilleure solution trouvée au cours du GRASP×ELS est insérée dans le front de Pareto s'il s'agit d'une solution non dominée. Toutes les solutions qui pourraient se retrouver dominées par l'arrivée de la nouvelle sont effacées de cette population. À la ligne 20, le prochain seuil de puissance est calculé. La valeur du pas est égale à la différence entre le seuil de puissance maximal qui peut être calculé avec une première exécution d'un GRASP×ELS sans tenir compte de la contrainte de seuil de puissance ($maxP$) et du seuil de puissance minimal sous lequel aucune solution n'existe ($minP$). Cette valeur est ensuite divisée par la taille maximale autorisée pour le front de Pareto ($paretoMaxSize$).

$$step = \frac{maxP - minP}{paretoMaxSize}$$

Comme dans la section précédente, la recherche locale tient un rôle prépondérant dans la qualité des solutions retournées par le GRASP×ELS. Du fait de la structure du problème, la recherche locale est adaptée du problème précédent. Elle n'est pas présentée ici car elle n'est pas significativement différente de celle présentée pour le JSPPR.

Algorithme 6 : iGRASP×ELS

Entrée

nb_g	Nombre de redémarrages (GRASP)
nb_els	Nombre d'itérations de l'ELS
nb_n	Nombre de voisins générés dans l'ELS
$initPow$	Seuil de puissance minimal en dessous duquel aucune solution ne peut être obtenue
m_pop	Taille maximale du front de Pareto
$step$	Pas de modification du seuil de puissance

Sortie

$pFront$	Front de Pareto obtenu à la fin de l'algorithme

Variables

S	Solution utilisée au cours de l'algorithme
nS, nS^*	Voisin et meilleur voisin trouvés durant les phases d'ELS
S^*	Meilleure solution trouvée pour un seuil de puissance fixé
pow	Seuil de puissance pour une exécution du GRASP×ELS

Début

1. $pow \coloneqq initPow$;
2. **POUR** $pop \coloneqq 0$ **A** m_pop **FAIRE**
3. $S^* \coloneqq \emptyset$;
4. **POUR** $g \coloneqq 1$ **A** nb_g **FAIRE**
5. $S \coloneqq$ **Phase_de_Construction** ;// construction d'une solution évaluée avec la puissance *pow*
6. $S \coloneqq$ **Phase_de_Recherche_Locale** ; // amélioration de la solution
7. **SI** ($S^* == \emptyset$ **OU** $f(S)$ est meilleur que $f(S^*)$) **ALORS** $S^* \coloneqq S$ **FIN SI**
8. **POUR** $els \coloneqq 1$ **A** nb_els **FAIRE**
9. $nS^* \coloneqq \emptyset$;
10. **POUR** $n \coloneqq 1$ **A** nb_n **FAIRE**
11. $nS \coloneqq$ voisin de S ; // permutation de deux opérations dans le vecteur par répétition
12. $nS \coloneqq$ **Phase_de_Recherche_Locale** ;
13. **SI**($f(nS)$ est meilleur que $f(nS^*)$) **ALORS** $nS^* \coloneqq nS$ **FIN SI**
14. **FIN POUR**
15. $S \coloneqq nS^*$;
16. **SI** ($f(nS^*)$ est meilleur que $f(S^*)$) **ALORS** $S^* \coloneqq nS^*$ **FIN SI**
17. **FIN POUR**
18. **FIN POUR**
19. Insérer S^* dans $pFront$; // insertion de S* et suppression des solutions dominées si existantes
20. $pow \coloneqq pow + step$; // passage au prochain seuil de puissance
21. **FIN POUR**
22. Retourner $pFront$;

FIN

4.2.3. NSGA-II hybride

Cette section décrit les composants requis pour élaborer une autre approche reposant sur une métaheuristique appropriée pour le problème Bi-JSPPR. Cette approche est basée sur un NSGA-II hybridé avec une procédure de recherche locale (hNSGA-II).

Le NSGA-II a été proposé par (Deb et al., 2002). Ce qui a rendu le NSGA-II populaire est sa facilité de construction à partir d'un algorithme génétique ; il a été appliqué avec succès à de nombreux

problèmes combinatoires multiobjectifs (Ahmadi et al., 2016 ; Lin and Yeh, 2012 ; Liu et al., 2014). L'objectif du NSGA-II est de retourner un ensemble de solutions non dominées, le plus proche possible de la frontière Pareto-optimale. À chaque itération, l'algorithme tend à améliorer ce front en utilisant les principes des algorithmes génétiques (croisements et mutations). Comme il a été montré dans le premier chapitre (§5.2.4.8), deux opérateurs principaux composent le NSGA-II : la procédure de tri non-dominé et la procédure de sélection basée sur un tournoi faisant intervenir le rang et la marge d'une solution.

Dans le NSGA-II, de nouvelles solutions sont générées en sélectionnant des solutions parentes et en appliquant deux opérateurs classiques des algorithmes évolutionnaires : le croisement et la mutation. Les parents sont sélectionnés selon un tournoi binaire. Compte tenu de deux solutions parentes choisies aléatoirement, celle qui appartient au front de rang le plus faible est gardée. Cependant, dans le NSGA-II, lorsque deux parents appartiennent à un même front, le plus isolé est conservé. Le degré d'isolement d'une solution est donné en calculant la distance entre les individus à gauche et à droite de la solution dans le front. Cette distance est également appelée *marge*, et elle représente donc une mesure de l'espace de recherche autour de la solution qui n'est occupé par aucune autre solution de la population (Lacomme et al., 2006). Les deux points à l'extrémité d'un front ont une marge infinie. Le calcul de la marge est donné dans le chapitre I (§5.2.4.8) lors de la présentation du NSGA-II.

Trois procédures importantes liées aux algorithmes génétiques doivent être définies pour la réalisation d'une métaheuristique de type NSGA-II : la génération de la première population, le croisement et la mutation. Chacune de ces procédures est présentée dans les paragraphes suivants.

4.2.3.1. Initialiser la population

La première étape d'une métaheuristique à population telle que le NSGA-II consiste à générer le pool de solutions initiales. Pour le problème étudié, les individus de la population sont générés de manière aléatoire afin d'apporter le plus de diversité possible à la population. Par conséquent, chaque individu est associé à un vecteur par répétition aléatoire. Le problème faisant intervenir la notion de seuil de puissance, chaque solution est également dotée d'une puissance maximale autorisée pour être évaluée.

4.2.3.2. Génération de nouvelles solutions

Dans les algorithmes évolutionnaires, deux opérateurs importants sont utilisés pour générer les descendants et diversifier la population : le croisement et la mutation. Le principe de ces opérateurs a été décrit dans l'algorithme mémétique de la section précédente. Deux opérateurs différents sont proposés pour le NSGA-II.

(I) Opérateur de croisement

Un opérateur de croisement basé sur le croisement à point unique (one-point crossover (Deb and Agrawal, 1995)) est proposé. Il est basé sur les occurrences des jobs dans le vecteur par répétition. Ce type de croisement permet d'éviter l'apparition de chromosomes illégaux, de la même manière que l'opérateur utilisé dans le MA à la section précédente. Si l'on considère deux parents P_1 et P_2, l'opérateur génère deux fils C_1 et C_2 par application de la procédure suivante :

1. Un point X est généré aléatoirement sur les deux chromosomes parents P_1 et P_2. La position de ce point est la même chez les deux chromosomes.

2. Les occurrences des jobs depuis le début de P_1 (resp. P_2) jusqu'à X sont copiées dans C_1 (C_2).

3. C_1 (C_2) est complété après X en lisant P_2 (P_1) de gauche à droite et en ajoutant chaque occurrence d'un job si l'occurrence correspondante n'est pas déjà présente dans C_1 (C_2).

Par exemple, dans un graphe disjonctif non orienté d'un problème de Job-shop dont les données sont les mêmes que celles présentés dans la Figure 17 (i.e. un graphe sans arcs orientés autres que les arcs conjonctifs et les arcs de Time-lags), [321132321312] et [121133322132] peuvent être des chromosomes réalisables pour deux parents. Si X est généré en 5ème position, deux fils sont construits comme dans la Figure 18.

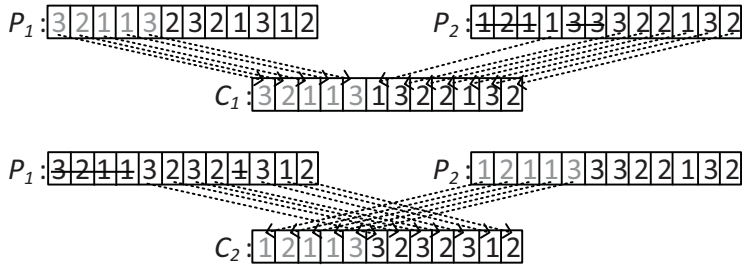

Figure 18 Exemple de construction de deux solutions à partir de deux parents

(II) Opérateur de mutation

Dans ce travail, l'opérateur de mutation est basé sur une permutation semblable à celle du MA (chapitre III, §3.3.4.3). Pour cet opérateur, deux gènes arbitraires différents sont choisis et leurs valeurs sont échangées. Par exemple, dans la solution C_1 précédemment générée lors de la procédure de croisement (i.e. [321131322132]), une mutation peut consister à permuter les jobs en 4ème et 11ème positions, ce qui entraîne le nouvel fils [321331322112]. Cet opérateur est utilisé pour générer des solutions voisines dans le iGRASP×ELS également (Algorithme 6, ligne 11). Cependant, l'opérateur de mutation pour le NSGA-II diffère légèrement de celui utilisé dans le GRASP×ELS. En effet, chaque parent a un seuil de puissance spécifique. Pour l'opérateur de mutation dans le hNSGA-II, le seuil de puissance autorisé pour le fils généré est choisi aléatoirement dans l'intervalle construit à partir des seuils de puissance des parents. Ce seuil de puissance autorisé est utilisé lors de l'évaluation (*max*P) et également dans la procédure de recherche locale adaptée au NSGA-II qui est présentée dans la suite.

Dans l'optimisation mono-objectif, un algorithme génétique standard doit être hybridé avec une procédure de recherche locale pour pouvoir concurrencer les métaheuristiques telle que la recherche tabou (Lacomme et al., 2006). Cependant, la recherche locale ne doit pas conduire à une convergence prématurée des solutions vers un seul point, ou zone, de l'espace des solutions, et ainsi perturber le mécanisme de recherche de l'algorithme génétique multiobjectif. La section suivante présente les mouvements et la structure générale de la procédure de recherche locale proposée pour le problème qui prend en compte les deux critères.

4.2.3.3. Procédure de recherche locale pour le hNSGA-II

La génération de solutions, mais aussi les opérateurs de croisement et de mutation au sein d'une métaheuristique multiobjectif, produisent rarement des solutions Pareto-optimales (Lacomme et al., 2006). Ainsi, il est utile d'explorer l'espace des solutions afin d'en obtenir de meilleures. Dans sa formulation de base, le NSGA-II n'intègre pas d'algorithme de recherche locale. Cette procédure vise à améliorer la qualité des solutions tout en gardant le front de Pareto bien dispersé. La recherche locale est ajoutée entre les lignes 5 et 6 dans l'algorithme présenté dans le chapitre I (§5.2.4.8), juste après les opérateurs de croisement et de mutation. Ce faisant, un NSGA-II hybride est développé (hNSGA-II). La phase de recherche locale consiste à appliquer la forme simplifiée de l'Algorithme 4 sur une solution en considérant plusieurs seuils de puissance afin d'explorer le voisinage de la solution actuelle et d'obtenir plusieurs points dans l'espace des solutions pour concevoir un meilleur front de Pareto. Pour ce faire, le seuil de puissance évolue par palier dans la recherche locale. Chaque palier est calculé en considérant un ratio donné entre la puissance minimale et maximale requise par les solutions dans le front de Pareto actuel. La structure de la procédure de recherche locale est donnée dans l'Algorithme 7. A partir d'une solution donnée, une population appelée $paretoLS$ de solutions non dominées est construite. La taille de la population est égale au nombre maximal $nbStep$. À la ligne 1, le pas de puissance $stepP$ est calculé selon la valeur $nbStep$ en considérant les consommations en puissance des points extrêmes dans le front de Pareto actuel. Ensuite, le seuil de puissance Pow est initialisé à une valeur inférieure à la puissance associée à la solution durant la phase de mutation. Pow est obligatoirement supérieur au minimum autorisé pour le seuil de puissance. L'objectif consiste à explorer le voisinage d'une solution comme le montre la Figure 19, où un exemple schématique des solutions améliorées obtenues après le processus de recherche locale est donné. Compte tenu du seuil de puissance (Pow), la solution est améliorée en effectuant des permutations sur les opérations en disjonction sur un chemin critique en utilisant la procédure de recherche locale simplifiée. À la ligne 8, le seuil de puissance Pow est mis à jour en fonction du pas de puissance calculé à la ligne 1 et Pow est alors utilisé comme nouveau seuil de puissance pour l'évaluation des solutions. Cette procédure est appliquée à chaque solution obtenue après un croisement et une mutation. Après cela, la population Pop du NSGA-II est mise à jour en appliquant la procédure de *tri non-dominé* et la procédure de *marge* sur la population initiale fusionnée avec toutes les solutions obtenues après le processus de recherche locale. Les meilleures solutions sont considérées comme étant la nouvelle population, et le processus général est réappliqué jusqu'à atteindre un critère d'arrêt.

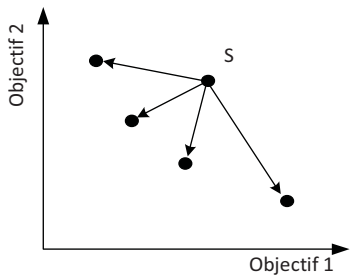

Figure 19 Solutions obtenues après recherche locale sur la solution S

Algorithme 7 : Recherche_Locale_hNSGA-II

Entrée

S	Solution dont le voisinage doit être exploré
$maxP$	Puissance maximale observable dans le front de Pareto actuel
$minP$	Puissance minimale observable dans le front de Pareto actuel
$minPow$	Puissance minimale en dessous de laquelle aucune solution n'est valide
$nbStep$	Nombre de solutions à générer
$maxIter$	Nombre maximal d'appels à la procédure de recherche locale

Sortie

$paretoLS$	Solutions non dominées obtenues à partir de S

Variables

$tmpS$, $savS$	Solutions temporaires
pow	Puissance actuelle utilisée par la recherche locale
$stepP$	Pas permettant de mettre à jour la puissance pow à chaque itération de l'algorithme
$cptIter$	Compteur

Début

1. $cptIter \coloneqq 0$; $paretoLS \coloneqq \emptyset$; $stepP \coloneqq \frac{maxP - minP}{nbStep}$; // initialisation des variables et du front
2. $pow \coloneqq max(minPow;\ S.power - (\frac{maxIter}{2}) * stepP)$; // *power* est donné par la mutation
3. **TANT QUE** $cptIter < maxIter$ **FAIRE**
4. $tmpS \coloneqq S$;
5. Evaluer la solution $tmpS$ avec le seuil de puissance pow ;
6. Appliquer la recherche locale avec $tmpS$ et pow ;
7. Ajouter $tmpS$ dans $paretoLS$; Eliminer les solutions dominées ;
8. $pow \coloneqq pow + stepP$; //on passe à la prochaine puissance
9. **FIN TQ**
10. Retourner $paretoLS$;

Fin

4.3. Expérimentations numériques

4.3.1. Instances

Tout d'abord, la génération d'instances est nécessaire car aucun jeu de données de taille raisonnable n'existe dans la littérature pour le problème considéré. Un ensemble de 40 instances de référence pour le problème de Job-shop classique conçues par Lawrence (Lawrence, 1984) est disponible dans la bibliothèque OR-library. Toutefois, ces instances ne tiennent pas compte des besoins en puissance des opérations. Par conséquent, cet ensemble de données est adapté au nouveau problème d'ordonnancement en introduisant des facteurs supplémentaires tels que la durée, la consommation du pic de puissance et la consommation nominale de puissance après la fin du pic.

Les temps de traitement d'origine des opérations sont gardés identiques et sont utilisés comme temps de traitement total des opérations. Les données de puissance pour ces instances ont été générées aléatoirement en considérant que, compte tenu de la durée totale d'une opération P_i, la consommation en puissance nominale $PC_{i,2}$ et la consommation de puissance de crête $PC_{i,1}$ sont exprimées comme suit : $PC_{i,2}$ suit une loi uniforme sur l'intervalle fermé $[0\ ;\ P_i/2]$. $PC_{i,1}$ suit une loi uniforme sur l'intervalle fermé $[PC_{i,2}\ ;\ P_i]$. La durée du pic de puissance maximal ($P_{i,1}$) est choisie aléatoirement dans

l'intervalle [0 ; $P_i/3$]. Concernant le nombre d'opérations considérées dans ces instances, il varie entre 50 et 300 ce qui permet de manipuler des instances de taille moyenne, plus importantes que dans le problème à seuil variable.

4.3.2. Mesures de performance adoptées

Pour évaluer la qualité d'un algorithme d'optimisation multiobjectif, différentes mesures peuvent être considérées, et le seul critère utilisé dans l'optimisation mono-objectif n'est pas suffisant, comme l'ont souligné (Zhang and Chiong, 2016). Dans leur travail, trois mesures sont utilisées : NS, OD et DS (Tan et al., 2006). Ces mesures sont décrites dans le chapitre I (§5.3.2) et sont rappelées ci-dessous :

(1) Le nombre de solutions non dominées (NS) trouvées par une métaheuristique. Ce critère est important car un grand nombre de solutions dans le front de Pareto suggère qu'un décideur a plusieurs choix possibles.

(2) Le degré d'optimalité de Pareto (OD) des solutions obtenues qui indique le nombre de solutions qui ne sont dominées par aucune autre solution obtenue lors d'une exécution d'un autre algorithme.

(3) La répartition des solutions le long du front de Pareto (DS). Une plus petite valeur de DS suggère que les solutions sont réparties plus uniformément dans le front de Pareto. Cela indique notamment qu'il y a entre deux solutions éloignées du front un ensemble de solutions potentiellement acceptables par un décideur. En d'autres termes, un DS faible correspond à une meilleure répartition des solutions au sein du front et empêche la formation de groupements éloignés entre eux sur le front comme le montre la Figure 20.

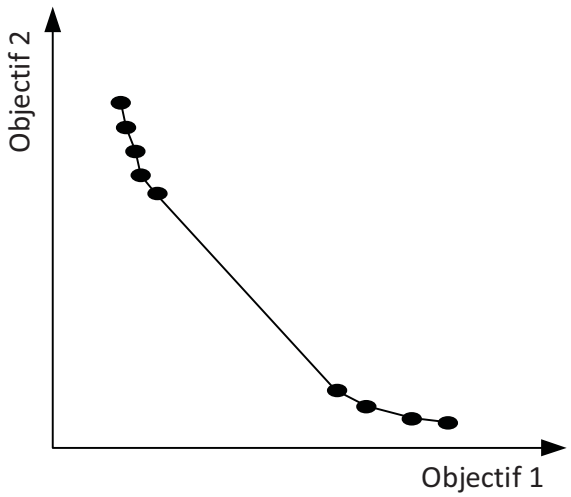

Figure 20 Exemple d'un front de Pareto avec des solutions mal réparties

4.3.3. Réglage des paramètres

La métaheuristique hNSGA-II est comparée à la version adaptée du GRASP×ELS appelée iGRASP×ELS. L'objectif est d'avoir de bons points de comparaison du front de Pareto obtenu avec le hNSGA-II. Pour chaque cas, et pour chaque métaheuristique, 5 réplications ont été réalisées. Le temps de calcul maximum est limité à 1000s pour les deux métaheuristiques.

Les valeurs de paramètres pour hNSGA-II ont été définies comme suit : la taille de la population, n, est fixée à 50 ; Le critère d'arrêt est un nombre d'itérations $iter\_max$ fixé à 50 ; La probabilité de croisement cp est choisie à 0,8 ; la probabilité de mutation mp est fixée à 0,2. Ces paramètres ont été obtenus après un plan d'expérience consistant à tester différents paramètres sur un sous-ensemble de toutes les instances. Ce sous-ensemble se compose de 8 instances avec un nombre différent de jobs ou de machines. Les paramètres testés sont les suivants : $n = \{50 ; 100\}$, $iter\_max = \{25 ; 50 ; 75\}$, $cp = \{0{,}6 ; 0{,}7 ; 0{,}8 ; 0{,}9\}$, $mp = \{0{,}1 ; 0{,}2 ; 0{,}3\}$ résultant en 72 configurations possibles. Les valeurs moyennes de NS, OD et DS sont calculées pour chacune des instances du sous-ensemble et la configuration avec les meilleures performances globales pour le hNSGA-II est sélectionnée pour toutes les autres instances. Pour la recherche locale, le nombre de pas ($nbStep$) est expérimentalement fixé à 40, tandis que le nombre maximal de solutions explorées ($maxIter$) est fixé à 20 pour éviter un temps de calcul trop important.

Pour le iGRASP×ELS, la même approche est utilisée avec les paramètres testés comme suit : le nombre d'itérations GRASP (nb_g) est expérimenté avec 5 et 10 itérations. Le nombre d'itérations ELS (nb_els) varie entre 75 et 150 avec un pas de 25 et le nombre de voisins générés (nb_n) est compris entre 20 et 25 avec un pas de 1. Ce plan d'expérience moins important que dans la précédente section se justifie par le nombre de fois que le GRASP×ELS va être appelé pour produire le front de Pareto. Comme cette métaheuristique n'est pas initialement dédiée à des problèmes biobjectifs, elle est appliquée itérativement avec différents seuils de puissance commençant par le plus bas possible. Un compromis doit donc être fait sur la durée globale permettant l'obtention du front. Aussi, pour éviter des temps de calcul importants, une limite de temps pour chaque GRASP×ELS est fixée à 50 secondes. Le plan d'expérience a abouti aux paramètres suivants : $nb\_g = 10$, $nb\_els = 100$ et $nb\_n = 23$. L'algorithme s'arrête après avoir trouvé 30 solutions différentes (i.e. $paretoMaxSize$). Comme certaines de ces solutions peuvent être dominées (lorsque le GRASP×ELS est incapable de trouver une bonne solution même si le seuil de puissance a été augmenté), les 30 solutions sont vérifiées et réorganisées dans un front de Pareto définitif.

4.3.4. Résultats

Tableau 5 Récapitulatif des notations principales utilisées

AVG_NS	Nombre moyen de solutions dans les fronts générés
AVG_DS	Répartition moyenne des solutions le long des fronts de Pareto
AVG_OD	Degré moyen d'optimalité des fronts de Pareto.

Les résultats sont présentés dans le Tableau 6 où la colonne **Ins** représente l'instance traitée. Afin de donner une indication sur la taille des instances traitées, le nombre de jobs est donné et noté **nbjob**, le nombre de machines est noté **nbmac**, **n** est le nombre d'opérations. **AVG_NS** représente la valeur moyenne pour NS ; **AVG_DS** est la valeur moyenne pour DS et **AVG_OD** est la valeur moyenne pour OD. Les résultats sont analysés en trois temps, en fonction des critères observés.

(1) En observant la colonne **AVG_NS**, il est possible de voir que le hNSGA-II est capable de trouver un plus grand nombre de solutions dans le front de Pareto que le iGRASP×ELS. Par exemple, dans la04_jsppr, le hNSGA-II trouve en moyenne 40.4 solutions dans le front de Pareto alors que le iGRASP×ELS trouve 22.6 solutions. De plus, après l'instance la15_jsppr, le nombre de solutions dans le front de Pareto retourné par le hNSGA-II est égal à la taille de la population. Ceci est dû à la procédure de recherche locale qui introduit des solutions améliorées dans la population. Parce que la taille de cette population est fixée à 50, et en raison de la diversité des solutions, toutes les solutions finales font partie du front de Pareto. Cela montre que la procédure de recherche locale améliore considérablement la qualité du front de Pareto et peut fournir de nombreuses solutions à un décideur potentiel. Si l'on peut noter que l'augmentation du nombre d'itérations du iGRASP×ELS, c'est-à-dire l'utilisation de plus de seuils de puissance, produit une plus grande quantité de solutions dans le front de Pareto, ceci a un impact énorme en temps de calcul car chaque itération de l'iGRASP×ELS a besoin de temps pour obtenir une seule solution du front. Notons que le compromis entre le temps de calcul et la qualité des solutions est un facteur important dans les problèmes d'optimisation, en particulier lorsque la taille des problèmes augmente. Dans l'approche actuelle, l'iGRASP×ELS nécessite déjà en moyenne le double de la durée du hNSGA-II, ce qui démontre la performance de cette dernière métaheuristique.

(2) Sur la colonne **AVG_OD**, il apparaît que les solutions trouvées par le hNSGA-II sont de meilleure qualité que les solutions obtenues par le iGRASP×ELS. Rappelons que ce critère correspond au pourcentage de solutions qui ne sont dominées par aucune solution issue de toute exécution d'une autre métaheuristique. Les fronts de Pareto sont meilleurs dans plus de 75% des cas avec la hNSGA-II. De plus, en moyenne, plus de 78% des solutions du iGRASP×ELS sont dominées par au moins une des solutions fournies par le hNSGA-II. A l'inverse, seulement 46% des solutions du hNSGA-II sont dominées par au moins une des solutions du iGRASP×ELS. Cependant, le iGRASP×ELS est capable de fournir de meilleurs fronts de Pareto que le hNSGA-II pour certaines instances (e.g. la12_jsppr, la16_jsppr). En outre, le critère AVG_RN, considéré comme un résultat agrégé, ne montre pas que le iGRASP×ELS fonctionne mieux lorsque le seuil de puissance est élevé, alors que le hNSGA-II a des difficultés à trouver des solutions dans ce cas. Ceci est présenté par la Figure 21 et la Figure 22. Cela montre que l'iGRASP×ELS, et donc le GRASP×ELS, peut être utilisé lors de la recherche de solutions ayant un makespan faible lorsque les seuils de puissance sont élevés.

(3) Sur la colonne **AVG_DS**, il semble que les solutions obtenues avec le hNSGA-II soient réparties plus uniformément que les solutions trouvées par le iGRASP×ELS. La valeur DS moyenne correspondant à iGRASP×ELS est supérieure à celle du hNSGA-II (0.82 par rapport à 0.70). En fait, le hNSGA-II offre une meilleure dispersion des solutions dans le front de Pareto dans plus de 70% de toutes les instances. Ceci est encore dû à la recherche locale et à la procédure de mutation qui introduit de nouveaux points dans le front de Pareto. Compte tenu de la valeur AVG_OD, ce critère suggère que le hNSGA-II est en moyenne supérieur au iGRASP×ELS.

Bien que le hNSGA-II surpasse le iGRASP×ELS dans la plupart des cas, cette dernière métaheuristique semble plus efficace dans certaines situations (instances : la12_jsppr, la15_jsppr). Cependant, le iGRASP×ELS possède plusieurs avantages qui ne sont pas visibles dans les résultats exposés ci-dessus. En effet, dans presque tous les cas, la métaheuristique présente de meilleurs résultats si l'on considère un seuil de puissance élevé comme souligné par la Figure 22. De plus, le GRASP×ELS est très utile lorsqu'une solution unique est nécessaire en fonction d'un seuil de puissance donné. Par exemple, si un fournisseur d'électricité informe un client que la puissance disponible pendant une période donnée est réduite à un niveau inférieur à celle contractualisée (en raison de pannes ou de

tâches de maintenance), il appartient à l'entreprise d'ajuster l'ordonnancement de ses ateliers en fonction du nouveau seuil de puissance. Dans un tel scénario, il est possible d'utiliser le GRASP×ELS utilisé dans le iGRASP×ELS car son schéma d'intensification est mieux adapté que le hNSGA-II, qui recherche un front complet de Pareto. De plus les temps de calcul d'une seule exécution du GRASP×ELS peuvent être relevés, conduisant à de meilleures solutions. En outre, l'iGRASP×ELS nécessite moins d'espace mémoire, car aucune population n'est utilisée pendant les phases de recherches locales, ce qui pourrait être utile pour traiter des instances de très grande taille. Il est donc possible avec le GRASP×ELS, en en modifiant les paramètres, d'obtenir de meilleures solutions lorsque le seuil de puissance est une contrainte donnée. C'est par exemple le cas dans la section précédente, où le GRASP×ELS présente des résultats légèrement meilleurs que l'algorithme mémétique dont la structure est proche du NSGA-II.

Tableau 6 Résultats obtenus avec le hNSGA-II et le iGRASP×ELS

				hNSGA-II			iGRASP×ELS		
Ins.	nbjob	nbmac	n	AVG_NS	AVG_DS	AVG_OD	AVG_NS	AVG_DS	AVG_OD
la01_jsppr	10	5	50	35.6	0.98	0.16	20.2	0.65	0.27
la02_jsppr	10	5	50	25.6	0.79	0.06	21	0.88	0.65
la03_jsppr	10	5	50	27	0.80	0.09	16.6	0.74	0.48
la04_jsppr	10	5	50	40.4	0.86	0.11	22.6	1.10	0.58
la05_jsppr	10	5	50	9.4	0.42	0.47	6	0.61	0.13
la06_jsppr	15	5	75	32.8	0.70	0.05	17	0.77	0.66
la07_jsppr	15	5	75	29	0.90	0.35	15.2	0.65	0.11
la08_jsppr	15	5	75	33	0.93	0.18	17	0.79	0.19
la09_jsppr	15	5	75	19.4	0.78	0.70	11.4	0.53	0.05
la10_jsppr	15	5	75	26.6	0.75	0.48	15.2	0.58	0.11
la11_jsppr	20	5	100	39	0.80	0.50	21.6	0.68	0.13
la12_jsppr	20	5	100	25	0.69	0.19	14.2	0.61	0.39
la13_jsppr	20	5	100	35.4	0.77	0.19	18.6	0.56	0.19
la14_jsppr	20	5	100	39.2	0.73	0.26	20.6	0.55	0.24
la15_jsppr	20	5	100	26.8	0.81	0.08	17.4	0.65	0.47
la16_jsppr	10	10	100	50	0.69	0.30	17.4	0.70	0.53
la17_jsppr	10	10	100	50	0.65	0.58	21.8	0.80	0.29
la18_jsppr	10	10	100	50	0.67	0.51	20.6	0.88	0.25
la19_jsppr	10	10	100	50	0.72	0.55	21.6	0.86	0.13
la20_jsppr	10	10	100	50	0.67	0.65	20.2	0.85	0.24
la21_jsppr	15	10	150	50	0.76	0.68	22.4	0.96	0.11
la22_jsppr	15	10	150	50	0.68	0.50	23.8	0.79	0.17
la23_jsppr	15	10	150	50	0.65	0.62	26.4	0.96	0.26
la24_jsppr	15	10	150	50	0.56	0.66	23	0.87	0.27
la25_jsppr	15	10	150	50	0.70	0.93	21.6	0.89	0.00
la26_jsppr	20	10	200	50	0.69	0.83	22.2	0.93	0.07
la27_jsppr	20	10	200	50	0.67	0.62	23.4	0.90	0.22
la28_jsppr	20	10	200	50	0.63	0.67	24.4	0.94	0.24
la29_jsppr	20	10	200	50	0.66	0.79	24	0.93	0.24
la30_jsppr	20	10	200	50	0.67	0.61	26.6	0.97	0.24
la31_jsppr	30	10	300	50	0.62	0.89	25	0.82	0.05
la32_jsppr	30	10	300	50	0.69	0.78	28.6	1.08	0.06
la33_jsppr	30	10	300	50	0.61	0.78	25.6	0.94	0.29
la34_jsppr	30	10	300	50	0.62	0.88	26.2	1.11	0.10
la35_jsppr	30	10	300	50	0.65	0.68	25.6	0.89	0.32
la36_jsppr	15	15	225	50	0.59	0.93	24	0.88	0.00
la37_jsppr	15	15	225	50	0.69	0.70	23.6	0.92	0.10
la38_jsppr	15	15	225	50	0.57	0.93	22.2	0.95	0.00
la39_jsppr	15	15	225	50	0.61	0.80	22.6	0.94	0.04
la40_jsppr	15	15	225	50	0.73	0.82	23.6	0.94	0.09
Moyenne :				42.36	0.70	0.54	21.025	0.83	0.22

Un exemple du front moyen de Pareto obtenu avec une exécution typique des deux algorithmes pour l'instance la24_jsppr est donné sur la Figure 21(1) où il apparaît qu'aucune des métaheuristiques

ne surpasse nettement l'autre. En effet, si l'iGRASP×ELS trouve de meilleures solutions moyennes lorsque le seuil de puissance est élevé (partie A de la figure), le hNSGA-II trouve lui de meilleures solutions moyennes dans la deuxième partie (partie B). En outre, tous les fronts de Pareto obtenus sur cette instance sont présentés sur la Figure 21(2) qui montre que l'iGRASP×ELS peut obtenir des fronts de Pareto de bonne qualité en moyenne, mais également que le hNSGA-II présente plus de solutions non dominées par rapport aux différentes exécutions même si la distance entre les fronts n'est pas très importante. Pour conclure cette expérience, la Figure 22 résume les résultats pour le critère AVG_OD. Dans cette figure, les rectangles correspondent à la valeur moyenne de AVG_OD, et les barres d'erreur correspondent à l'écart type. Cette dernière figure montre que le hNSGA-II est nettement meilleur sur l'ensemble des données en moyenne.

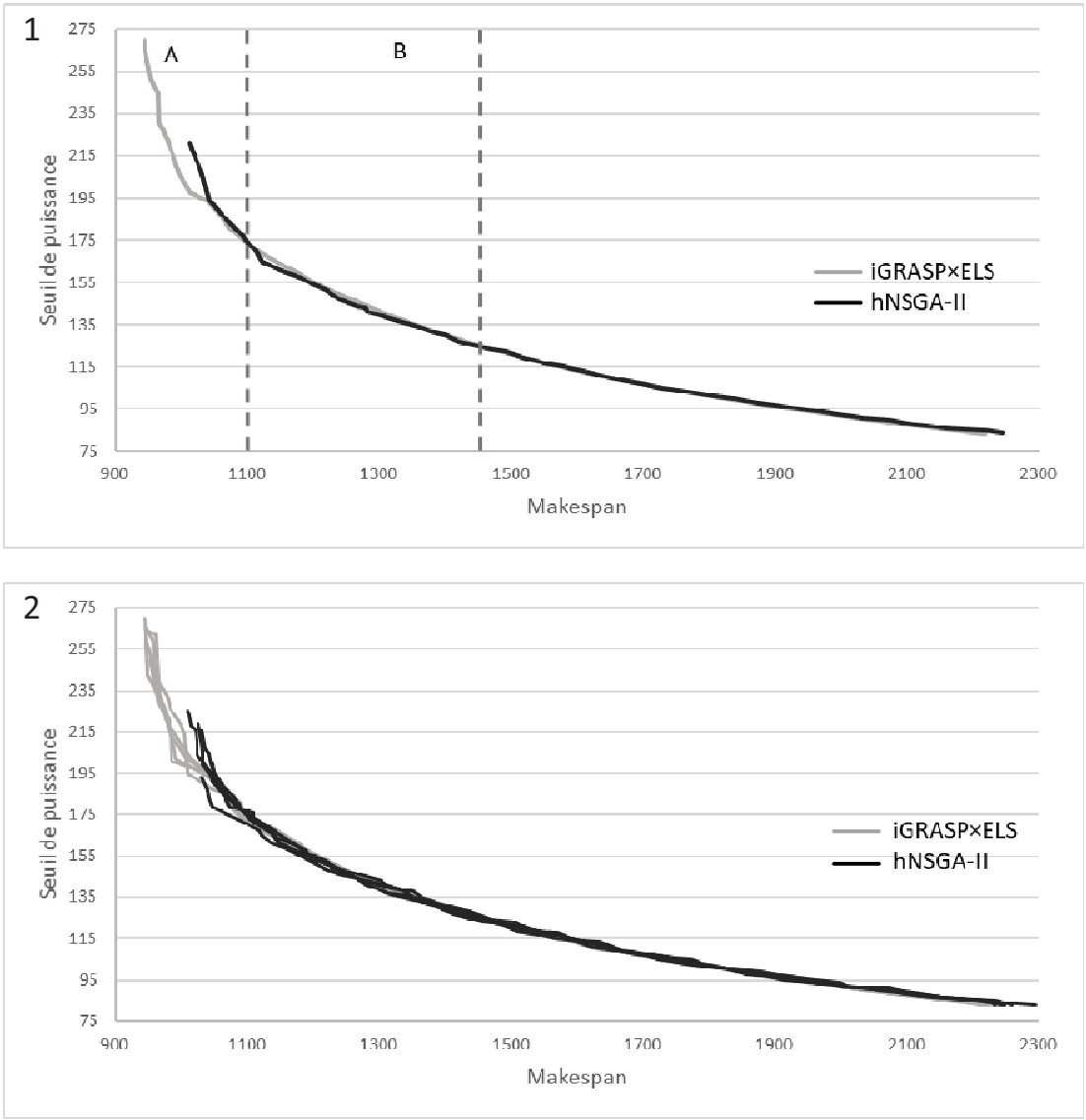

Figure 21 Front de Pareto moyen (1) et fronts de Pareto obtenus après 5 réplications (2) de chaque métaheuristique sur la24_jsppr

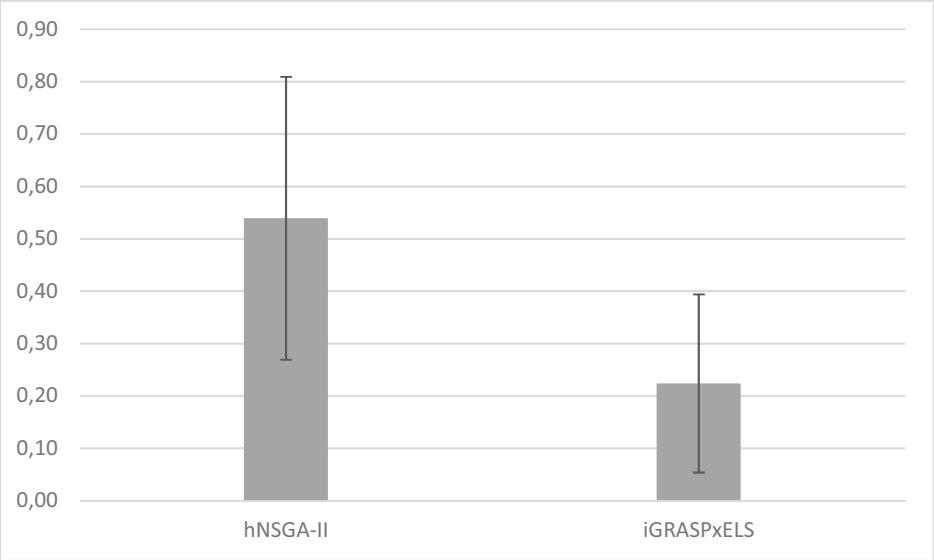

Figure 22 Valeurs moyennes pour le critère OD (rectangles gris) et écart type (barres d'erreurs)

4.4. Conclusion

Dans cette section, une approche biobjectif pour l'ordonnancement d'ateliers avec la considération d'un seuil de puissance a été étudiée. Deux objectifs sont optimisés : le makespan et le seuil de puissance. L'optimisation du seuil de puissance est un objectif important, en particulier dans les entreprises où les machines ne peuvent pas être arrêtées, car peu d'améliorations peuvent être réalisées en ce qui concerne les consommations totales d'énergie dans un tel environnement. Dans le problème étudié dans ce travail, chaque opération présente deux types de puissance pour représenter plus précisément les profils de puissance que l'on peut observer dans le monde réel. Les instances générées représentent des systèmes de production, où jusqu'à 300 opérations doivent être planifiées. Il est difficile d'obtenir des solutions optimales à l'aide d'un solveur linéaire sur ces instances, et l'obtention d'un front de Pareto avec une telle approche nécessiterait beaucoup de temps. Par conséquent, deux métaheuristiques sont proposées : un NSGA-II hybride (hNSGA-II) et un GRASP×ELS itéré (iGRASP×ELS). Le but de ces métaheuristiques est d'obtenir un front de Pareto pseudo-optimal avec un effort de calcul limité. Le NSGA-II a montré son efficacité dans la recherche de bons fronts de Pareto pour des problèmes avec moins de 3 objectifs. La recherche locale ajoutée au NSGA-II est une procédure d'intensification qui permet au hNSGA-II d'obtenir de meilleurs résultats que le NSGA-II seul. Cette métaheuristique est comparée au iGRASP×ELS, basé sur le GRASP×ELS précédemment proposé pour la recherche de solutions dans le cas mono-objectif lorsque le seuil de puissance est considéré comme une donnée. Les expérimentations numériques montrent l'efficacité des approches proposées. Comme le montrent les résultats, le hNSGA-II fonctionne en général mieux que l'iGRASP×ELS, et ce dans un temps de calcul plus court. Cependant, le GRASP×ELS utilisé dans iGRASP×ELS peut également être une métaheuristique intéressante car elle fournit de meilleures solutions lorsque le seuil de puissance est élevé et peut également être utilisée si l'objectif est de trouver une solution en tenant compte d'un seuil de puissance spécifique en augmentant son temps de calcul.

5. Conclusion

Après une analyse de la littérature des problèmes avec contraintes énergétiques, il est apparu que peu de travaux traitent du problème de Job-shop et encore moins considèrent l'optimisation contrainte par un seuil de puissance disponible. Or ce problème constitue un enjeu important dans l'ordonnancement.

Dans ce chapitre, deux problèmes liés à l'ordonnancement des systèmes de production sont proposés. Ils prennent en compte des profils de puissance pour les opérations et un seuil de puissance à ne pas dépasser au cours de la production. Ce seuil modélise donc la puissance autorisée pour la réalisation des opérations au sein du système de production. Chaque opération a un profil de puissance variable en fonction de l'avancement de l'usinage. Ce profil de puissance est propre à chaque opération et est composé de deux parties distinctes : la première correspond à un pic de puissance en début de processus, tandis que la seconde partie correspond à la puissance nominale en cours d'usinage.

Dans la première partie de ce chapitre, l'attention est portée sur l'ordonnancement d'opérations ayant un profil de puissance variable dans le cas où un seuil de puissance variable est considéré. Ce seuil modélise à la fois la puissance contractualisée par l'industriel auprès de son fournisseur d'électricité, mais peut aussi représenter les chutes de puissances prévisionnelles dans le cas d'utilisation de source d'énergie renouvelables et dont la disponibilité peut varier au fil du temps. L'objectif est alors d'ordonnancer la production au mieux en considérant ce seuil de puissance. Trois métaheuristiques sont proposées pour la résolution du problème : un GRASP×ELS, un VNS et un MA. Les résultats montrent que le GRASP produit des résultats légèrement meilleurs que les autres métaheuristiques.

Dans la seconde partie de ce chapitre, deux approches biobjectifs pour obtenir un ensemble de solutions sous forme de front de Pareto sont proposées. Ici, l'objectif est de définir la puissance maximale requise pour obtenir des ordonnancements de bonne qualité. Deux métaheuristiques sont proposées pour ce problème : un GRASP×ELS itéré (iGRASP×ELS), et un NSGA-II hybride (hNSGA-II). Les résultats montrent que le NSGA-II propose de meilleurs fronts de Pareto en moyenne, et dans des temps de calcul plus compétitifs que ceux du iGRASP×ELS. Cependant ce dernier présente de meilleures solutions lorsque le seuil de puissance autorisé est élevé. Les deux approches sont donc complémentaires.

Ces travaux ouvrent la voie à d'autres perspectives. Comme souligné par (Giret et al., 2015) les articles ne prennent généralement pas en compte à la fois l'optimisation des consommations énergétiques et les émissions en sortie du système. Il est donc important de définir des approches intégrant l'ensemble de ces problématiques. Il pourrait être également intéressant d'ajouter d'autres objectifs tels que la diminution de la consommation totale en énergie ou des coûts liés à des tarifications variables. La conception d'instances qui considèrent des opérations avec des profils de puissance plus diversifiés est également une direction intéressante pour étudier l'impact de la discrétisation sur la qualité et la robustesse des solutions (Salido et al., 2016a). Au-delà de la problématique liée aux temps de calcul induits par le nombre de sous-opérations correspondant au profil de puissance (Rager et al., 2015), ces problèmes de robustesse doivent être abordés (Biel and Glock, 2016). En effet, plus un profil de puissance est précis, plus il est difficile de s'assurer que la puissance autorisée instantanément ne soit pas dépassée en raison de perturbations possibles dans les exigences en puissance d'une opération (défaut sur la pièce, sur la machine, …).

Dans le futur, l'ensemble des données créées sera étendu à des instances de plus grande taille. De plus, il est courant de trouver des procédures d'estimation pour aborder les problèmes d'optimisation afin de réduire les temps de calcul. Pour améliorer les performances de la métaheuristique sur ces cas, une procédure d'estimation incluant des considérations de puissance sera développée. Enfin, étant donné que les entreprises sont confrontées à un monde non statique, où les systèmes de fabrication sont dynamiques et où de nombreux événements aléatoires peuvent se produire, tels que les pannes de machines, l'utilisation de la simulation afin d'évaluer la robustesse des solutions pourrait également être une perspective de recherche intéressante.

Chapitre IV : Optimisation d'ateliers à cheminements multiples : le Job-shop Flexible

Ce chapitre présente le problème de Job-shop Flexible et les principales techniques utilisées pour sa résolution, telles que le graphe conjonctif-disjonctif adapté au problème, le chemin critique et les systèmes de voisinages. Ces outils sont utilisés pour définir une métaheuristique adaptée du GRASP×ELS afin de résoudre le problème.

Sommaire

Liste des figures

Liste des tableaux

Liste des algorithmes

1. Introduction

Dans ce chapitre, les techniques utilisées pour résoudre le problème de Job-shop Flexible (FJSP) sont présentées. Il s'agit d'un problème classique de la littérature, et l'un des plus généraux des problèmes d'ordonnancement d'atelier (Chaudhry and Khan, 2016). En effet, le problème de Job-shop Flexible :

- généralise de nombreux problèmes classiques d'ordonnancement comme le problème à une machine, le problème à machines parallèles, les problèmes de Flow-shop et leurs extensions Hybrides/Flexibles, ainsi que le problème de Job-shop. Ce dernier est cependant au cœur du problème de FJSP ; la différence entre les deux se trouve dans l'ajout d'un problème d'affectation des opérations aux machines par rapport au problème de Job-shop. De ce fait, un algorithme capable de résoudre un problème de Job-shop Flexible est également capable de résoudre des instances des problèmes précédemment cités.
- ne traite pas le même problème que le Resource Constrained Project Scheduling Problem (RCPSP). Ce dernier est plus généralement répandu dans le contexte de l'ordonnancement de projet(s). En effet, s'il est possible de représenter un problème de Job-shop Flexible à l'aide d'un problème de RCPSP, ce dernier peut prendre en compte, dans sa formalisation, des contraintes de précédences plus complexes entre les tâches. Ainsi, si les algorithmes de RCPSP peuvent résoudre les problèmes de FJSP, ils sont plus généraux et par conséquent moins adaptés aux problèmes d'ateliers à cheminements multiples.

La suite de cette section est illustrée par un exemple de problème de Job-shop Flexible. Ce problème est composé de trois jobs. Le premier comporte trois opérations, le second en comporte quatre, et le troisième en comporte deux. Chaque opération est à réaliser sur une machine issue d'un ensemble de machines compatibles. Il y a ici quatre machines. Les temps de traitement dépendent de la machine choisie pour effectuer l'opération. Les couples (machine ; durée) sont donnés dans le Tableau 1 ci-dessous. L'objectif du problème est de trouver une affectation et un ordonnancement minimisant la durée totale de traitement de toutes les opérations (i.e. makespan).

Tableau 1 Instance exemple pour le Job-shop Flexible (P1)

Job 1	$O_1=(M_1 ; 4)(M_2 ; 7)$	$O_2=(M_3 ; 2)$	$O_3=(M_1 ; 8)(M_4 ; 2)$	
Job 2	$O_4=(M_3 ; 1)$	$O_5=(M_1 ; 8)(M_4 ; 4)$	$O_6=(M_1 ; 9)(M_2 ; 5)(M_3 ; 7)$	$O_7=(M_2 ; 4)(M_4 ; 7)$
Job 3	$O_8=(M_2 ; 5)$	$O_9=(M_3 ; 14)(M_1 ; 12)$		

Il est possible d'extraire à partir des données du problème une borne inférieure et une borne supérieure. La première est égale à la durée la plus courte permettant de traiter le job le plus long. Dans l'exemple présenté ici, une borne inférieure du problème vaut 17. Cette valeur correspond à la somme des durées de traitement du Job 3, lorsque les machines les plus performantes sont affectées aux opérations. Ainsi, une solution qui vaudrait 17 serait optimale. Il est également intéressant de noter que, bien que quatre machines soient disponibles, elles ne sont pas toutes adaptées à l'ensemble des opérations. On fait la distinction entre deux types de problèmes de Job-shop Flexible (Kacem et al.,

2002) : les problèmes totalement flexibles, où chaque opération peut être effectuée sur n'importe quelle machine de l'atelier ; et les problèmes partiellement flexibles, où seulement certaines machines sont disponibles pour effectuer une opération.

Ce chapitre est composé de cinq parties. La suivante présente le problème de Job-shop Flexible. La troisième section présente la représentation permettant de coder l'affectation des opérations aux machines. Un état de l'art du problème de job-shop Flexible est ensuite proposé (section 4). Cette section présente plus particulièrement le graphe conjonctif-disjonctif, un exemple de voisinage, et différentes méthodes utilisées pour résoudre le problème. La partie suivante (section 5) consiste en la présentation d'une métaheuristique pouvant être appliquée sur le problème de Job-shop Flexible. Des résultats obtenus sur des instances classiques de la littérature sont fournis. La dernière section consiste en une conclusion.

2. Définition du problème de Job-shop Flexible

Les notations suivantes sont introduites pour décrire le problème ; elles sont inspirées du travail de (Brucker and Schlie, 1990). Soit un ensemble J de r jobs $J_1, J_2, \dots J_r$ et M un ensemble de m machines $M_1, M_2, \dots M_m$. Chaque job J_i est composé d'un ensemble de n_i opérations, notées $O_{i,1}, O_{i,2}, \dots O_{i,n_i}$. La taille d'une instance est notée n, et on a $n = \sum_{i\in[1,r]} n_i$. Une opération ne peut appartenir qu'à un seul job. Pour chaque opération $O_{i,j}$, il y a un ensemble $M_{i,j} \subseteq \{M_1, \dots, M_m\}$ de machines compatibles pouvant être assignées à l'opération. En fonction de la machine v sélectionnée, un temps de traitement est défini et est noté $P_{i,j,v}$. Les opérations constituent la gamme opératoire d'un job, ainsi elles doivent être exécutées dans un ordre prédéterminé, et aucune opération ne peut être planifiée avant que l'opération précédente ne soit terminée. On note A l'ensemble des couples d'opérations $(O_{i,j}, O_{i,j+1})$ qui doivent être exécutées successivement. Une opération $O_{i,j}$ ne peut être exécutée que sur une seule machine à la fois ; et une machine ne peut traiter qu'une seule opération à la fois. Aucune préemption n'est autorisée. On note E_v l'ensemble des opérations pouvant être affectées à la machine v. Du fait de la présence d'un problème d'affectation, il n'est pas possible de déterminer à l'avance combien d'opérations seront traitées par une machine donnée (sauf dans le cas où les ensembles $M_{i,j}$ sont des singletons).

Une affectation α associe à chaque opération $O_{i,j}$ une unique machine $v = \alpha(O_{i,j}) \in M_{i,j}$. Une fois que les opérations sont affectées à une machine, elles doivent être ordonnées. Un ordonnancement μ est défini par les dates de début $s_{i,j}$ de chaque opération $O_{i,j}$. Dans la suite, et sans perte de généralité, les opérations d'un problème seront notées $\{O_i\}_{i=[1,n]}$; les autres notations sont adaptées également.

Du fait du nombre variables d'opérations d'un job à un autre, les instances communément utilisées dans la littérature ne sont pas rectangulaires, contrairement à celles définies pour le Job-shop classique. Les hypothèses suivantes sont retenues pour le problème de Job-shop Flexible :

- Les jobs sont disponibles à la date 0 ;
- Aucune indisponibilité des machines n'est considérée sur l'horizon de la planification (pas de pannes, pas de maintenance, …) ;
- Une fois l'affectation d'une opération à une machine connue, les temps de traitement sont déterministes ;
- Il n'y a pas de durées de préparation des machines (pas de montage/démontage, …) ;

- Les temps de transports entre les machines ne sont pas considérés ;
- Chaque machine ne peut réaliser à un instant donné qu'une seule opération ;
- Les zones de stockages intermédiaires sont considérées infinies ;
- Aucune ressource additionnelle n'est comprise.

Certaines variantes du problème de Job-shop Flexible consistent à supprimer certaines de ces simplifications afin de se rapprocher au plus près des systèmes réels de production.

2.1. Formalisation mathématique

Afin de décrire le problème plus précisément, la formalisation mathématique proposée par (Brucker and Schlie, 1990) est adaptée ici. Elle repose sur les notations introduites dans la section précédente.

(i) $c_i \leq c_{i+1} - p_i$;

(ii) $\sum_{v \in M_i} y_{i,v} = 1$;

(iii) $p_i = \sum_{v \in M_i} y_{i,v} P_{i,v}$;

(iv) $\alpha(O_i) \neq \alpha(O_j)\ \forall O_i \neq O_j, [c_i - p_i, c_i) \cap [c_j - p_j, c_j) \neq \emptyset$;

Dans cette formalisation, c_i est la date de fin de la $i^{ème}$ opération du problème. La dernière contrainte (iv) implique que si deux opérations dont les intervalles correspondant aux durées de traitement se chevauchent, alors les deux opérations doivent être affectées à des machines différentes. A noter que les équations (ii) et (iii) sont rajoutées, la première formalisation faite contenant des durées de traitement identiques pour un ensemble de machines pouvant traiter une opération. Dans ces contraintes, p_i désigne la durée associée à l'opération une fois qu'elle est affectée à une machine. Enfin, la variable y_{iv} vaut 1 si l'opération O_i est affectée à la machine v, 0 sinon.

L'objectif est de minimiser le makespan, c'est-à-dire de minimiser le maximum des c_i :

(v) $min(max_i(c_i))$.

2.2. Programme linéaire

La formalisation mathématique présentée ci-dessus peut être linéarisée en introduisant les variables binaires x_{ijv}. La variable x_{ijv} est définie par $x_{ijv} = 1$ si l'opération i est réalisée avant l'opération j sur la machine v, $x_{ijv} = 0$ sinon. La contrainte (iv) est alors remplacée par les contraintes suivantes :

(iv') $x_{i,j,v} + x_{j,i,v} - y_{i,v} - y_{j,v} \geq -1, \forall (i,j) \in V, \forall v \in M_i \cap M_j, J_i \neq J_j$;

(iv") $2x_{i,j,v} + 2x_{j,i,v} - y_{i,v} - y_{j,v} \leq 0, \forall (i,j) \in V, \forall v \in M_i \cap M_j, J_i \neq J_j$;

(iv"') $s_j - s_i - p_i + H(1 - x_{i,j,v}) \geq 0, \forall (i,j) \in V, \forall v \in M_i \cap M_j, J_i \neq J_j$;

Dans ce programme linéaire, H est une constante suffisamment grande : $H > \sum_{i \in V} max_{m \in M_i}(P_{i,m})$.

L'ensemble des contraintes (i), (ii), (iii), (iv'), (iv") et (iv"") représentent les contraintes du problème de Job-shop Flexible. Associées à la fonction objectif (v), ces contraintes forment un programme linéaire en nombres entiers modélisant le problème de Job-shop Flexible.

Ce programme linéaire se ramène facilement à celui du Job-shop. Dans ce dernier, l'affectation est connue en amont de l'ordonnancement. De ce fait, chaque ensemble de machines pouvant traiter une opération est un singleton, et la durée de traitement p_i est également connue. Il est donc possible de supprimer les contraintes (ii) et (iii) et de reformuler les contraintes (iv') et (iv") comme suit :

(iv') $\quad x_{i,j,v} + x_{j,i,v} = 1, \forall (i,j) \in V, \forall v \in M_i \cap M_j, J_i \neq J_j.$

La formalisation linéaire ci-dessus permet de résoudre tous les problèmes de Job-shop Flexible. Cependant, les temps de calcul nécessaires peuvent rapidement devenir prohibitifs. De ce fait, il est délicat d'aborder la résolution d'un problème de taille importante (dépassant 100 opérations) à l'aide d'un programme linéaire tel que mentionné ci-dessus. Ainsi, la plupart des articles traitant du problème de Job-shop Flexible reposent principalement sur des méthodes heuristiques et/ou métaheuristiques afin de proposer des solutions pour ces problèmes (Chaudhry and Khan, 2016).

Les méthodes de résolutions présentes dans la littérature proposent des ordonnancements de différentes natures et dont les représentations peuvent varier d'un article à un autre. Les représentations pour l'ordonnancement ne sont pas détaillées car elles sont principalement basées sur celle pour le Job-shop (présentées dans le chapitre II, §3.2). Cependant, la représentation la plus utilisée pour le problème d'affectation inclus dans le problème de Job-shop Flexible est introduite.

3. Représentation pour l'affectation

Pour le problème de Job-shop Flexible, deux sous-problèmes sont considérés : un problème d'ordonnancement et un problème d'affectation. Du fait de la structure du problème, les représentations permettant de coder une solution d'un problème de Job-shop peuvent être utilisées pour le problème d'ordonnancement présent dans le Job-shop Flexible. Pour le problème d'affectation, la représentation utilisée est détaillée dans cette section.

En effet, la particularité du Job-shop Flexible, par rapport au Job-shop, est la possibilité d'affecter une opération à une machine sélectionnée parmi un ensemble de machines compatibles. Ainsi, une représentation intuitive consiste à stocker dans un tableau la machine affectée à une opération. Cependant, cet élément de codage de l'affectation est utilisé en parallèle de celui utilisé pour l'ordonnancement, il est donc préférable d'indexer le tableau des affectations par rapport au numéro des opérations, et non par rapport à l'emplacement des opérations dans l'outil de codage de l'ordonnancement. L'objectif étant de garder une totale indépendance entre les deux éléments de codage. En effet, dans le cadre du Job-shop Flexible, le module d'optimisation va effectuer des modifications sur les solutions partielles afin d'explorer l'espace des solutions, à la fois au niveau de l'ordonnancement et de l'affectation. De ce fait, il faut éviter de faire des modifications sur un vecteur qui impliquent des modifications sur un autre vecteur dans le but de préserver l'intégrité des solutions et éviter des modifications superflues.

La représentation choisie consiste donc à sauvegarder la machine affectée à une opération au sein d'un vecteur. Dans le cadre du Job-shop Flexible, le nombre de vecteurs d'affectations ainsi

construits vaut $\prod_{i \in V} card(M_i)!$. Il n'y a pour ces vecteurs aucune surreprésentation, ni multireprésentation possible. Deux vecteurs d'affectations possibles pour le problème P1 sont présentés dans le Tableau 2.

Tableau 2 Représentations pour l'affectation dans un problème de Job-shop Flexible

	O_1	O_2	O_3	O_4	O_5	O_6	O_7	O_8	O_9
A	2	3	1	3	4	1	2	2	1
B	1	3	4	3	4	2	4	2	3

Dans le Tableau 2 (ligne A), l'opération 1 est affectée à la machine M_2, l'opération 2 est affectée à la machine M_3, et ainsi de suite. Le vecteur B présente une autre affectation possible, conduisant à une solution différente (Figure 1(B)). Dès lors que les opérations sont affectées aux machines, le problème devient un problème de Job-shop. Il est donc possible de représenter sous forme de graphe les sous-problèmes obtenus à partir des vecteurs d'affectation du Tableau 2.

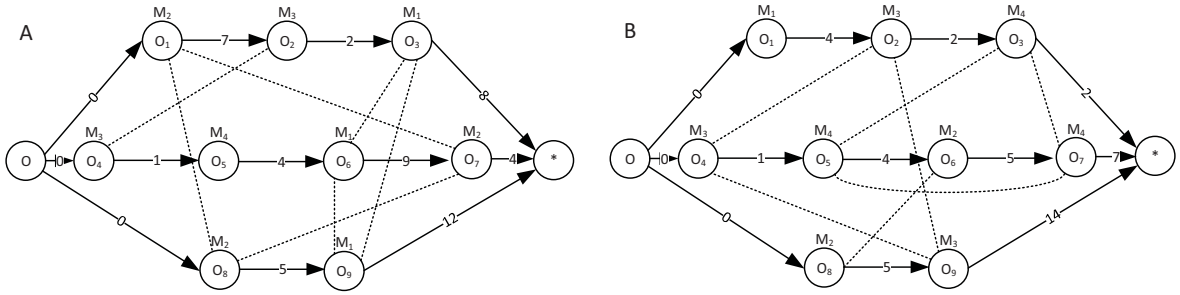

Figure 1 Exemple de deux graphes obtenus après affectation

A l'aide de la Figure 1 il est possible de souligner que l'affectation joue un rôle primordial dans la structure des graphes manipulés, et que par conséquent, elle a une incidence sur la qualité des solutions retournées par un algorithme d'optimisation telle qu'une métaheuristique. Dans la suite certains travaux sur le Job-shop Flexible utilisant la représentation présentée dans cette section sont synthétisés

4. Etat de l'art du problème de Job-shop Flexible

La littérature sur les problèmes d'ordonnancement de type Job-shop Flexible est plus réduite que celle sur le problème de Job-shop, mais reste conséquente. Citer tous les articles n'est donc pas envisageable, et seulement certains d'entre eux sont présentés dans cette partie. S'il existe des études visant à référencer les articles traitant de certaines extensions du problème, son caractère relativement récent entraîne, pour la forme classique, une rareté d'état de l'art. Le dernier trouvé est celui de (Chaudhry and Khan, 2016), qui montre l'évolution rapide de l'attention portée à ce type de problèmes.

Dans ce paragraphe, l'état de l'art de (Chaudhry and Khan, 2016) ainsi que certains articles permettant d'avoir une idée générale des travaux menés jusqu'à ce jour sont présentés. Les outils sur lesquels reposent les algorithmes de résolution proposés sont également détaillés.

4.1. Généralités

L'état de l'art récent proposé par (Chaudhry and Khan, 2016) concernant le Job-shop Flexible a permis de mettre en valeur plus de 400 articles abordant le problème ainsi que ses extensions. Sur l'ensemble des articles que les auteurs ont pu trouver, près de 200 articles publiés dans des revues scientifiques ont été extraits. Le problème est présenté bien qu'aucune formalisation mathématique ne soit introduite. Chaque article retenu bénéficie d'une présentation rapide de la ou des méthodes employées. Les auteurs font cependant une étude plus générale du contexte dans lequel ces articles sont publiés, en notant par exemple que c'est à partir des années 2010 que l'on fait face à une augmentation importante du nombre d'articles traitant des problèmes de Job-shop Flexible, et que plus de 93% des articles consistent en des travaux de recherche « pure ». Parmi les fonctions objectifs recensées, la plus utilisée reste le makespan, avec plus de 84% des articles utilisant ce critère. Un autre point important concerne les méthodes utilisées, car si la majeure partie des algorithmes sont des métaheuristiques, ces dernières sont principalement des métaheuristiques hybrides, reposant sur des algorithmes évolutionnaires ou des recherches tabou. Cependant certains critères de leur étude ne présentent que peu d'intérêt, tel que le nombre d'articles par pays, car ce critère n'est pas pondéré par le nombre de chercheurs dudit pays et un ratio eût été plus révélateur.

De manière générale, les méthodes de la littérature pour le Job-shop Flexible reprennent des éléments clés du problème de Job-shop : le graphe conjonctif-disjonctif adapté au problème, les chemins critiques, ou encore l'évaluation des solutions. Ces deux derniers éléments ont été présentés dans le chapitre II (§4.1.2 et §4.1.3) et ne sont pas de nouveau détaillés ici. Cependant, une présentation du graphe conjonctif-disjonctif adapté au problème étudié est proposée, ainsi qu'un système de voisinage pour le Job-shop Flexible, avant d'introduire un aperçu des méthodes utilisées dans la littérature pour résoudre le problème.

4.2. Outils pour la résolution des problèmes de FJSP

4.2.1. Graphe conjonctif-disjonctif pour le Job-shop Flexible

(Dauzère-Pérès and Paulli, 1997) ont étendu la notion de graphe conjonctif-disjonctif au Job-shop Flexible. Graphiquement, chaque couple d'opération pouvant être affecté à la même machine est relié par une arête. Chaque arête associée à une machine a une forme différente. Lorsqu'une affectation est choisie, certaines arêtes sont supprimées du graphe car elles ne représentent plus les potentielles précédences. Le graphe disjonctif avant l'affectation, qui représente donc le problème P1 est présenté ci-dessous. (Dauzère-Pérès and Paulli, 1997) ont utilisé cette approche pour concevoir une recherche tabou efficace.

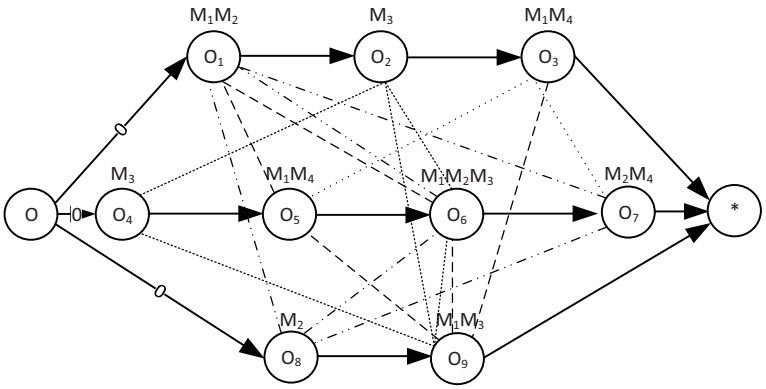

Figure 2 Graphe Disjonctif adapté au Job-shop Flexible (Dauzère-Pérès and Paulli, 1997)

D'après la Figure 2, il faut donc sélectionner une machine pour chaque opération ainsi que l'orientation des arêtes restantes. Certaines arêtes sont omises sur la figure car elles lient des opérations appartenant au même job, elles n'ont donc pas de sens compte tenu des contraintes de précédences. Si l'on se concentre sur les arcs restants, la démarche d'obtention d'une solution pour un problème de Job-shop Flexible est présentée sur la Figure 3.

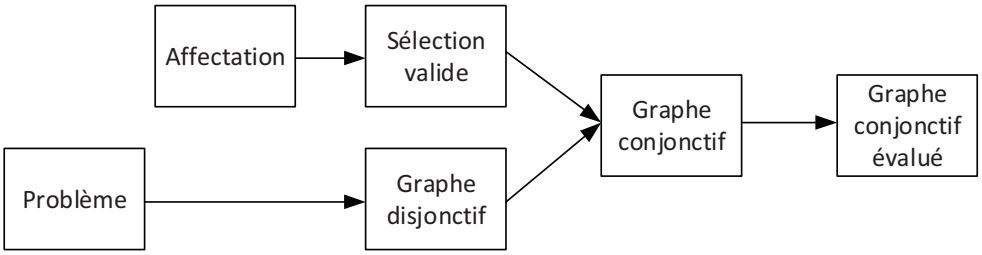

Figure 3 Démarche d'obtention d'un ordonnancement semi-actif pour le Job-shop Flexible

Dans la Figure 3, l'affectation va permettre d'éliminer un certain nombre d'arêtes entre les opérations pouvant être affectées à la même machine, puis la sélection va permettre d'obtenir un graphe conjonctif, dont l'évaluation conduit à la valeur du critère de performance mesuré.

4.2.2. Le voisinage de (Mastrolilli and Gambardella, 2000)

Les voisinages pour le problème de Job-shop Flexible sont souvent découpés pour générer d'un côté des voisins pour l'ordonnancement et de l'autre des voisins pour l'affectation. La manière dont les voisinages sont appliqués dépend fortement de la métaheuristique et de l'exploration de l'espace des solutions. Il existe cependant des voisinages intégrant les deux approches, comme ceux de (Mastrolilli and Gambardella, 2000). Les auteurs proposent deux voisinages NFJ1 et NFJ2 basés sur la notion de *k*-insertion.

Dans le premier (NFJ1), une opération du chemin critique est sélectionnée puis supprimée de la séquence des opérations sur la machine qui lui est affectée (i.e. les arêtes connectant l'opération à toute autre opération affectée à la même machine sont supprimées). Pour chaque machine compatible avec l'opération, les arêtes correspondantes sont ajoutées au graphe, puis elles sont orientées afin que la

séquence des opérations sur la machine conduise au meilleur makespan. Le voisinage est alors formé de toutes les solutions obtenues par application de cette procédure à chaque opération du chemin critique.

Le second voisinage (NFJ2) contient le premier. Il comprend toutes les solutions obtenues avec NFJ1. Lorsque la machine considérée est celle actuellement affectée à l'opération, toutes les solutions valides obtenues en réorientant les arcs qui connectent l'opération courante avec les autres opérations affectées à la machine sont ajoutées. Les auteurs ont montré que ce second voisinage est faiblement connecté. Cependant, ils ont utilisé le voisinage NFJ1 qui présente de meilleurs résultats en pratique. Une illustration du principe de leur voisinage est donnée ci-dessous, sur un problème plus important que P1.

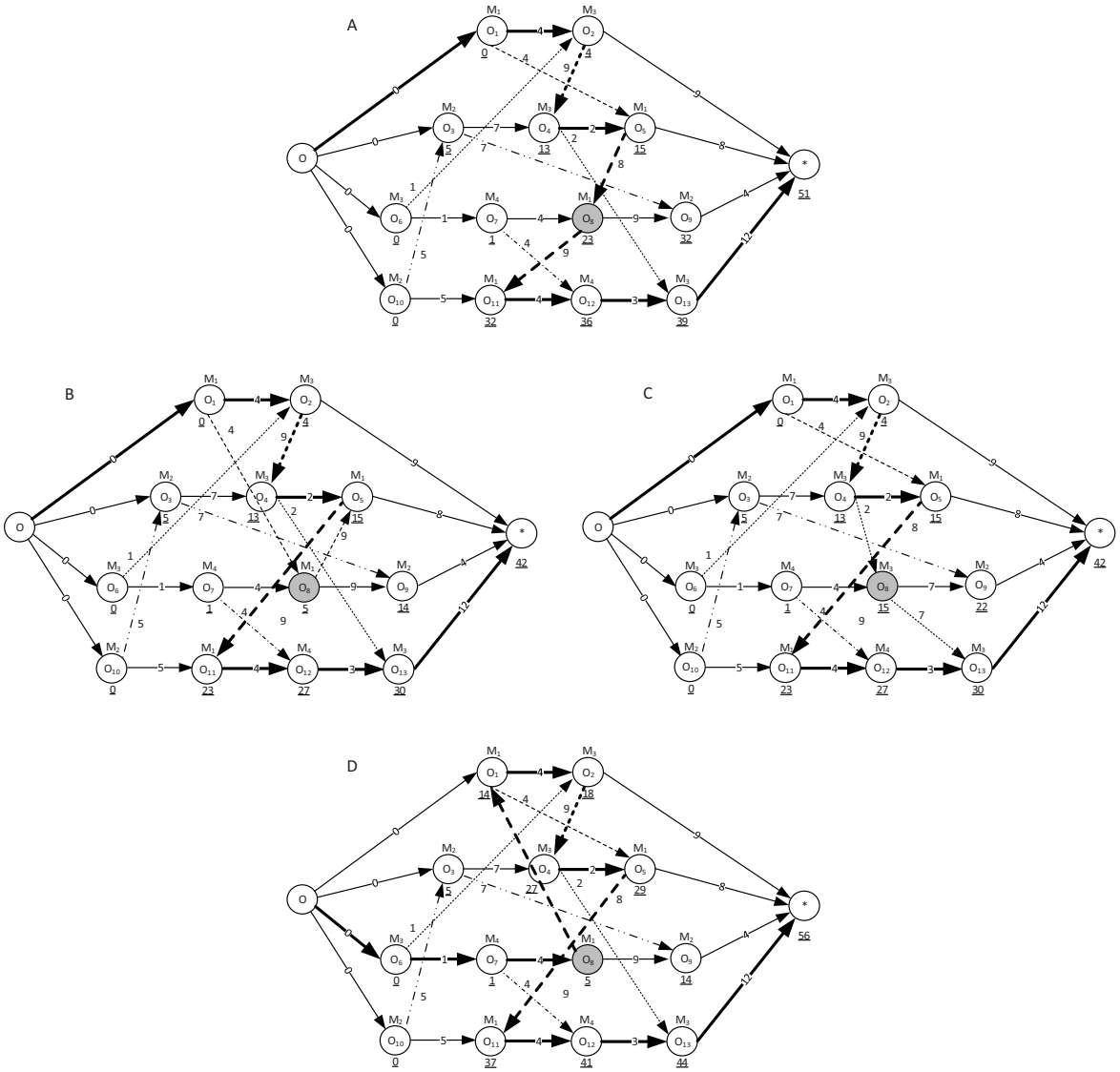

Figure 4 Exemples d'application du voisinage de (Mastrolilli and Gambardella, 2000) – changement de l'ordre de passage (B-D), ou changement de machine puis réinsertion (C)

Dans la Figure 4(A), un graphe est considéré ; le chemin critique y est représenté à l'aide d'arcs noirs épais. L'opération O_8 est sur ce chemin (opération en gris). On suppose qu'elle peut être affectée uniquement aux machines M_1 ou M_3. Si le voisinage NFJ1 est considéré, alors il est possible de créer une nouvelle solution à partir de la solution courante (A) en changeant l'ordre des opérations utilisant une machine donnée. Par exemple, considérons la machine M_1et l'opération O_8. Les ordres de traitement des opérations O_1, O_5, O_8 et O_{11} sur la machine M_1 sont testés. Ainsi les ordres suivants sont évalués : O_8, O_1, O_5 et O_{11} ; O_1, O_8, O_5 et O_{11} ; O_1, O_5, O_{11} et O_8. Cette solution, qui doit être la meilleure dans NFJ1, est montrée sur la partie B de la Figure 4. L'opération peut également être affectée à la machine M_3 ; la solution obtenue (Figure 4(C)) en choisissant cette nouvelle affectation et en insérant O_8 entre les opérations O_4 et O_{13} dans l'ordre de passage sur la machine est une solution de NFJ1. Le voisinage NFJ2 est similaire lorsque l'opération est affectée à une nouvelle machine. Lorsque l'affectation n'est pas modifiée, toute solution obtenue en changeant l'ordre de passage de l'opération O_8 est dans NFJ2. La solution exprimée par le graphe Figure 4(D) est donc dans ce voisinage.

Les voisinages ainsi que le graphe conjonctif-disjonctif ont permis l'élaboration de méthodes efficaces au sein de recherches locales et de métaheuristiques. La plupart de ces méthodes reposent sur des propriétés du chemin critique, couramment utilisées dans les problèmes disjonctifs. Dans la suite, certains articles qui mettent en œuvre ces outils dans le cadre de la résolution du Job-shop Flexible sont présentés.

4.3. Travaux récents sur le problème de FJSP

Les méthodes de la littérature sur le Job-shop Flexible reposent principalement sur des outils particuliers dont certains sont communs avec ceux utilisés pour le Job-shop. Le graphe conjonctif-disjonctif, le chemin critique, les systèmes de voisinage et les heuristiques/métaheuristiques sont donc couramment utilisés pour résoudre le problème. Certaines des méthodes de résolution employées dans la littérature reposant sur ces outils sont présentées ci-dessous.

Deux approches sont couramment utilisées pour résoudre le problèmes de FJSP : (i) l'approche hiérarchisée, qui consiste à trouver le meilleur ordonnancement tout en considérant fixée l'affectation des opérations aux machines (Brandimarte, 1993); ; *(ii)* ou l'approche intégrée qui considère simultanément l'ordonnancement et l'affectation par l'utilisation de voisinages efficaces (Mastrolilli and Gambardella, 2000). De manière générale, la deuxième approche est bien plus répandue dans la littérature, car la première est en pratique trop couteuse en temps de calcul.

(Gao et al., 2008) proposent un algorithme génétique hybride (hGA) pour résoudre le problème. Deux vecteurs sont utilisés pour coder les machines et les opérations. Ils utilisent un ordonnancement actif avec un décodage basé sur la priorité. Leur algorithme contient une technique de croisement classique pour les opérations et deux techniques pour les machines. La première (extended) consiste à choisir la machine utilisée par le parent dont l'opération est ajoutée dans le vecteur de codage des opérations du fils. La seconde (uniform) consiste à sélectionner de manière aléatoire la machine entre les deux parents. Deux mutations sont ensuite utilisées : une mutation « basée allèle » et une autre appelée « immigration ». La mutation basée allèle permet de changer la machine affectée à une opération ou de permuter deux opérations ; celle appelée immigration génère de nouveaux membres dans la population. Les éléments de la population sont sélectionnés en fonction de leur qualité. Les auteurs proposent d'améliorer leur algorithme à l'aide d'une métaheuristique de type VND (Variable

Neighbourhood Descent), qui consiste à réinsérer une opération du chemin critique à un emplacement plus favorable. Les résultats obtenus à l'aide de leur métaheuristique hybride sont de bonne qualité.

(Ben Hmida et al., 2010) ont proposé un algorithme appelé Climbing Depth-bound Discrepancy Search (CDDS). Cette métaheuristique est une méthode à divergence limitée, qui consiste en un parcours d'arborescence. En partant d'une solution obtenue de manière heuristique, l'arbre est construit en effectuant des modifications sur la solution. Cette méthode permet de n'explorer qu'une partie de l'espace de recherche, en élaguant les branches non prometteuses. Dans leur approche, six différents voisinages sont proposés. Le premier consiste à (i) insérer chaque opération du chemin critique dans la séquence des opérations devant être traitées par une autre machine - la priorité est donnée à l'opération insérée en cas de conflit avec une autre opération effectuée sur la machine - ou (ii) à réordonnancer toutes les opérations à l'intérieur d'un bloc critique (différentes des opérations en tête et en fin de bloc) en les plaçant en tête ou en fin du bloc. Le second voisinage est similaire au premier à ceci près que chaque opération est affectée à une autre machine si son insertion dans la séquence de passage modifie la date de début d'une opération déjà planifiée en fonction des dates de fin au plus tôt. Lors d'un déplacement d'une opération en tête ou en fin de bloc, son successeur est également réinséré en fonction des dates de fin au plus tôt. Les troisièmes et quatrièmes voisinages sont des généralisations, respectivement, du premier et du second voisinage. Dans ces deux voisinages, le changement d'affectation des opérations d'un bloc diffère, avec seulement une opération sélectionnée. Les cinquièmes et sixièmes voisinages généralisent les voisinages trois et quatre, en considérant le déplacement d'une opération d'un bloc (différente de la première ou la dernière) avant chaque opération du bloc. Les auteurs ont montré que ce type d'approche permettait d'obtenir des solutions de qualité en un temps raisonnable.

(Yuan and Xu, 2013a) ont proposé une approche basée sur une métaheuristique évolutionnaire de type algorithme à évolution différentielle (Differential Evolution Algorithm) qui prend en compte des nombres réels dans la représentation des individus. Les briques logicielles classiques des algorithmes génétiques sont présentes, telles que le croisement et la mutation, adaptés aux vecteurs de réels. Chaque vecteur est transformé en un vecteur d'entier par une procédure dédiée. Une recherche locale est également utilisée afin d'améliorer la qualité de la population, en réinsérant les opérations critiques à une meilleure position dans les différentes séquences de machines pouvant traiter l'opération. Par la suite, (Yuan and Xu, 2013b) ont proposé une deuxième méthode davantage orientée vers les problèmes de grande taille. La métaheuristique proposée est basée sur un algorithme de type recherche harmonique hybride (Hybrid Harmony Search - HHS). Cette métaheuristique est également une méthode à population utilisant des vecteurs de réels. Une conversion de ces vecteurs est faite pour passer là aussi à des vecteurs d'entiers. Différentes méthodes propres à la métaheuristique sont implémentées, telle que l'improvisation (méthode visant à générer la nouvelle harmonie), proche du croisement et de la mutation. Une recherche locale est également développée par les auteurs afin d'améliorer la qualité des solutions. Cette recherche locale consiste à changer l'affectation et la position dans le chromosome d'une opération du chemin critique. Les auteurs vont encore plus loin en couplant la métaheuristique HHS avec une métaheuristique de type 'recherche de voisinage étendu' (Large Neighbourhood Search – LNS), et notée HHS/LNS. Ce type d'approche consiste à modifier en profondeur une solution en effectuant de nombreuses modifications, alors que les recherches locales classiques vont modifier une ou deux opérations. Les métaheuristiques ainsi définies par les auteurs permettent l'obtention de bonnes solutions, mais les temps de calcul semblent importants.

(González et al., 2015) ont proposé une métaheuristique basée sur de la recherche dispersée (Scatter Search) couplée à une recherche tabou et à une reconnexion de chemin (Path Relinking) ;

cette métaheuristique est notée SSPR. La méthode est une méthode à population qui consiste à faire évoluer les individus en fonction d'un ensemble de solutions de références. A chaque itération, un couple de solutions est choisi dans l'ensemble de référence et une nouvelle solution est obtenue à l'aide du Path Relinking. Cette solution est améliorée en utilisant une recherche tabou. Si aucune nouvelle solution n'entre dans l'ensemble de référence après avoir traité tous les couples d'opérations de l'ensemble, une phase de diversification est appliquée. Différentes structures de voisinages sont utilisées. Trois voisinages sont proposés pour le problème d'ordonnancement. Le premier consiste à déplacer une opération d'un bloc à un autre emplacement dans le bloc. Le second consiste à déplacer une opération intérieure à un bloc critique au début ou à la fin du bloc. Le troisième procède à une permutation de deux opérations consécutives sur la même machine. Concernant le problème d'affectation, un voisinage est proposé permettant de modifier l'affectation d'une opération à une machine si elle appartient au chemin critique. De la même manière, une généralisation de ce voisinage consiste à changer l'affectation de n'importe quelle opération. Des voisinages combinant ceux précédemment mentionnés sont également définis. Certains voisinages sont utilisés dans le Path Relinking, ou dans la recherche tabou car ils sont plus appropriés à l'un ou à l'autre. Une procédure d'estimation est également proposée afin de réduire les temps de calcul. Leur méthode présente d'excellents résultats et est l'une des meilleures méthodes publiées à ce jour.

(Palacios et al., 2015) ont proposé un algorithme génétique couplé à une recherche tabou, noté HGTS, dans le cadre d'un problème de « Fuzzy Flexible Job-shop ». Leur métaheuristique est appliquée au problème classique dans un premier temps. L'opérateur de croisement utilisé est JOX pour Job Order Crossover. Considérant deux parents, l'opérateur de croisement sélectionne un ensemble de jobs et réinsère dans le premier descendant les jobs au même emplacement que dans le premier parent. Les jobs restants sont réintroduits à partir de leur position relative dans le deuxième parent. Le second descendant est obtenu en inversant le rôle des deux parents. Une approche similaire est utilisée pour le problème d'affectation. Les auteurs n'utilisent cependant aucune procédure de mutation. Une recherche locale est utilisée, reposant sur une recherche tabou. Cette recherche locale repose sur trois voisinages. Le premier change l'affectation d'une opération à une nouvelle machine. Le second intervertit l'ordre de traitement de deux opérations sur la même machine, et le troisième est une combinaison des deux. Deux voisinages réduits sont également proposés, basés sur les blocs critiques. De bons résultats sont obtenus à l'aide de leur approche.

(Li and Gao, 2016) ont également proposé un algorithme hybride, couplant un algorithme génétique à une recherche tabou, et noté HA. Deux opérateurs de croisements sont proposés. Le premier (POX) est proche de celui utilisé dans (Palacios et al., 2015). Les jobs sont divisés aléatoirement en deux ensembles. Considérant deux parents, l'opérateur de croisement sélectionne les jobs du premier ensemble et réinsère dans le premier descendant les jobs au même emplacement que dans le premier parent. Ensuite les jobs restant sont réintroduits à partir de leur position relative dans le deuxième parent. Le second descendant est obtenu en inversant le rôle des deux parents, et en choisissant le deuxième ensemble de Jobs. Le deuxième opérateur (JBX) est très proche du premier. Deux ensembles de jobs sont également considérés. Dans un premier temps les jobs du premier ensemble du premier parent sont insérés à la même position dans le premier descendant, les éléments du deuxième parent viennent remplir les espaces vacants. Le deuxième descendant est généré en insérant les éléments du deuxième parent appartenant au deuxième ensemble, puis à remplir les espaces vacants avec les éléments du premier parent. Deux opérateurs de mutations sont utilisés pour le problème d'ordonnancement. Le premier consiste à permuter deux éléments d'une solution, tandis que le second procède à un échange entre trois opérations. Pour le problème d'affectation, la moitié

des opérations se voit affecter une nouvelle machine. La recherche tabou implémentée repose sur le voisinage de (Mastrolilli and Gambardella, 2000).

Comme il est possible de constater au travers de cette vue d'ensemble des méthodes actuelles, la conception des voisinages est au cœur de la réalisation de métaheuristiques efficaces. Les résultats obtenus sur des instances classiques de la littérature à l'aide des travaux présentés dans cette section ont été utilisés comme éléments de comparaison avec une métaheuristique développée pour traiter le problème de Job-shop Flexible. Cette métaheuristique est présentée ci-dessous.

5. Proposition d'un GRASP-mELS (Kemmoé-Tchomté et al., 2017)

D'après les articles précédemment mentionnés, de nombreux travaux effectués sur le Job-shop Flexible reposent sur l'utilisation d'algorithmes à population. Cette observation est validée par l'état de l'art de (Chaudhry and Khan, 2016) qui recensent de nombreux algorithmes évolutionnaires utilisés pour la résolution du problème. Face à ce constat, le choix a été fait d'étudier une approche à solution unique (Boussaïd et al., 2013). Les différentes méthodes proposées et les résultats de notre démarche sont présentés dans cette section. Ces travaux reposent sur une première étude faite par (Kemmoe et al., 2016).

5.1. Présentation générale

Dans les problèmes d'ordonnancement, il est courant d'utiliser des concepts tirés de la littérature et adaptés au problème étudié. Parmi les approches utilisées, le graphe disjonctif et ses propriétés sont abordés dans plusieurs travaux. De nombreuses métaheuristiques sont également proposées. La qualité des résultats dépend donc de la manière dont est utilisée la connaissance extraite des articles de la littérature (Yuan and Xu, 2013a).

Pour résoudre le FJSP, beaucoup d'attention a été portée sur les métaheuristiques telles que la recherche tabou, les algorithmes évolutionnaires ou encore le GRASP (Rajkumar et al., 2011). La plupart d'entre eux s'appuient sur des métaheuristiques existantes qui sont améliorées pour répondre aux problèmes rencontrés, conduisant ainsi à des métaheuristiques hybrides. La première idée a été d'utiliser le GRASP×ELS pour le FJSP, car cette métaheuristique a permis d'obtenir des solutions de bonne qualité dans le chapitre II (§5.2) sur le Job-shop. Cependant, bien que de nombreux paramètres aient été testés au cours d'un plan d'expérience, les résultats n'étaient pas convaincants. Néanmoins, il a été choisi d'explorer plus en détail la piste des métaheuristiques à démarrages multiples en modifiant leur structure. Par conséquent, la métaheuristique présentée dans cette section est une version améliorée du GRASP×ELS, où la procédure GRASP est utilisée conjointement avec une approche ELS à niveaux multiples. Cette métaheuristique est notée GRASP-mELS, pour « GRASP with multi-level ELS ». L'une des caractéristiques de cet algorithme est qu'il fonctionne sur l'espace de codage et l'espace des solutions de la même manière que le GRASP×ELS. Les principaux points du GRASP-mELS proposé pour résoudre les problèmes FJSP sont les suivants :

- Une représentation sous forme de graphe conjonctif-disjonctif ;
- Une représentation indirecte des solutions qui permet de manipuler une représentation compacte de la solution ;

• Des structures de voisinages efficaces qui permettent d'explorer l'espace de solution en passant d'une solution à une autre ;

• Une heuristique de construction adaptée au problème étudié ;

• Une recherche locale qui procède à des changements d'affectation et à des permutations sur le chemin critique ;

• Une procédure d'estimation qui permet de réduire les temps de calcul ;

• Une métaheuristique efficace, qui permet un bon équilibre entre l'intensification et la diversification.

Dans la section suivante, une analyse approfondie est faite pour proposer un GRASP-mELS efficace, en commençant par les structures de voisinages utilisées.

5.2. Méthodes de résolution

5.2.1. Voisinages utilisés

L'état de l'art a montré que les voisinages tiennent une place importante dans la présentation des travaux récents effectués sur les problèmes d'ordonnancement. Habituellement, les métaheuristiques intègrent divers algorithmes pour explorer l'espace des solutions. Les structures de voisinages font partie de ces algorithmes. Si dans le JSP, les structures de voisinage sont axées sur les opérations, dans le FJSP, l'affectation des opérations sur les machines doit également être prise en compte. Ainsi, diverses structures de voisinages peuvent être utilisées pour le FJSP. Dans cette étude, quatre structures de voisinages sont appliquées.

Pour le problème d'ordonnancement, deux voisinages sont utilisés, N_n^{π} et $N_1^{\pi-\rho}$, où n correspond au nombre de fois où le voisinage est appliqué (1 signifie que le voisinage n'est utilisé qu'une fois) et ρ représente le chemin critique. Ces deux voisinages étendent les voisinages de (Nowicki and Smutnicki, 1996 ; Van Laarhoven et al., 1992).

Définition : structure N_n^{π}

Soit S une solution du problème considéré. Soient O^1 et O^2 deux ensembles d'opérations de taille n, avec $O_i^1 \neq O_i^2$ et où O_i^1 et O_i^2 sont affectées à la même machine. Une solution voisine S' est obtenue à partir de S en intervertissant l'ordre de traitement de O_i^1 et O_i^2 pour chaque i.

Ce voisinage opérant sur l'ensemble de la solution, il est probable que certains mouvements ne conduisent pas à une solution améliorante. Il est donc peu adapté à l'élaboration d'une recherche locale, mais approprié pour les phases de diversification au sein d'une métaheuristique. Dans la suite un voisinage plus restrictif, noté $N_1^{\pi-\rho}$, est présenté.

Définition : structure $N_1^{\pi-\rho}$

Soit S une solution du problème considéré. Soit O l'ensemble des opérations consécutives en début ou en fin des blocs sur le chemin critique. Une solution voisine S' est obtenue à partir de S en intervertissant l'ordre de traitement des deux opérations conduisant à la meilleure amélioration du makespan de S.

Dans le travail de (Mastrolilli and Gambardella, 2000), la notion de *k*-insertion est proposée, où l'objectif est de modifier l'affectation d'une opération à une autre machine, puis de trouver la meilleure insertion pour cette opération parmi les autres opérations affectées à la machine. Dans le travail effectué, deux structures de voisinages plus simples sont utilisées. Elles consistent à modifier l'affectation d'une ou de plusieurs opérations.

Définition : Structure N_n^{α}

Soit S une solution du problème considéré. Soit O un ensemble de n opérations O_i. Une solution voisine S' est obtenue à partir de S en changeant la machine affectée à l'opération O_i.

Comme on peut le remarquer, une modification de l'affectation d'une opération en dehors du chemin critique n'entraînera pas une amélioration directe du critère d'optimisation. Une restriction de N_n^{α} est donnée dans (González et al., 2015) et redéfinie ci-dessous.

Définition : Structure $N_1^{\alpha-\rho}$

Soit S une solution du problème considéré. Soit O_i une opération sur un chemin critique ρ. Une solution voisine S' est obtenue à partir de S en changeant la machine affectée à l'opération O_i. (On applique donc N_1^{α} à l'opération sélectionnée).

L'ensemble de ces voisinages est utilisé dans le cadre de différentes méthodes au sein de la métaheuristique, que ce soit à des fins de diversification des solutions, ou lors des phases d'intensification.

5.2.2. Représentation d'une solution

Afin de représenter efficacement les solutions d'un problème de Job-shop Flexible, la représentation par répétition (Bierwirth, 1995) pour le problème d'ordonnancement et la représentation pour l'affectation présentée dans la deuxième section de ce chapitre sont utilisées. Pour rappel, les deux vecteurs permettant de coder une solution sont présentés dans la Figure 5. Dans cette représentation, la première opération du Job 1 est affectée à la machine M_1, la première opération du job 2 à la machine M_3, la première opération du Job 3 est affectée à la machine M_3, et ainsi de suite.

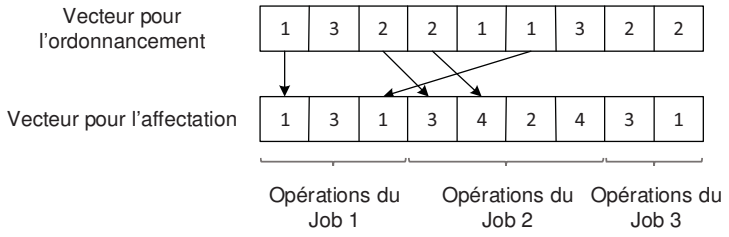

Figure 5 Vecteurs permettant le codage d'une solution de P1.

5.2.3. Composants du GRASP-mELS

Dans cette section, une métaheuristique originale basée sur des voisinages efficaces pour le FJSP avec minimisation du makespan est présentée. Des algorithmes visant à améliorer la qualité des solutions ainsi qu'une estimation du makespan des voisins sont proposés. Ces composants sont incorporés dans une métaheuristique qui utilise le GRASP avec ELS à niveaux multiples (GRASP-mELS) comme méthode d'optimisation.

L'Algorithme 1 illustre les principes et la mise en œuvre du GRASP-mELS en pseudo-code. *nb_g* représente le nombre de redémarrages du GRASP (étape de diversification), *nb_ol* et *nb_els* représentent le nombre d'itérations du mELS (étape d'intensification) et *nb_n* correspond au nombre de voisins qui doivent être générés. La fonction *f* réfère à la fonction objectif optimisée dans le processus (makespan).

Le bloc d'instructions entre les lignes 2 et 25 est exécuté itérativement ; chaque itération se compose de trois phases comme suit :

i. Phase de construction (ligne 3) : les solutions initiales sont construites, un élément à la fois, avec une heuristique randomisée. À chaque itération de la construction, l'élément suivant à ajouter est déterminé en ordonnant tous les éléments dans une liste de candidats. La composante probabiliste d'un GRASP dans cette phase se caractérise par le choix aléatoire d'un des candidats dans la liste, qui n'est pas toujours le meilleur.

ii. Phase de recherche locale (ligne 4) : puisque la solution retournée après la phase de construction n'est pas garantie être localement optimale, une recherche locale est effectuée. Dans la ligne 5, la qualité de la solution obtenue est comparée à celle actuellement trouvée et, si nécessaire, la solution est mise à jour.

iii. Phase de recherche locale évolutionnaire à niveaux multiples (mELS) (ligne 6-31) : pour mieux explorer le voisinage de l'optimum local, un voisin de la solution actuelle est généré. Ce voisin est ensuite utilisé comme point de départ de la phase ELS classique (ligne 8-18). A la ligne 20, la qualité de la solution obtenue est comparée à la meilleure trouvée et, si nécessaire, la meilleure solution est mise à jour ; on retourne alors au premier niveau du mELS afin d'explorer davantage le voisinage de la nouvelle solution trouvée.

Algorithme 1 : GRASP-mELS

Entrée

$data$	Données du problème
nb_g	Nombre de redémarrages (GRASP)
nb_ol	Nombre d'itérations du premier niveau du mELS
nb_els	Nombre d'itérations du second niveau du mELS
nb_n	Nombre de voisins à générer dans le mELS
β	Coefficient pour la phase de construction

Sortie

S^*	Meilleure solution trouvée

Variables

S, S'	Une solution et son voisin
nS, nS^*	Un voisin et le meilleur voisin rencontré durant les phases d'ELS
elS^*	Meilleure solution trouvée durant la phase d'ELS

Début

1. $S^* \coloneqq \emptyset$;
2. **POUR** $g \coloneqq 1$ **A** nb_g **FAIRE**
3. $S \coloneqq$ **Phase_de_Construction**$(data, \beta)$;
4. $S \coloneqq$ **Phase_de_Recherche_Locale**$(data, S)$;
5. **SI** $(S^* == \emptyset$ OU $f(S) < f(S^*))$ **ALORS** $S^* \coloneqq S$ **FIN SI**
6. **POUR** $ol \coloneqq 1$ **A** nb_ol **FAIRE**
7. $S' \coloneqq$ **Generation_Voisin**$(data, S)$;
8. $elS^* \coloneqq \emptyset$;
9. **POUR** $els \coloneqq 1$ **A** nb_els **FAIRE**
10. $nS^* \coloneqq \emptyset$;
11. **POUR** $n \coloneqq 1$ **A** nb_n **FAIRE**
12. $nS \coloneqq$ **Generation_Voisin**$(data, S')$;
13. $nS \coloneqq$ **Phase_de_Recherche_Locale**$(data, nS)$;
14. **SI** $(f(nS) < f(nS^*))$ **ALORS** $nS^* \coloneqq nS$ **FIN SI**
15. **FIN POUR**
16. $S' \coloneqq nS^*$;
17. **SI** $(f(nS^*) < f(elS^*))$ **ALORS** $elS^* \coloneqq nS^*$ **FIN SI**
18. **FIN POUR**
19. $S \coloneqq elS^*$;
20. **SI** $(f(S) < f(S^*))$ **ALORS**
21. $S^* \coloneqq S$;
22. $ol \coloneqq 1$;
23. **FIN SI**
24. **FIN POUR**
25. **FIN POUR**
26. Retourner S^* ;

Fin

Cette métaheuristique peut être représentée graphiquement à l'aide de la Figure 6. Dans cette figure, un premier ELS (rectangle mELS1) est appliqué après avoir construit une solution initiale. Pendant cet ELS, le meilleur voisin obtenu est sauvegardé (disque gris). Une fois que l'ELS atteint un critère d'arrêt, un voisin du meilleur voisin (précédemment enregistré) est généré et une nouvelle phase d'ELS (rectangle mELS2) est appliquée sur cette nouvelle solution. Ce processus peut être répété plusieurs

fois. Une fois que le nombre maximum de mELS est atteint, une nouvelle solution est générée et le processus d'exploration peut recommencer.

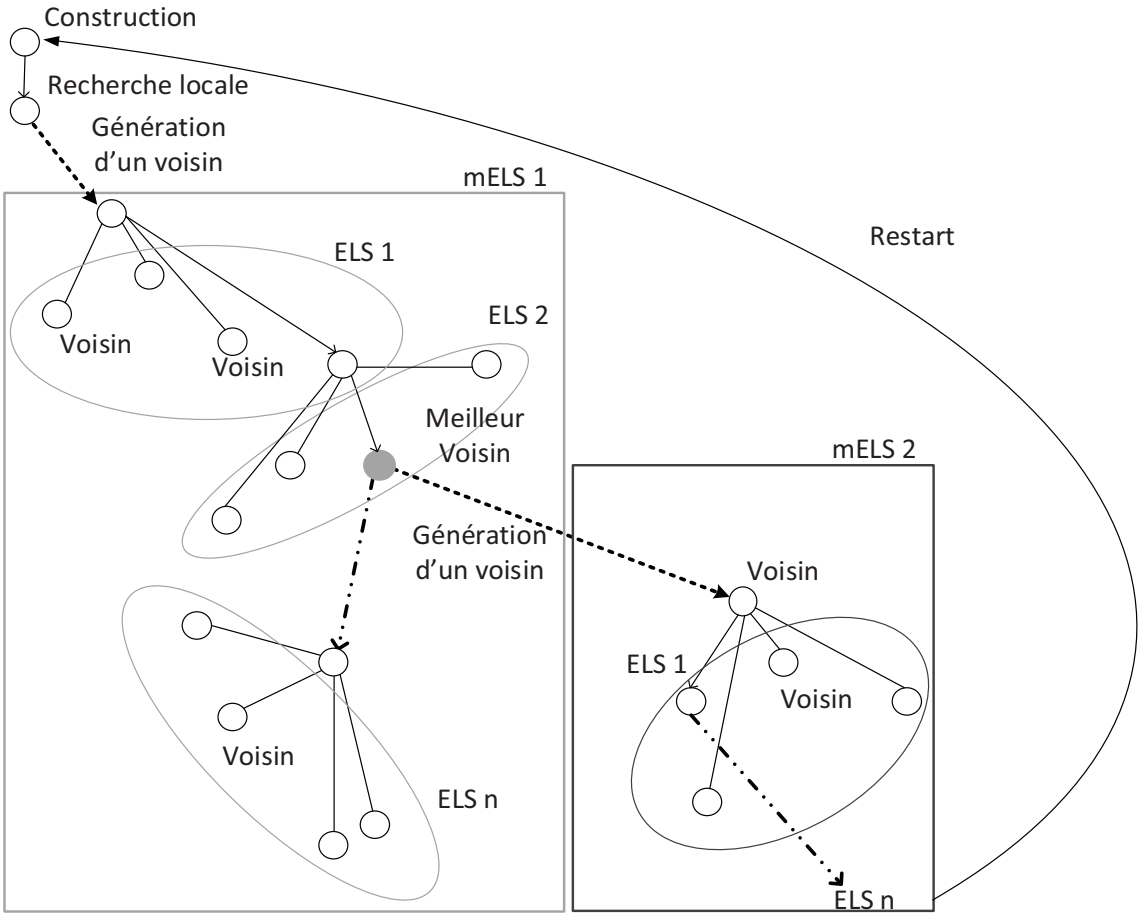

Figure 6 Représentation graphique de la métaheuristique GRASP-mELS

Dans le paragraphe suivant sont présentés les différents algorithmes appartenant aux étapes i, ii et iii mentionnées précédemment.

5.2.3.1. Phase de construction

La phase de construction du GRASP appliquée sur le Job-shop (chapitre II, §5.1.1) est étendue au problème de Job-shop Flexible. À partir de l'ensemble contenant les jobs avec toutes les opérations antérieures déjà ordonnancées, un sous-ensemble de jobs est sélectionné en respectant un critère visant à limiter l'augmentation du makespan. Soit O_i une opération qui doit être planifiée. Considérons $O_{i,v}$ l'opération O_i affectée à la machine, $v \in M_i$. L'ensemble d'opérations à planifier est noté O_a, alors que les opérations déjà planifiées sont définies dans l'ensemble O_p. À chaque itération de la phase de construction, une liste CL est construite, chaque opération de la liste étant affectée à la machine la plus prometteuse par rapport à l'évolution du makespan. On a alors :

$$CL = \{O_i \in L | C_{max}(O_i) = min_{v \in M_i}(C_{max}(O_{i,v}))\}$$

Le plus petit makespan parmi toutes les opérations dans CL est noté $\underline{C_{max}}$; le plus élevé est noté $\overline{C_{max}}$. La liste de candidats restreints (RCL) est alors définie ainsi :

$$RCL = \left\{ O_i \in CL | \underline{C_{max}} \leq C_{max}(O_i) \leq \underline{C_{max}} + \beta(\underline{C_{max}} + \overline{C_{max}}) \right\}, \beta \in [0,1]$$

L'opération suivante entrant dans le sous-ensemble O_p est ensuite choisie de manière aléatoire parmi les opérations de RCL. Toutes les informations précalculées sont sauvegardées, telles que l'affectation ou la date de début de l'opération insérée dans la séquence. L'Algorithme 2 fournit la procédure de l'heuristique de construction adaptée au Job-shop Flexible.

Algorithme 2 : Phase_de_Construction

Entrée

$data$	Structure stockant les données du problème
β	Nombre réel permettant de définir la valeur maximale h conduisant à la liste RCL

Sortie

S	Solution vide au début, contient la solution construite à la fin

Variables

$minMkpn$, $maxMkpn$	Plus petit ou plus grand makespan à chaque étape
h	Makespan maximal autorisé pour insérer une opération dans la séquence (calculé)
RCL	Liste restreinte de candidats formée des jobs retenus après calcul de h
$infinite$	Nombre entier très grand
$mkpn$	Makespan temporaire pour rechercher la meilleure machine pour une opération

Début

1. $S \coloneqq \emptyset$;
2. **POUR** $i \coloneqq 1$ **A** $data.tailleProbleme$ **FAIRE**
3. $minMkpn \coloneqq infinite$; $maxMkpn \coloneqq 0$;
4. **POUR CHAQUE** job dont une opération reste à planifier **FAIRE**
5. $mkpn \coloneqq infinite$;
6. **POUR CHAQUE** machine qui peut traiter l'opération **FAIRE**
7. Calculer chaque valeur importante pour l'opération si elle est affectée à la machine
8. (makespan, date de début,…) ;
10. **SI** le makespan est meilleur que $mkpn$ **ALORS**
11. Mettre à jour $mkpn$; sauvegarder toute valeur utile ;
12. **FIN SI**
13. **FIN POUR**
14. Mettre à jour $minMkpn$ si le makespan est plus petit ;
15. Mettre à jour $maxMkpn$ si le makespan est plus grand ;
16. **FIN POUR**
17. $h \coloneqq minMkpn + \beta * (maxMkpn - minMkpn)$;
18. Construire RCL avec les jobs dont le makespan est plus petit que h ;
19. Choisir aléatoirement un job dans RCL ;
20. Insérer le job sélectionné dans le vecteur par répétition de S ;
21. Insérer la machine dans le vecteur d'affectation de S ;
22. Sauvegarder toute valeur définitive (date de début, date de fin, …) ;
23. **FIN POUR**
24. Retourner S ;

Fin

5.2.3.2. Phase de recherche locale

Pour améliorer la qualité des solutions, il est nécessaire d'échanger l'ordre de traitement de deux opérations consécutives partageant la même machine et/ou de modifier l'affectation de la machine d'une opération donnée sur le chemin critique. En d'autres termes, il est essentiel de résoudre simultanément deux problèmes: l'affectation des opérations aux machines et l'ordonnancement des opérations. Pour le problème actuel, la recherche locale explore le chemin critique. Pour chaque opération sur le chemin, l'algorithme essaie de réaffecter l'opération à une autre machine (problème d'affectation) en utilisant la structure de voisinage $N_1^{\alpha-\rho}$, puis applique une recherche locale basée sur la structure de voisinage $N_1^{\pi-\rho}$. L'algorithme de principe de cette recherche locale est donné dans l'Algorithme 3. Dans cet algorithme, le voisinage d'une solution est exploré de deux manières: en effectuant des modifications de machine et en appliquant des permutations sur les opérations en disjonction. La ligne 1 consiste à appliquer une recherche locale spécifique basée sur la permutation des opérations d'un bloc (Recherche_Locale_Disj). L'objectif de l'utilisation de cette première recherche locale basée sur les disjonctions dans le graphe est de démarrer la recherche locale à partir d'une solution qui est un minimum local compte tenu de l'affectation actuelle. Ensuite, la recherche locale commence à explorer le chemin critique (lignes 3-19).

Algorithme 3 : Phase_de_Recherche_Locale

Entrée

$data$	Structure stockant les données du problème

Entrée/Sortie

S	Solution à améliorer

Variables

op	Opération sur le chemin critique
ok	Variable booléenne détectant si une solution a été améliorée
sav, $newS$	Solutions temporaires

Début

1. **Recherche_Locale_Disj**($data$, S) ;
2. Initialiser op à la fin du chemin critique de S ;
3. **TANT QUE** op != 0 **FAIRE**
4. **SI** (une autre machine peut être affectée à op) **ALORS**
5. $ok \coloneqq false$;
6. $sav \coloneqq S$;
7. **POUR CHAQUE** machine différente de celle actuellement affectée à op **FAIRE**
8. Changer la machine affectée à op ;
9. $ok \coloneqq$ **Recherche_Locale_Disj(**$data$, S**)** ; // ok = *true* si S est améliorée, *false* sinon
10. **SI** ok **ALORS**
11. $op \coloneqq$ dernière opération sur le chemin critique de S ;
12. **SINON**
13. $op \coloneqq$ prédécesseur de op sur le chemin critique de S ;
14. **FIN SI**
15. **FIN POUR**
16. **SINON**
17. $op \coloneqq$ prédécesseur de op sur le chemin critique de S ;
18. **FIN SI**
19. **FIN TQ**
20. Retourner S **;**

Fin

Au cours de cette exploration, la recherche locale modifie successivement l'affectation d'une opération sur le chemin critique et effectue une recherche locale sur l'ordonnancement afin de planifier efficacement toutes les opérations. Étant donné que la recherche locale (Recherche_Locale_Disj) a été appliquée au début de l'algorithme (ligne 1), seules les affectations différentes de celles actuelles sont testées pour une opération donnée. Comme la recherche locale pour l'ordonnancement est une procédure clé de la recherche locale globale, elle est présentée dans l'Algorithme 4.

Algorithme 4 : Recherche_Locale_Disj

Entrée

$data$	Structure stockant les données du problème

Entrée/Sortie

S	Solution à améliorer

Sortie

ok	Variable booléenne détectant si une solution a été améliorée

Variables

$op, prec$	Opération et son prédécesseur sur le chemin critique
$mkpn$	Makespan temporaire après estimation
$newS$	Solution temporaire
$savOp$, $savPrec$	Sauvegarde d'une opération et de son prédécesseur

Début

1. $ok := false$; $mkpn := f(S)$; //on initialise *mkpn* pour qu'il vaille celui de *S*
2. **TANT QUE** ! ok **FAIRE**
3. $op := *$; //on initialise *op* à la fin du chemin critique ;
4. **TANT QUE** op != 0 **FAIRE**
5. **SI** (op a un prédécesseur disjonctif) **ALORS**
6. $prec :=$ prédécesseur disjonctif de op ;
7. $C^e_{max} :=$ **Estimer_Cmax(**$S, prec$, op, $data$**)** ;
8. **SI** C^e_{max} est meilleur que $mkpn$ **ALORS**
9. $mkpn := C^e_{max}$;
10. $savOp := op$; $savPrec := prec$; $ok := true$;
11. **FIN SI**
12. **FIN SI**
13. $op :=$ prédécesseur de op sur le chemin critique ;
14. **FIN TQ**
15. **SI** ok **ALORS**
16. $newS := S$;
17. Appliquer la permutation de $savOp$ et $savPrec$ dans $newS$; // $N_1^{\pi-r}$
18. **Evaluer**($data$, $newS$) ;
19. **SI** $f(newS)$ est meilleure que $f(S)$ **ALORS**
20. $S := newS$; $ok := false$; $mkpn := f(S)$;
21. **FIN SI**
22. **FIN SI**
23. **FIN TQ**
24. Retourner ok ;

Fin

Dans l'Algorithme 4, la première boucle TANT QUE (lignes 2-23) explore le chemin critique du sommet terminal $*$ vers l'origine O. La deuxième boucle TANT QUE (TQ) (lignes 4-14) recherche les permutations d'opérations qui pourraient améliorer la solution. Enfin, si une permutation qui

pourrait améliorer la qualité de la solution est trouvée pendant la boucle TQ (ligne 5 à 15), cette permutation est appliquée et une évaluation est effectuée pour s'assurer de l'intégrité de la solution (ligne 16 à 23). L'algorithme d'évaluation pour le FJSP s'appuie sur un calcul de plus long chemin tel que mentionné dans le chapitre II (§4.1.2). Comme on peut le constater, beaucoup d'arcs disjonctifs peuvent être permutés lors de la recherche locale (ligne 5-14), donc l'évaluation de chacune de ces permutations pourrait prendre beaucoup de temps. Ainsi, au cours de la phase de recherche locale, chaque permutation de deux opérations consécutives sur le chemin critique qui pourrait améliorer la qualité du makespan est estimée.

Comme on peut le voir à la ligne 7 de l'algorithme 4, un appel à une procédure appelée Estimer_Cmax est effectué. Les procédures d'estimation sont efficaces pour économiser du temps de calcul et sont largement répandues dans la littérature (Dauzère-Pérès and Paulli, 1997 ; González et al., 2015 ; Palacios et al., 2015). Une fonction d'estimation a été implémentée et présentée dans l'Algorithme 5. La meilleure permutation estimée est conservée dans l'algorithme 4 (ligne 8-11) et appliquée par la suite (ligne 15-26). Cette estimation est basée sur la longueur des chemins allant de chaque opération au sommet terminal. La Figure 7 et la Figure 8 illustrent l'approche implémentée.

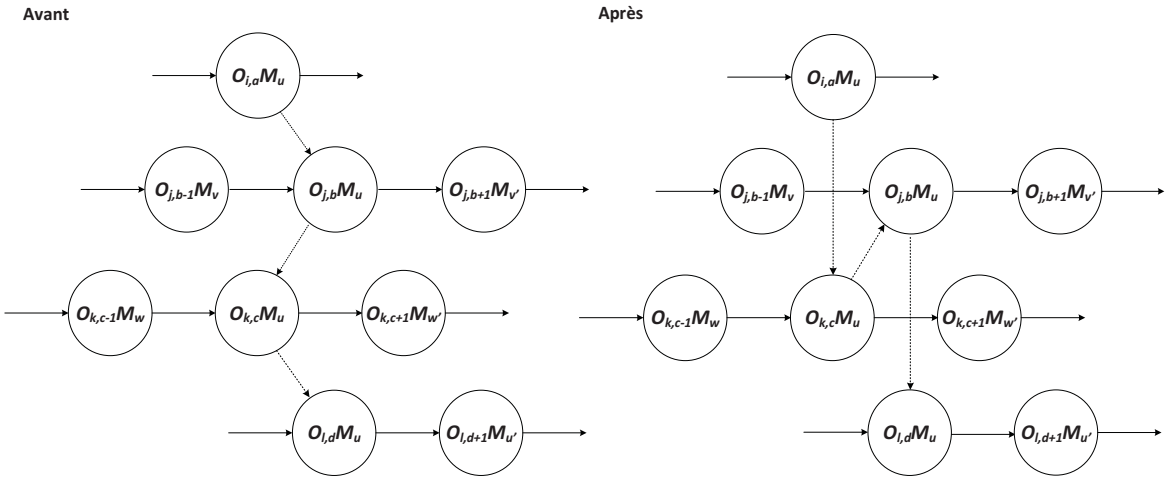

Figure 7 Exemple d'une permutation entre deux opérations sur le chemin critique

Dans la Figure 7, les opérations sont notées $O_{i,k}$ où i représente le job, et k le numéro de l'opération dans la gamme du job. Aussi, chaque opération est suivie de la machine à laquelle elle est affectée. L'objectif de la procédure d'estimation consiste à retourner une valeur estimée du makespan en considérant les chemins allant de $O_{j,b+1}$ vers le sommet terminal $*$, de $O_{k,c+1}$ vers le sommet terminal $*$ et de $O_{l,d+1}$ vers le sommet terminal $*$. En effet, dans le cas où on permute les opérations $O_{j,b}$ et $O_{k,c}$, il est possible d'obtenir la date de début au plus tôt de l'opération $O_{k,c}$ en considérant que son nouveau prédécesseur disjonctif est $O_{i,a}$, son prédécesseur conjonctif ne changeant pas. Si l'opération $O_{k,c}$ a un successeur conjonctif, la longueur du chemin allant de ce successeur vers le sommet terminal $*$ est connue. Deux concepts sont requis pour le calcul de l'estimation : la tête (head – $s_{O_{k,c}}$) et la queue (tail - $q_{O_{k,c}}$) d'une opération $O_{k,c}$. La formalisation mathématique de $s_{O_{k,c}}$ et $q_{O_{k,c}}$ est donnée dans la suite.

$s_{start} = 0$; $q_{end} = 0$;

$$s_{O_{k,c}} = max(s_{O_{k,c-1}} + p_{O_{k,c-1},\alpha(O_{k,c-1})}, s_{PM_{O_{k,c}}} + p_{PM_{O_{k,c}},\alpha(PM_{O_{k,c}})}) ;$$

$$s_{end} = max_{O_{k,c} \in PJ_{end}, v=\alpha(O_{k,c-1})}\{s_{O_{k,c}} + p_{O_{k,c},v}\} ;$$

$$q_{O_{k,c}} = max(q_{O_{k,c+1}} + p_{O_{k,c+1},\alpha(O_{k,c+1})}, q_{SM_{O_{k,c}}} + p_{SM_{O_{k,c}},\alpha(SM_{O_{k,c}})}) ;$$

$$q_{start} = max_{O_{k,c} \in SJ_{start}, v=\alpha(O_{k,c+1})}\{q_{O_{k,c}} + p_{O_{k,c},v}\} .$$

Dans ces équations, s_{start} correspond à la date de début de l'ordonnancement ; q_{end} est la longueur du chemin allant du puits à lui-même. SJ_{start} et PJ_{end} représentent respectivement les ensembles composés des premières et des dernières opérations de chaque job. SM signifie « Successeur sur la machine » (i.e.: $SM_{O_{k,c}} = O_{l,d}$ dans la Figure 7 (Avant)), et PM correspond au « Prédécesseur sur la machine ». L'obtention de la première valeur estimée, $C^e_{max}(O_{k,c})$, est illustrée sur la Figure 8(A) et est calculée comme suit :

$$s_{O_{k,c}} = max(s_{O_{k,c-1}} + p_{O_{k,c-1},M_w}, s_{O_{i,a}} + p_{O_{i,a},M_u}) ;$$

$$C^e_{max}(O_{k,c}) = s_{O_{k,c}} + p_{O_{k,c},M_u} + q_{O_{k,c+1}} + p_{O_{k,c},M_u} .$$

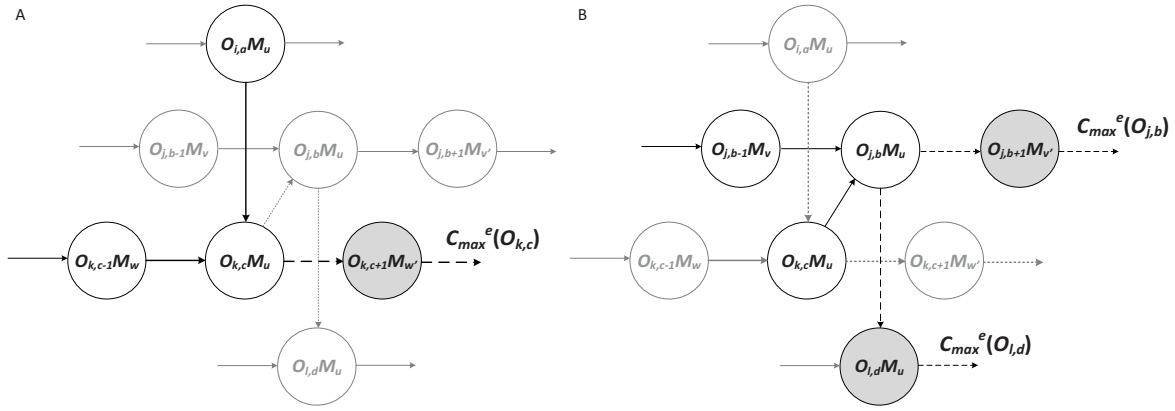

Figure 8 Exemple d'obtention du makespan estimé à partir de $O_{k,c}$ dans le cas d'une permutation

La même démarche est appliquée aux sommets $O_{j,b}$ et $O_{l,d}$ afin d'avoir un makespan estimé le plus juste possible. Ces estimations sont obtenues en calculant la longueur du chemin allant de $O_{j,b+1}$ à $*$ (i.e. $C^e_{max}(O_{j,b+1})$) et du nouveau successeur de $O_{j,b}$ à $*$ (i.e. $C^e_{max}(O_{l,d})$). Ces valeurs sont présentées sur la Figure 8(B) et calculées comme suit :

$$s_{O_{j,b}} = max(s_{O_{j,b-1}} + p_{O_{j,b-1},M_v}, s_{O_{k,c}} + p_{O_{k,c},M_u}) ;$$

$$C^e_{max}(O_{j,b}) = s_{O_{j,b}} + p_{O_{j,b},M_u} + q_{O_{j,b+1}} + p_{O_{j,b+1},M_{v'}} ;$$

$$C^e_{max}(O_{l,d}) = s_{O_{j,b}} + p_{O_{j,b},M_u} + q_{O_{l,d}} + p_{O_{l,d},M_u} ;$$

La valeur la plus élevée entre $C^e_{max}(O_{j,b})$, $C^e_{max}(O_{l,d})$ et $C^e_{max}(O_{k,c})$ est retournée en tant qu'estimation du makespan si les opérations $O_{j,b}$ et $O_{k,c}$ sont permutées.

$$C^e_{max} = max\{C^e_{max}(O_{k,c}), C^e_{max}(O_{j,b}), C^e_{max}(O_{l,d})\}.$$

Une expérience a été conduite pour évaluer la qualité de la procédure d'estimation. Une comparaison a été faite entre le makespan estimé et le makespan obtenu après la permutation. Au cours de cette expérience, 300 millions de permutations sur les opérations du chemin critique ont été effectuées sur différentes instances du problème. Les résultats montrent que dans 84% des cas le makespan estimé est égal au makespan évalué, et que dans 16% des cas il est inférieur. A aucun moment le makespan estimé n'a été supérieur au makespan réel. Par conséquent, l'algorithme d'estimation est efficace. En outre, le temps de calcul utilisé pour la recherche locale est réduit de 50% grâce à cette procédure d'estimation. Ainsi, cette procédure permet au GRASP-mELS d'explorer l'espace de solution plus longtemps compte tenu du temps gagné, ce qui a permis d'améliorer la qualité des solutions de manière significative.

Algorithme 5 : Estimer_Cmax

Entrée

S	Solution à améliorer
$opFather$, $opSon$	Opérations consécutives sur le chemin critique de S à permuter
$data$	Données du problème

Sortie

$cmax\_esti$	Makespan estimé pour la permutation de $opFather$ et $opSon$

Variables

$sDate$, $eDate$	Date de début ou date de fin estimée pour une opération
$cmaxOp$	Makespan estimé à partir de $opSon$
$cmaxOpFC$	Makespan estimé à partir de $opFather$ (arc conjonctif)
$cmaxOpFD$	Makespan estimé à partir de $opFather$ (arc disjonctif)
op	Opération temporaire

Début

1. $sDate \coloneqq$ date de début de $opSon$ en supposant que $opFather$ n'est plus son prédécesseur ;
2. $eDate \coloneqq$ date de fin de $opSon$;
3. $cmaxOp \coloneqq sDate + S.lPath[opSon]$; // estimation du makespan depuis *opSon*
4. **SI** $opSon$ a un successeur conjonctif **ALORS** // on affine l'estimation
5. $op \coloneqq$ successeur conjonctif de $opSon$;
6. $cmaxOp \coloneqq eDate + S.lPath[op]$;
7. **FIN SI**
8. $sDate \coloneqq$ date de début de $opFather$ si $opSon$ est son nouveau prédécesseur (permutation) ;
9. $eDate \coloneqq$ date de fin de $opFather$;
10. $cmaxOpFC \coloneqq sDate$; $cmaxOpFD \coloneqq sDate$;
11. **SI** $opFather$ a un successeur conjonctif **ALORS** // on affine l'estimation
12. $op \coloneqq$ successeur conjonctif de $opFather$;
13. $cmaxOpFC \coloneqq eDate + S.lPath[op]$;
14. **FIN SI**
15. **SI** $opSon$ a un successeur disjonctif **ALORS** // quand *opFather* est le prédécesseur de $opSon$
16. $op \coloneqq$ successeur disjonctif de $opSon$;
17. $cmaxOpFD \coloneqq eDate + S.lPath[op]$;
18. **FIN SI**
19. $cmax\_esti \coloneqq max(cmaxOpFD, cmaxOpFC, cmaxOp)$;
20. Retourner $cmax\_esti$;

Fin

Comme souligné précédemment, après avoir estimé la qualité de la permutation de toutes les opérations disjonctives dans le chemin critique, on applique le meilleur mouvement (ligne 17 de l'Algorithme 4). On pourrait penser qu'un tel mouvement consiste uniquement à échanger les deux jobs correspondant aux opérations en disjonction dans le vecteur par répétition. Cependant, travailler sur les permutations à l'intérieur du vecteur par répétition implique de faire plus qu'un échange simple. En raison de la renumérotation implicite des opérations à l'intérieur du vecteur par répétition, certains mouvements pourraient conduire à des résultats indésirables comme le montre la Figure 9. Dans cette figure, la première occurrence du job 3 apparaît avant la troisième occurrence de job 2 dans le vecteur, et les opérations correspondantes sont en disjonction, donc l'opération liée au job 3 est traitée avant celle du job 2. Cependant, si on permute les jobs dans le vecteur, il apparaît que la première opération du job 3 est toujours prévue avant la troisième du job 2 comme indiqué dans la deuxième partie de la Figure 9.

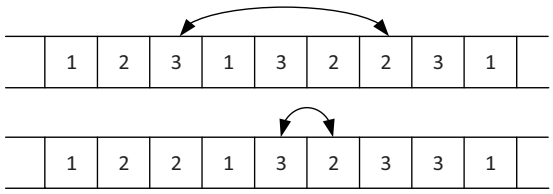

Figure 9 Exemple de permutation n'impliquant aucune modification dans le vecteur par répétition

Pour éviter ce cas, une procédure simple réorganise le vecteur entre l'indice de la première occurrence du job 3 et la troisième occurrence du job 2 en fonction des dates de début et de fin des opérations situées entre les deux qui doivent être échangées. En fait, toutes les opérations qui se terminent avant la fin de la première opération à échanger (job 3 dans la première partie de la Figure 10) sont placées avant elle et toutes les opérations qui commencent après la dernière (job 2 dans la première partie de la Figure 10) sont placées à la fin de la sous-séquence. Cela conduit à avoir les deux opérations à échanger sur la machine côte à côte dans le vecteur, comme indiqué sur la deuxième partie de la Figure 10.

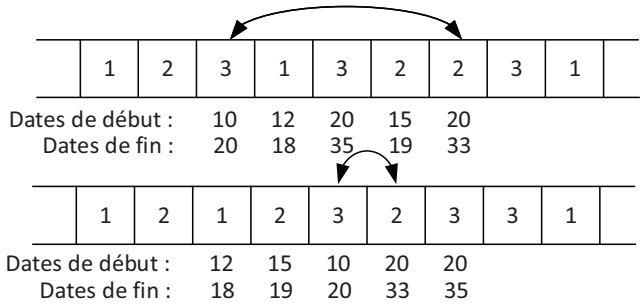

Figure 10 Réarrangement des opérations au sein du vecteur par répétition

5.2.3.3. Génération de voisins

Dans le GRASP-mELS, les voisins des solutions sont générés à deux moments: à la ligne 7 de l'Algorithme 1 et pendant la phase d'ELS. Dans cette étude, deux voisinages différents sont utilisés à cette fin. Le premier consiste à appliquer la structure N_n^{π} où n est sélectionné au hasard entre 1 et le

nombre moyen de machines par opération. Le deuxième consiste à appliquer N_n^{α} pour modifier l'affectation d'un ensemble d'opérations. Cela se fait de manière aléatoire comme dans le premier voisinage.

Ces deux structures de voisinages sont choisies de manière aléatoire en fonction de la flexibilité de l'instance du problème qui doit être résolu. La flexibilité ($flex$) correspond au nombre moyen de machines par opération dans un problème. Ensuite, un nombre réel aléatoire r est sélectionné dans [0, 20] - la valeur 20 est une valeur empirique choisie après plusieurs expérimentations - et un nombre a est calculé en multipliant un nombre aléatoire sélectionné dans [0, 1] par $flex$; si r est inférieur à a alors N_n^{α} est appliqué, sinon N_n^{π} est sélectionné. La sélection du voisinage se fait comme souligné dans l'Algorithme 6.

Algorithme 6 : Generation_Voisin

Entrée

$data$	Structure stockant les données du problème
S	Solution à modifier

Sortie

NS	Solution voisine

Variables

a, r	Nombres aléatoires

Début

1. $NS \coloneqq S$; Initialiser a et r ;
2. **SI** $r < a$ **ALORS**
3. Appliquer N_n^{α} sur NS ; // n est choisi aléatoirement dans [1, *flex*]
4. **SINON**
5. Appliquer N_n^{π} sur NS ;
6. **FIN SI**
7. Retourner NS ;

Fin

5.3. Résultats expérimentaux

Les exemples utilisés dans cette étude sont les exemples bien connus de BCdata, DPdata, BRdata et HUdata, respectivement proposés par (Chambers and Barnes, 1998), (Dauzère-Pérès and Paulli, 1997), (Brandimarte, 1993) et (Hurink et al., 1994). Le premier ensemble de données (BRdata) est un ensemble de 10 problèmes où le nombre de jobs varie de 10 à 20 et le nombre de machines indépendantes, de 4 à 15. Le nombre moyen de machines par opération (*flex*) varie entre 1,43 et 4,10.

Le deuxième ensemble de données (BCdata) est composé de 21 problèmes construits à partir de trois problèmes classiques de Job-shop (Fisher and Thompson, 1963 ; Lawrence, 1984) (mt10, la24, la40) en répliquant des machines. Les délais de traitement sont égaux entre les opérations car les machines sont censées être identiques. Le nombre de jobs varie de 10 à 15, et le nombre de machines varie de 11 à 18. La flexibilité par opération varie entre 1,07 et 1,30.

Le troisième ensemble de données (DPdata) est un ensemble de 18 problèmes difficiles. La flexibilité par opération varie entre 1,13 et 5,02 et le nombre d'opérations varie de près de 200 à 400.

Le quatrième échantillon de test (HUdata) est un ensemble de 129 problèmes divisés en trois ensembles de problèmes de test: edata, rdata et vdata. Les problèmes ont été construits à partir de trois problèmes proposés par (Fisher and Thompson, 1963) et 40 problèmes proposés par (Lawrence, 1984) en permettant à d'autres machines de traiter les opérations. Pour chaque opération, les délais de traitement sur les différentes machines sont égaux. Le premier ensemble contient les problèmes avec la moindre flexibilité (1.15), alors que la flexibilité moyenne est égale à 2 dans rdata et varie de 2.5 à 7.5 dans vdata.

Avant d'appliquer la métaheuristique à toutes les instances proposées dans ces ensembles de données, un plan d'expérience a été mené pour choisir les paramètres. Plusieurs configurations du GRASP-mELS ont été testées avec :

- Le nombre d'itérations du GRASP, nb_g, allant de 20 à 60 par pas de 20 ;
- Le nombre d'application du mELS, nb_ol, allant de 25 à 50 par pas de 5 ;
- Le nombre d'application de l'ELS, nb_els, allant de 100 à 200 par pas de 2 ;
- Le nombre de voisins générés, nb_n, allant de 10 à 40 par pas de 1.

Le plan d'expérience a conduit à la sélection des paramètres suivants

- nb_g = 50 ;
- nb_ol = 50 ;
- nb_els = 140 ;
- nb_n = 14 ;
- Le paramètre β utilisé pour générer une solution dans l'heuristique de construction est fixé à 0,33 afin d'avoir de bonnes solutions de démarrage.

Pour chaque instance, 10 réplications sont réalisées. Le temps de calcul maximal est fixé à 90 secondes pour les instances BRdata et BCdata et 300s pour les instances DPdata et HUdata afin d'avoir des conditions de calcul proches de celles proposées dans la littérature.

Dans cette section, les résultats sont comparés à des travaux récents présentant de bons résultats :

• Le CDDS de (Ben Hmida et al., 2010) ;
• Le HDE-N2 de (Yuan and Xu, 2013a) (HDE-N2 signifie "deuxième voisinage", qui est celui présentant les meilleurs résultats dans leur travail), et le HHS/LNS de (Yuan and Xu, 2013b) ;
• Le MA (Algorithme mémétique) de (González Fernández et al., 2013) ;
• Le SSPR de (González et al., 2015) ;
• Le HGTS de (Palacios et al., 2015) ;
• Le HA de (Li and Gao, 2016).

Afin d'avoir une comparaison équitable entre les méthodes, les temps de calcul sont normalisés selon le processeur utilisé en suivant la méthode énoncée dans le chapitre II (§5.2) ; la valeur du GFLOPS la plus basse est considérée dans le Tableau 3. De la même manière, lorsque la dénomination du processeur utilisé dans un article n'est pas assez précise, le GFLOPS le plus bas est sélectionné afin de ne pas désavantager les travaux correspondants.

Tableau 3 Performance et facteur de vitesse des ordinateurs utilisés pour comparaison

	GRASP-mELS.	Hmida et al., 2010	Gonzalez et al., 2013-2015	Yuan et al., 2013	Li and Gao, 2016	Palacios et al., 2015
Processeur	i7 4800MQ 2,7GHz	Core 2 duo 2,9GHz	E6750 2,66GHz	Xeon 2,83Ghz	Core 2 duo 2,0GHz	Xeon E5520
GFLOPS	3,51	2,73	2,41	2,20	1,62	2,02
Facteur de vitesse	1,00	0,78	0,69	0,63	0,46	0,58

Dans les tableaux suivants, **INS** renvoie à l'instance testée, **LB** à la borne inférieure ; ces valeurs sont celles fournies sur la page web du solveur Quintiq2 car les valeurs récentes ne sont, à notre connaissance, pas données dans la littérature. Un astérisque (*) est utilisé pour les solutions optimales. $\mathbf{S^+}$ et $\bar{\mathbf{S}}$ font référence à la meilleure solution trouvée et au makespan moyen sur plus de 10 réplications. $\overline{\mathbf{S^+_{CPU}}}$ est le temps de calcul moyen en secondes pour chaque instance, alors que $\overline{\mathbf{S_{CPU}}}$ est le temps de calcul moyen sur l'ensemble des données. Le $\overline{\mathbf{S^+_{DEV}}}$ est l'écart relatif moyen entre les meilleures solutions trouvées et la valeur **LB** en pourcentage ; $\overline{\mathbf{S_{DEV}}}$ est l'écart relatif moyen entre la moyenne du makespan par rapport à la borne inférieur. **Magnitude** est le nombre de solutions optimales trouvées. **CI-CPU** est le temps de calcul normalisé obtenu par pondération avec le facteur de vitesse du Tableau 3. Le Tableau 4 reprend les notations utilisées dans cette partie.

Tableau 4 Notations utilisées dans les tableaux

LB	Borne inférieure de l'instance testée
S^+	Meilleur makespan trouvé
$\bar{S}$	Makespan moyen
$\overline{S^+_{CPU}}$	Temps de calcul moyen pour trouver S^+
$\overline{S_{CPU}}$	Temps de calcul moyen par jeu de données
$\overline{S^+_{DEV}}$	Moyenne des déviations entre S^+ et LB
$\overline{S_{DEV}}$	Moyenne des déviations entre $\bar{S}$ et LB
Magnitude	Nombre de solutions optimales trouvées
CI-CPU	Temps de calcul normalisé

Avant d'exposer les performances du GRASP-mELS sur les instances dédiées au Job-shop Flexible, un aperçu des performances du GRASP-mELS par rapport au GRASP × ELS et aux métaheuristiques susmentionnées est présenté ci-dessous.

5.3.1. Vue d'ensemble des performances du GRASP-mELS

Pour démontrer l'efficacité du GRASP-mELS, le comportement de cette métaheuristique est comparé à la performance moyenne du GRASP×ELS et les autres métaheuristiques.

2 http://www.quintiq.com/optimization/fjssp-world-records.html

Tableau 5 Performances moyennes du GRASP×ELS, GRASP-mELS et des autres méthodes

		Performances moyennes des autres méthodes **	GRASP×ELS	GRASP-mELS
BRdata	Magnitude	8	8	8
	$\overline{S_{CPU}}$	17	13	15
	$\overline{S_{DEV}^{+}}$	0.54	1.04	0.6
	$\overline{S_{DEV}}$	0.85	1.4	0.83
BCdata	Magnitude	13	16	21
	$\overline{S_{CPU}}$	14	30	18
	$\overline{S_{DEV}^{+}}$	0.08	0.06	0
	$\overline{S_{DEV}}$	0.19	0.24	0.07
DPdata	Magnitude	2	0	3
	$\overline{S_{CPU}}$	144	190	138
	$\overline{S_{DEV}^{+}}$	1.43	2.1	1.49
	$\overline{S_{DEV}}$	1.66	2.53	1.73

** Les résultats de (Yuan and Xu, 2013b, 2013a) ne sont pas considérés car ils impactent sévèrement les temps de calcul

Comme le montre le Tableau 5, le GRASP×ELS est comparable au GRASP-mELS et à d'autres métaheuristiques concernant le nombre de solutions trouvées (Magnitude) pour les instances BRdata. Comme le GRASP×ELS utilise moins de temps durant les phases d'intensification que le GRASP-mELS, une réduction sur les temps de calcul pour des instances faciles (comme celles de BRdata) est observée. Cependant, la déviation moyenne (valeurs de RPD) par rapport aux bornes inférieures connues à ce jour est meilleure avec le GRASP-mELS. En ce qui concerne les instances BCdata, le GRASP×ELS est légèrement meilleur en moyenne que les autres métaheuristiques concernant le nombre de solutions optimales trouvées. Il est également préférable concernant la déviation moyenne par rapport à la borne inférieure, mais il est 40% plus lent que le GRASP-mELS et trouve moins de solutions optimales que ce dernier. De plus, par rapport à l'approche GRASP×ELS classique, la différence dans $\overline{S_{DEV}^{+}}$ et $\overline{S_{DEV}}$ est en faveur du GRASP-mELS dans tous les jeux de données. Enfin, le GRASP×ELS est inefficace sur les instances DPdata, pour lesquelles il ne trouve aucune solution optimale, alors que GRASP-mELS en trouve trois. Ceci valide l'approche du GRASP-mELS qui a un meilleur comportement que le GRASP×ELS pour un effort de calcul moins élevé. Dans la Figure 11, une comparaison de la convergence entre GRASP×ELS et le GRASP-mELS est donnée sur une instance de l'ensemble de données DPdata (16a). Une comparaison détaillée du GRASP-mELS avec les métaheuristiques récentes de la littérature est donnée dans la sous-section suivante.

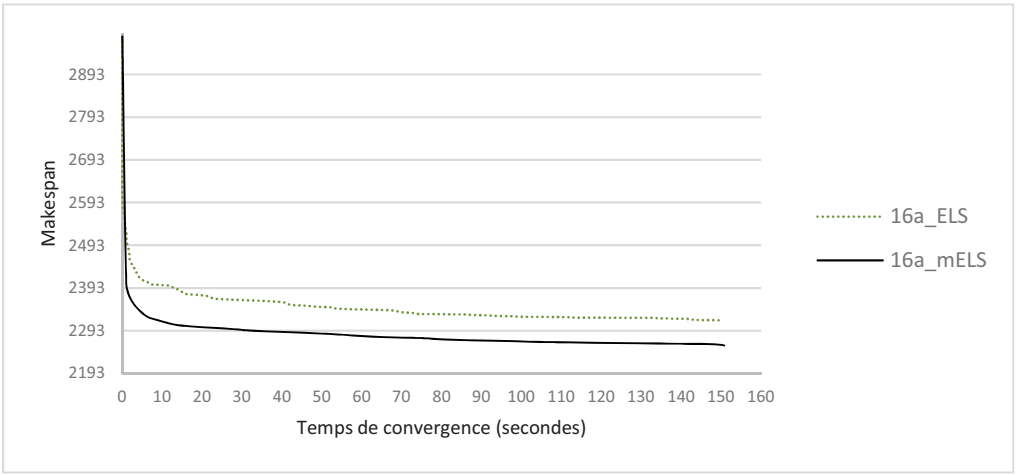

Figure 11 Comparaison de la vitesse de convergence entre le GRASP-mELS et le GRASP×ELS

5.3.2. Comparaison entre le GRASP-mELS et les méthodes de la littérature

Les résultats présentés dans le Tableau 6 montrent que la méthode fonctionne correctement sur les instances BRdata, fournissant des résultats comparables à ceux des autres métaheuristiques. En outre, les temps de calcul dans certains cas tels que Mk01, Mk02 et Mk08 sont vraiment faibles car les meilleures solutions sont presque toujours trouvées au début de l'algorithme. Parmi tous les articles, seule la méthode proposée par (Li and Gao, 2016) surpasse le GRASP-mELS dans les deux valeurs CI-CPU et Magnitude.

La méthode est beaucoup plus efficace sur les jeux de données BCdata et DPdata en terme de comportement par rapport à d'autres métaheuristiques, comme le montrent le Tableau 7 et le Tableau 8. Dans le Tableau 7, il est possible de constater que la métaheuristique est capable de fournir des solutions d'excellente qualité pour les instances BCdata. En fait, le GRASP-mELS atteint toutes les solutions optimales au moins une fois dans un temps de calcul relativement court. Il parvient à obtenir des solutions optimales pour les instances setb4xyz, seti5c12 et seti5xx. En ce qui concerne les temps de calcul normalisés, la métaheuristique est comparable aux travaux récents, même si elle n'est pas aussi rapide que le travail de (Ben Hmida et al., 2010) ou les travaux récents de (Li and Gao, 2016). Cependant, le GRASP-mELS a atteint les solutions optimales pour les 21 instances alors que les travaux mentionnés ci-dessus ont atteint respectivement 7 et 15 solutions optimales.

On constate dans le Tableau 8 que la métaheuristique est capable de fournir de bonnes solutions pour les instances de l'ensemble DPdata. Le GRASP-mELS fournit de meilleurs temps de calcul que les travaux de (Ben Hmida et al., 2010 ; Palacios et al., 2015 ; Yuan and Xu, 2013b) ou (González Fernández et al., 2013) tout en atteignant au moins autant de solutions optimales que ces travaux. Même si la métaheuristique de (Li and Gao, 2016) a de meilleurs temps de calcul, elle obtient une solution optimale de moins que le GRASP-mELS. La métaheuristique proposée a également une meilleure déviation moyenne pour les valeurs BFS et AVG que les travaux de (Ben Hmida et al., 2010 ; González Fernández et al., 2013 ; Yuan and Xu, 2013b). Seule l'application du SSPR (González et al., 2015) montre un meilleur comportement que le GRASP-mELS en moyenne sur cet ensemble de données pour le FJSP.

Pour conclure cette expérience, trois ensembles de données proposés par (Hurink et al., 1994) ont également été testés, nommés vdata, edata et rdata. Comme chacun de ces ensembles de données est composé de 43 instances, il est commun d'avoir des résultats agrégés dans la littérature. Il est donc difficile d'avoir une comparaison équitable avec des travaux présentant de bons résultats tels que les métaheuristiques de (Ben Hmida et al., 2010 ; Yuan and Xu, 2013b). Cependant, (González et al., 2015) ont fourni des résultats détaillés sur le site de leur institution, et leur travail étant à notre connaissance le meilleur sur cet ensemble de données, leurs solutions sont comparées à celles trouvées par le GRASP-mELS sur ces jeux de données. La métaheuristique présente de bons résultats sur cet ensemble de données avec de bons résultats dans un temps de calcul plus court que le SSPR en moyenne, même si ce dernier fournit de meilleures solutions sur l'ensemble d'instances vdata. La comparaison est présentée dans le Tableau 9, où l'on peut voir que l'approche est également pertinente pour cet ensemble d'instances.

Tableau 6 Résultats sur les instances proposées par (Brandimarte, 1993)

		Hmida et al., 2010			Yuan et al., 2013			Gonzalez et al., 2013			Gonzalez et al., 2015			Palacios et al., 2015			Li and Gao, 2016			Ce travail		
		CDDS			HDE-N2			GA+TS			SSPR			HGTS			HA			GRASP-mELS		
INS.	LB	S^+	$\bar{S}$	$\overline{S^+_{CPU}}$	S^+	$\bar{S}$	$\overline{S^+_{CPU}}$	S^+	$\bar{S}$	$\overline{S^+_{CPU}}$	S^+	$\bar{S}$	$\overline{S^+_{CPU}}$	S^+	$\bar{S}$	$\overline{S^+_{CPU}}$	S^+	$\bar{S}$	$\overline{S^+_{CPU}}$	S^+	$\bar{S}$	$\overline{S^+_{CPU}}$
Mk01	40*	40*	40	-	40*	40	4	40*	40	5	40*	40	11	40*	40	5	40*	-	0	40*	40	0
Mk02	26*	26*	26	-	26*	26	6	26*	26	15	26*	26	15	26*	26	15	26*	-	1	26*	26	10
Mk03	204*	204*	204	-	204*	204	31	204*	204	2	204*	204	24	204*	204	2	204*	-	0	204*	204	0
Mk04	60*	60*	60	-	60*	60	13	60*	60	10	60*	60	19	60*	60	10	60*	-	0	60*	60	0
Mk05	172*	173	174	-	172*	173	38	172*	172	18	172*	172	57	172*	172	18	172*	-	5	172*	173	15
Mk06	57*	58	59	-	57*	59	98	58	58	63	57*	58	40	57*	58	63	57*	-	54	58	58	36
Mk07	139*	139*	139	-	139*	139	26	139*	139	33	139*	141	84	139*	139	33	139*	-	20	139*	140	32
Mk08	523*	523*	523	-	523*	523	189	523*	523	3	523*	523	83	523*	523	3	523*	-	0	523*	523	0
Mk09	307*	307*	307	-	307*	307	123	307*	307	24	307*	307	52	307*	307	24	307*	-	1	307*	307	0
Mk10	189	197	198	-	198	202	266	199	200	104	196	197	94	198	199	104	197	-	33	197	199	59
$\overline{S^+_{DEV}} - \overline{S_{DEV}}$		0.66	0.94		0.48	1.1		0.7	0.76		0.37	0.74		0.48	0.71		0.42	-		0.60	0.83	
Magnitude		7			9			8			9			9			9			8		
$\overline{S_{CPU}}$		15			79			28			48			28			11			15		
CI-CPU		12			50			19			33			16			5			15		

* solutions optimales

Tableau 7 Résultats sur les instances proposées par (Chambers and Barnes, 1998)

		Hmida et al., 2010			Yuan et al., 2013			Gonzalez et al.,			Gonzalez et al.,			Palacios et al.,			Li and Gao.,			Ce travail		
		CDDS			HDE-N2			GA+TS			SSPR			HGTS			HA			GRASP-mELS		
INS.	LB	S^+	$\bar{S}$	$\overline{S^+_{CPU}}$	S^+	$\bar{S}$	$\overline{S^+_{CPU}}$	S^+	$\bar{S}$	$\overline{S^+_{CPU}}$	S^+	$\bar{S}$	$\overline{S^+_{CPU}}$	S^+	$\bar{S}$	$\overline{S^+_{CPU}}$	S^+	$\bar{S}$	$\overline{S^+_{CPU}}$	S^+	$\bar{S}$	$\overline{S^+_{CPU}}$
mt10c1	927*	928	929	-	927*	928	179	927	927	14	927*	928	26	927*	927	13	927*	-	12	927*	927	8
mt10cc	908*	910	911	-	908*	911	180	908	909	14	908*	908	20	908*	910	13	908*	-	10	908*	909	17
mt10x	918*	918*	918	-	918*	919	179	918	922	18	918*	918	23	918*	918	15	918*	-	11	918*	918	2
mt10xx	918*	918*	918	-	918*	918	170	918	918	16	918*	918	19	918*	918	12	918*	-	11	918*	918	2
mt10xx	918*	918*	918	-	918*	918	160	918	918	19	918*	918	20	918*	918	12	918*	-	11	918*	918	2
mt10xy	905*	906	906	-	905*	906	174	905	905	16	905*	906	21	905*	905	13	905*	-	11	905*	905	26
mt10xy	847*	849	851	-	847*	851	166	849	850	21	847*	847	20	847*	850	18	847*	-	9	847*	847	26
setb4c9	914*	919	919	-	914*	917	338	914	914	22	914*	916	28	914*	914	16	914*	-	15	914*	914	11
setb4cc	907*	909	911	-	907*	910	336	907	907	22	907*	907	21	907*	908	15	907*	-	15	907*	907	29
setb4x	925*	925*	925	-	925*	926	354	925	925	18	925*	925	19	925*	925	15	925*	-	13	925*	925	4
setb4xx	925*	925*	925	-	925*	926	330	925	925	19	925*	925	21	925*	925	14	925*	-	5	925*	925	2
setb4xx	925*	925*	925	-	925*	926	315	925	925	20	925*	925	22	925*	925	15	925*	-	9	925*	925	3
setb4xy	910*	916	916	-	910*	914	313	910	910	25	910*	912	32	910*	910	19	910*	-	12	910*	910	18
setb4xy	902*	905	907	-	903	905	317	905	905	19	905	905	21	905	905	15	905	-	14	902*	904	11
seti5c12	1169	1174	117	-	1171	117	111	117	117	41	1170	117	25	1170	117	41	1170	-	31	1169	117	39
seti5cc	1135	1136	113	-	1136	113	107	113	113	40	1135	113	29	1136	113	34	1136	-	17	1135	113	24
seti5x	1198	1201	120	-	1200	120	108	119	120	43	1198	119	41	1199	120	38	1198	-	27	1198	199	36
seti5xx	1194	1199	119	-	1197	120	125	119	119	38	1197	119	37	1197	119	34	1197	-	29	1194	119	26
seti5xxx	1194	1197	119	-	1197	120	124	119	119	40	1194	119	38	1197	119	31	1197	-	19	1194	119	27
seti5xy	1135	1136	113	-	1136	113	114	113	113	39	1135	113	29	1136	113	34	1136	-	17	1135	113	28
seti5xyz	1125	1125	112	-	1125	113	122	112	112	41	1125	112	35	1125	112	43	1125	-	33	1125	112	42
$\overline{S^+_{DEV}} - \overline{S_{DEV}}$		0.19	0.26		0.05	0.3		0.08	0.14		0.03	0.12		0.07	0.13		0.05	-		0	0.07	
Magnitude		7			13			12			18			14			15			21		
$\overline{S_{CPU}}$		15			555			26			26			22			16			18		
CI-CPU		12			349			18			18			13			7			18		

* solutions optimales

Tableau 8 Résultats sur les instances proposées par (Dauzère-Pérès and Paulli, 1997)

INS.	LB	Hmida et al., 2010			Yuan et al., 2013			Gonzalez et al., 2013			Gonzalez et al., 2015			Palacios et al., 2015			Li and Gao., 2016			Ce travail		
		CDDS			HDE-N2			GA+TS			SSPR			HGTS			HA			GRASP-mELS		
		S^+	$\bar{S}$	$\overline{S^+_{CPU}}$	S^+	$\bar{S}$	$\overline{S^+_{CPU}}$	S^+	$\bar{S}$	$\overline{S^+_{CPU}}$	S^+	$\bar{S}$	$\overline{S^+_{CPU}}$	S^+	$\bar{S}$	$\overline{S^+_{CPU}}$	S^+	$\bar{S}$	$\overline{S^+_{CPU}}$	S^+	$\bar{S}$	$\overline{S^+_{CPU}}$
01a	2505	2518	252	-	2505	251	838	2505	251	74	2505	250	68	2505	250	122	2505	-	108	2505	250	62
02a	2228	2231	223	-	2230	223	973	2232	223	120	2229	223	100	2230	223	205	2230	-	133	2229	223	86
03a	2228	2229	223	-	2228	222	116	2229	223	143	2228	222	110	2228	223	181	2229	-	97	2228	223	94
04a	2503	2503	251	-	2506	250	850	2503	250	72	2503	250	57	2503	250	112	2503	-	87	2503	250	31
05a	2192	2216	221	-	2212	221	931	2219	222	123	2211	221	112	2214	221	208	2212	-	116	2212	221	126
06a	2163	2196	220	-	2187	219	116	2200	220	157	2183	219	181	2193	219	260	2197	-	93	2195	220	181
07a	2216	2283	229	-	2288	230	154	2266	228	201	2274	228	139	2270	228	344	2279	-	204	2276	228	127
08a	2061	2069	206	-	2067	207	190	2072	207	197	2064	206	181	2070	207	318	2067	-	184	2069	207	144
09a	2061	2066	206	-	2069	207	943	2066	206	291	2062	206	213	2067	206	376	2065	-	201	2069	207	170
10a	2212	2291	230	-	2297	230	159	2267	227	240	2269	228	120	2247	226	369	2287	-	238	2263	227	110
11a	2018	2063	207	-	2061	206	182	2068	207	222	2051	205	193	2064	206	294	2060	-	181	2065	206	170
12a	1969	2031	203	-	2027	203	914	2037	204	266	2018	202	280	2027	203	486	2027	-	151	2039	204	148
13a	2197	2257	226	-	2263	226	290	2271	227	241	2248	225	119	2250	226	416	2248	-	293	2252	226	158
14a	2161	2167	217	-	2164	216	323	2169	217	340	2163	216	269	2170	217	396	2167	-	210	2170	217	191
15a	2161	2165	217	-	2163	216	211	2166	216	470	2162	216	376	2168	216	523	2163	-	192	2172	217	173
16a	2193	2256	225	-	2259	226	280	2266	227	253	2244	225	131	2246	225	384	2249	-	160	2243	225	151
17a	2088	2140	214	-	2137	214	309	2147	215	333	2130	213	299	2142	214	483	2140	-	203	2145	215	190
18a	2057	2127	213	-	2124	212	248	2138	214	488	2119	212	409	2129	213	650	2132	-	133	2146	215	164
$\overline{S^+_{DEV}} - \overline{S_{DEV}}$		1.55	1.8		1.5	1.73		1.59	1.77		1.18	1.4		1.34	1.59		1.43	-		1.49	1.73	
Magnitude		1			2			2			3			3			2			3		
$\overline{S_{CPU}}$		200			1738			235			187			340			166			138		
CI-CPU		156			1095			162			129			197			76			138		

* solutions optimales

Tableau 9 Résultats sur les instances proposées par (Hurink et al., 1994)

	edata				**rdata**				**vdata**			
	GRASP-mELS		SSPR		GRASP-mELS		SSPR		GRASP-mELS		SSPR	
	$\overline{S^{+}_{DEV}}$	$\overline{S_{DEV}}$	$\overline{S^{+}_{DEV}}$	$\overline{S_{DEV}}$	$\overline{S^{+}_{DEV}}$	$\overline{S_{DEV}}$	$\overline{S^{+}_{DEV}}$	$\overline{S_{DEV}}$	$\overline{S^{+}_{DEV}}$	$\overline{S_{DEV}}$	$\overline{S^{+}_{DEV}}$	$\overline{S_{DEV}}$
mt06/10/20	0	0	0	0.04	0	0	0	0	0	0	0	0
la01-la05	0	0	0	0	0	0.07	0.07	0.09	0	0	0	0
la06-la10	0	0	0	0	0	0	0	0.01	0	0	0	0
la11-la15	0	0	0	0	0	0	0	0	0	0	0	0
la16-la20	0	0	0	0	0	0	0	0.03	0	0	0	0
la21-la25	0	0.24	0.08	0.23	2.63	3.27	2.53	2.91	0.49	0.8	0.23	0.35
la26-la30	0.33	0.61	0.43	0.66	0.36	0.71	0.36	0.48	0.17	0.24	0.06	0.08
la31-la35	0.05	0.11	0.01	0.07	0.05	0.12	0.04	0.05	0.04	0.07	0.01	0.02
la36-la40	0	0.04	0	0.05	0.36	1.22	0.66	0.90	0	0	0	0
Magnitude	38		36		25		24		30		34	
CI-CPU	21		26		47		32		28		44	

Pour résumer, 178 instances du FJSP ont été considérées. Comme le montrent les résultats, aucune méthode de la littérature n'est meilleure sur toutes les instances ce qui corrobore les travaux de (Wolpert and Macready, 1997). Le GRASP-mELS a atteint la meilleure solution connue pour 125 de ces instances. Ce résultat est comparable à celui du SSPR de (González et al., 2015) qui obtient 124 solutions optimales. Si l'on regarde la somme des CI-CPU, le GRASP-mELS met en moyenne 267 secondes pour résoudre l'ensemble des jeux de données, tandis-que le SSPR met 282 secondes. Le GRASP-mELS est donc comparable à la meilleure méthode de l'ensemble des articles présentés dans ce chapitre.

Pour tous les ensembles de données testés, les résultats détaillés et la structure des solutions optimales obtenues sont disponibles en ligne (http://damienlamy.com). Des améliorations sur les temps de calcul, la qualité des solutions et la constance de la métaheuristique, en particulier pour les instances avec de nombreuses machines disponibles pour les opérations, pourraient être explorés dans les futures études. En effet, la métaheuristique s'est généralement révélée la plus compétitive sur les cas où le nombre moyen de machines par opération est relativement réduit, ce qui invite à explorer la piste d'autres voisinages plus adaptés aux instances impliquant un nombre moyen de machines par opérations élevé.

6. Conclusion

Dans ce chapitre certains outils utilisés pour résoudre le problème de Job-shop Flexible ont été mis en évidence, et un aperçu des méthodes récentes employées a été proposé. Du fait de la structure du problème, certains outils mis en œuvre dans ces articles proviennent du problème de Job-shop, tandis que d'autres sont dédiés au Job-shop Flexible.

Une métaheuristique pour traiter le problème du Job-shop Flexible a également été proposée. Elle repose sur un schéma de type GRASP, qui est amélioré pour fournir de meilleures solutions ce

qui débouche sur une nouvelle métaheuristique nommée GRASP-mELS. Le but de cette métaheuristique est d'explorer davantage le voisinage des solutions en exécutant des ELS sur des mutations d'une solution. Deux voisinages sont proposés pour la partie mELS et sont intégrés dans une procédure d'exploration de voisinage plus globale qui sélectionne l'un ou l'autre de manière aléatoire. Deux autres structures de voisinages sont utilisées pendant l'algorithme de recherche local. Une procédure d'estimation est également mise en œuvre pour accélérer la phase de recherche locale. La métaheuristique ne manipule qu'une solution à la fois et fournit des résultats de bonne qualité à la fois en termes de temps de calcul et en termes de qualité du critère de performance des solutions, ce qui montre la pertinence d'une telle approche parmi toutes les autres métaheuristiques qui reposent principalement sur des populations.

Chapitre V : Couplages Optimisation/Simulation pour l'ordonnancement

Dans ce chapitre des méthodes conjointes de résolutions mettant en œuvre la simulation et l'optimisation sont décrites. L'objectif est double : définir des ordonnancements robustes dans le cadre d'un Job-shop Stochastique, et proposer une approche réactive au problème de Job-shop Flexible. Le module d'optimisation utilisé dans chaque cas repose sur une métaheuristique, tandis que la simulation est gérée par un module reposant sur l'environnement de simulation ARENA.

Sommaire

Liste des figures

Liste des tableaux

Liste des algorithmes

1. Introduction

Comme il a été énoncé dans le chapitre I, cette thèse s'inscrit dans le cadre du projet industriel ECOTHER et d'un partenariat entre le laboratoire IRCCyN et le LIMOS. Le projet vise à réduire les consommations énergétiques des systèmes de production par la mise en place de méthodes innovantes de gestion de production. Plus précisément, l'objectif consiste à évaluer la performance énergétique des ordonnancements qui peuvent être appliqués sur le système de production. Il s'agit de proposer un couplage Optimisation/Simulation permettant d'apporter à un industriel des informations sur la performance énergétique de l'ordonnancement mis en œuvre au sein des différents ateliers d'une usine en prenant en compte les aléas. La notion de couplage Optimisation/Simulation apparaît sous différentes formes dans la littérature. En effet, la littérature sur les problèmes d'ordonnancement les classe généralement en fonction de leur structure. Parmi les termes employés, il est fait mention de problèmes déterministes ou stochastiques, de problèmes statiques ou dynamiques ou encore de offline ou online (Abedinnia et al., 2017).

Dans les problèmes dits déterministes, l'ensemble de l'information nécessaire à la résolution du problème est connu en amont de toute phase d'optimisation, et aucune évolution de ces informations n'est considérée dans le temps. Les problèmes stochastiques vont plus loin en prenant en compte des données dont les valeurs numériques suivent des lois de probabilités afin de modéliser des systèmes incertains. Ces deux types de problèmes appartiennent aux problèmes statiques : même si les données peuvent avoir des valeurs incertaines, la structure du problème (e. g. le nombre d'opérations à ordonnancer dans le cadre d'un problème d'ordonnancement) reste inchangée au fil du temps. Ceci est en opposition avec les problèmes dynamiques, où l'ensemble de l'information n'est pas connu avant le processus d'optimisation, et où des ordres de fabrication peuvent arriver au fur et à mesure. Dans un problème dynamique (le terme de réactif est également utilisé), il faut redéfinir l'ordonnancement lorsqu'un événement aléatoire est considéré, alors que ce n'est pas le cas dans un problème statique. De manière générale, des règles de gestion sont appliquées dans un contexte dynamique car elles sont plus faciles à mettre en œuvre sur un système réel et elles nécessitent peu de temps de calcul. Ainsi, si dans les problèmes statiques l'ordonnancement n'a pas vocation à évoluer dans le temps, ce n'est pas le cas dans un environnement dynamique où l'ordonnancement initial (s'il a été défini) ne sera probablement pas identique à l'ordonnancement qui aura été réellement appliqué une fois que tous les ordres de fabrication auront été traités. Les problèmes *online* et *offline* sont mentionnés, cependant une ambiguïté subsiste quant à la différence entre statique/offline et dynamique/online même si (Lee et al., 2010) considèrent deux types de problèmes onlines : « online over list » et « online over time ». Dans le premier les jobs sont ordonnés en liste alors que dans le second les jobs arrivent aléatoirement à différents moments de l'horizon de temps. Un diagramme récapitulatif (Figure 1) inspiré de (Lin et al., 2012) donne la hiérarchie derrière les problèmes classiques énoncés ci-dessus. Ce diagramme permet d'avoir une vue d'ensemble des problèmes et, bien qu'il ne soit pas exhaustif, il donne les classes de problèmes présentés dans la section suivante.

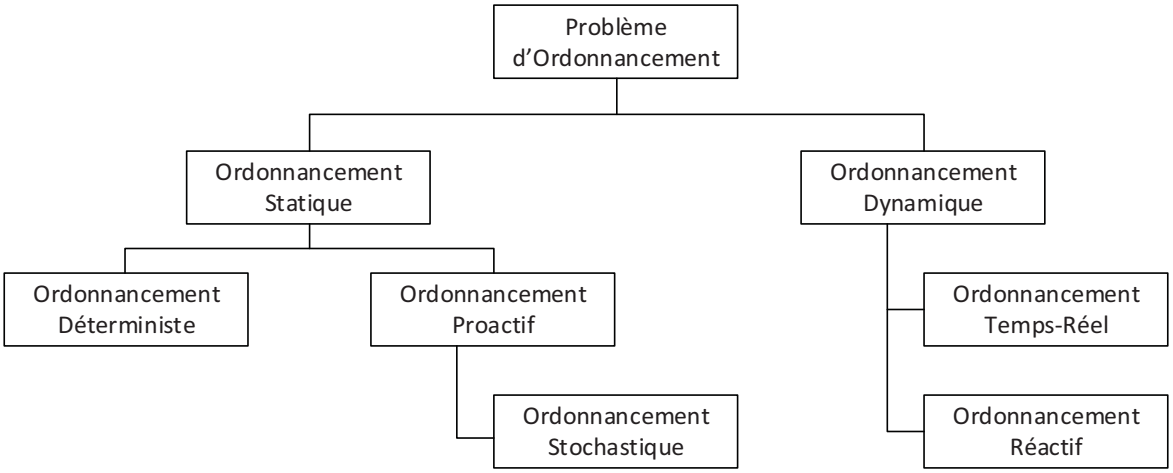

Figure 1 Classification des problèmes d'ordonnancement

La plupart du temps, les études menées dans la littérature se concentrent sur l'aspect déterministe des problèmes de planification et d'ordonnancement. Dans un tel contexte, toutes les données sont connues avant tout processus d'optimisation. Cependant, il est extrêmement difficile de prévoir le comportement des systèmes de production dans le monde réel car ils ne sont pas déterministes mais exposés à des perturbations qui peuvent affecter les objectifs à atteindre. Ces perturbations peuvent être des pannes sur les machines, des aléas dans les temps de transport, des opérations dont les durées varient, et bien d'autres. Il existe deux manières de prendre en compte ces problématiques lors d'un processus d'optimisation : en intégrant l'aspect incertain lors de la résolution afin de proposer une solution « robuste » et dont l'application est supposée faire face aux événements aléatoires pouvant survenir (Juan et al., 2014) ; ou proposer des ajustements de la solution au fur et à mesure que l'information arrive dans le système d'information (nouvel ordre de fabrication, machine défaillante, …) (Gholami and Zandieh, 2009). On parle d'optimisation stochastique dans le premier cas et d'optimisation réactive dans le second. Comme souligné dans les travaux de (Abedinnia et al., 2017; Horng et al., 2012), moins d'études sont réalisées sur des systèmes de production stochastiques que déterministes alors que les décideurs ont besoin d'outils efficaces pour gérer correctement la production dans un contexte incertain.

Étant donné que le laboratoire est impliqué dans un projet qui vise à définir des outils combinant le logiciel de simulation ARENA et des méthodes d'optimisation, le travail réalisé dans ce chapitre vise à proposer une manière d'intégrer les deux approches. Ainsi, dans la continuité des travaux présentés dans les chapitres précédents, le Job-shop stochastique où des temps de traitement aléatoires sont associés à chaque opération est considéré. L'objectif est de proposer un ordonnancement avec le plus petit makespan à l'aide d'une méthode de résolution qui consiste à intégrer un modèle de simulation au sein de la métaheuristique. Tout d'abord, la méthode d'optimisation explore l'espace des solutions afin de trouver une bonne solution dans un contexte déterministe, où les temps de traitement sont donc constants. Cette solution est ensuite utilisée en tant que scénario par le module de simulation. Ce dernier n'applique plus des durées déterminées, mais des temps de traitement aléatoires afin d'obtenir la performance simulée de la solution. Le processus est ensuite répété afin de trouver différentes solutions qui pourraient être moins bonnes du point de vue déterministe, mais meilleures du point de vue de la robustesse. À la fin du processus, l'algorithme renvoie la meilleure solution trouvée.

Dans la section suivante, un ensemble d'articles se concentrant sur les aspects stochastiques ou réactifs des systèmes de production est présenté. Dans la troisième section, le problème du Job-shop stochastique est introduit où un couplage optimisation/simulation est présenté afin d'obtenir des ordonnancements robustes. La quatrième section consiste en un modèle de simulation pour un Job-shop Flexible réactif où les prises de décisions sont faites par un appel à un module d'optimisation. Enfin, la dernière partie consiste en une conclusion générale.

2. Présentation des problèmes d'optimisation-simulation

Dans la littérature, de nombreux problèmes d'ordonnancement appartiennent à la classe des ordonnancements proactifs ou dynamiques. Il est possible de mentionner différents articles traitant de systèmes de production très variés. Par exemple, (Sevaux and Sörensen, 2004) abordent un problème à une machine pour lequel les dates de disponibilités des jobs sont aléatoires. (Juan et al., 2014; Rahmani and Ramezanian, 2016) traitent de problèmes de Flow-shop, stochastique dans le premier cas et réactif dans le deuxième. De nombreux travaux sont également menés sur les problèmes de Job-shop et Job-shop Flexible (Gholami and Zandieh, 2009; Mahdavi et al., 2010; Zhang and Wu, 2011).

(Juan et al., 2014) étudient un Flow-shop à permutation où les temps de traitement des opérations suivent une loi de probabilité. Leur approche combine simulation et métaheuristique (simheuristics). L'hypothèse au cœur de cette approche se base sur le fait qu'il est probable que les solutions déterministes de bonne qualité sont a fortiori de bonnes solutions stochastiques. Les auteurs espèrent notamment découvrir des solutions finales de bonne qualité qui ne soient pas forcément des solutions optimales du problème déterministe. Ils explorent donc l'espace des solutions déterministes et évaluent ces solutions avec un module de simulation. Des simulations avec un nombre de réplications faible (deux cents réplications) afin d'estimer le potentiel des solutions sont effectuées au sein de la métaheuristique. Si une solution présente un meilleur makespan simulé, alors elle est sauvegardée. La métaheuristique réitère l'ensemble du processus en essayant de déterminer une solution différente de la meilleure solution courante qui pourrait être de meilleure qualité. La qualité de la meilleure solution explorée est ensuite affinée à l'aide d'une simulation avec un nombre de réplications plus conséquent (un million) à la fin de la métaheuristique.

(Golenko-Ginzburg and Gonik, 2002) considèrent un Job-shop stochastique où chaque opération a une durée aléatoire basée sur son espérance et sa variance. Chaque job a une date d'échéance et tout retard se voit sanctionné par deux pénalités ; la première est applicable une seule fois si le job est en retard, tandis que la seconde est journalière. Lorsqu'un job est terminé avant sa date d'échéance, des coûts de stockage sont considérés. L'objectif est de déterminer la date de début au plus tôt de chaque job. Un modèle de simulation est proposé pour résoudre le problème. Lorsque des conflits interviennent, tels que deux opérations devant être traitées par la même machine et qu'il faut les départager, une règle de décision est appliquée. Cette règle est basée sur des comparaisons entre les opérations après calculs de certains critères, tels que la probabilité qu'ont les jobs de finir dans les temps s'ils sont sélectionnés, ou la probabilité que le job concurrent finisse dans les temps. Une méthode de descente est utilisée en plus de la simulation afin de déterminer les dates de début au plus tôt des jobs afin de réduire les coûts liés aux retards. Les méthodes de descente sont des méthodes itératives dans lesquelles à chaque itération un objectif est minimisé de manière approchée en faisant varier certaines valeurs d'un vecteur tout en gardant d'autres fixées (Wright, 2015).

(Tavakkoli-Moghaddam et al., 2005) s'intéressent à un problème de Job-shop stochastique dans lequel les durées des opérations sont randomisées. Ils cherchent à optimiser un problème biobjectif visant à minimiser les coûts associés à l'utilisation des ressources (usinage et inactivité) ainsi que les retards par rapport aux dates d'échéance des jobs. Une méthode de résolution en deux étapes est proposée, basée sur un réseau de neurones et sur un recuit simulé. La première a pour objectif de trouver une solution réalisable qui est ensuite améliorée à l'aide du recuit simulé.

(Gu et al., 2009) proposent un algorithme génétique pour le problème de Job-shop stochastique en considérant des temps de traitement suivant une loi normale. Leur objectif est d'obtenir des solutions minimisant l'espérance mathématique du makespan. Ce critère est calculé à l'aide de simulations effectuées au sein de la métaheuristique. Par la suite, le problème est de nouveau étudié par (Gu et al., 2010) qui ont proposé des modifications par rapport à la précédente métaheuristique afin d'en améliorer les performances.

(Lei, 2011) propose un algorithme génétique simple pour le problème du Job-shop stochastique avec pannes sur les machines. Les temps de traitement suivent une loi exponentielle. L'objectif à atteindre est la minimisation du makespan. Lorsqu'une panne intervient lors du traitement d'un job alors celui-ci doit être entièrement recommencé sur la machine. Les pannes machines sont gérées lors du décodage de la séquence et la procédure associée prend également en compte la réparation de la machine incriminée lors d'une panne et l'usinage de l'opération interrompue.

(Zhang and Wu, 2011) considèrent des temps de traitement randomisés dans le contexte d'un Job-shop stochastique dans le but de minimiser le retard total. Une métaheuristique à population est développée dans leur travail. Différentes populations sont générées au cours des itérations de la métaheuristique. Afin de déterminer la valeur de la fonction objectif de ces solutions, des simulations sont effectuées. Elles requièrent un nombre de réplications qui est déterminé dynamiquement en fonction de la qualité des solutions par rapport aux réplications déjà effectuées. Certaines solutions sont directement supprimées lorsque leur fonction objectif est de moins bonne qualité que la solution qu'elles devraient remplacer dans la population (dont la fonction objectif a été obtenue après simulation). Ces approches sont intéressantes et permettent de réduire les temps de calcul dans un contexte où la simulation nécessite de nombreuses réplications pour que les résultats soient statistiquement significatifs.

(Horng et al., 2012) proposent un algorithme évolutionnaire afin de réduire la somme attendue des coûts de stockage et les pénalités liées aux retards dans le cadre d'un Job-shop stochastique avec des durées des opérations qui suivent une loi probabiliste. Une fois que l'algorithme évolutionnaire a déterminé une population acceptable, un nombre d'individus est sélectionné parmi les meilleurs de la population. Dans un premier temps, une estimation de la fonction objectif de chaque individu est faite avec un nombre de réplications faible. Un sous-ensemble des meilleurs individus à ce stade est construit. La procédure est réitérée avec ce nouveau sous-ensemble en augmentant le nombre de réplications afin d'améliorer la précision de l'estimation. A chaque itération, le nombre de réplications augmente. Lorsque la procédure est terminée, le meilleur candidat est retourné à la fin de l'algorithme. Le principe du processus proposé est similaire à celui de (Zhang and Wu, 2011). En effet, il affine au fur et à mesure la qualité de l'estimation de la fonction objectif ce qui en fait un moyen efficace de limiter les temps de calcul tout en proposant de bonnes solutions.

(Hao et al., 2015) s'intéressent à un problème de Job-shop stochastique avec des temps de traitement variables. L'objectif est de minimiser le makespan moyen et le retard moyen. Une

métaheuristique multiobjectif à population est proposée, ainsi qu'une procédure de recherche locale. Le calcul des critères de performance se fait par une procédure au sein de la métaheuristique.

En ce qui concerne la simulation, un travail intéressant est celui de (Nawara and Hassanein, 2013), dont l'objectif est la résolution d'un problème de Job-shop à l'aide du logiciel de simulation ARENA. Les temps de traitement des opérations sur les machines sont connus à l'avance (déterministe) et l'ordre de traitement des opérations sur une machine est géré de manière aléatoire. En faisant de nombreuses réplications les auteurs arrivent à obtenir des solutions de bonne qualité. Leur approche peut donc être considérée comme de l'optimisation par simulation. Cependant, en raison de la structure du modèle appliqué, et à cause du moteur de simulation et de la façon dont les files d'attentes sont gérées, certains ordonnancements ne peuvent être obtenus, et par conséquent, des solutions pertinentes peuvent être ignorées. Par exemple, la Figure 2(B) montre une solution non atteignable.

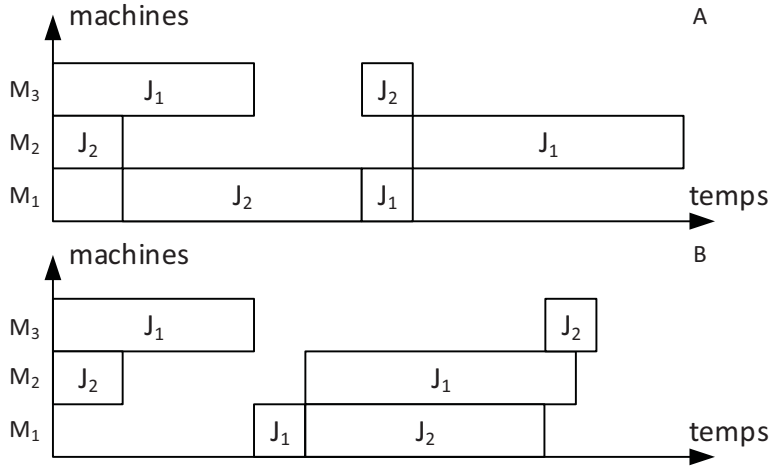

Figure 2 Ordonnancement probable (A) et non atteignable (B) dans le cadre d'une simulation

Dans la Figure 2(A), le job J_2 atteint la file d'attente de la machine M_1 avant le Job J_1, qui n'a pas fini son traitement sur la machine M_3. Quelle que soit la configuration testée, il n'est pas possible d'obtenir la solution exposée dans la Figure 2(B), alors qu'elle est de meilleure qualité concernant le makespan.

(Adibi et al., 2010) étudient un problème de Job-shop dynamique dans lequel les jobs arrivent de manière aléatoire et les machines sont soumises à des pannes. Les auteurs mesurent deux critères de performance, le makespan et le retard par rapport à des dates d'échéance au sein d'une fonction objectif agrégée. Le problème est traité par l'utilisation d'un réseau de neurones (ANN) et d'une métaheuristique de type recherche par voisinage variable (VNS). Un réseau de neurones est une structure composée de différents niveaux. Chaque niveau est composé d'un ensemble de nœuds et prend ses entrées sur les nœuds du niveau précédent. Des poids sont associés à chaque nœud et des fonctions internes au réseau permettent de produire à partir des données en entrée, une ou plusieurs valeurs en sortie. Les auteurs utilisent l'ANN afin de définir les paramètres utilisés dans la procédure VNS en se basant sur l'apprentissage effectué sur différents problèmes. Parmi les paramètres utilisés en entrée de l'ANN, le nombre de jobs ou encore le temps moyen par opération peuvent être mentionnés. Lorsqu'un nouvel événement (apparition d'une panne ou d'un nouveau job) est observé

sur le système, un problème statique est construit à partir des données actuelles. Le VNS, dont les paramètres sont donnés par l'ANN en fonction du problème construit, a alors pour rôle de définir un nouvel ordonnancement afin de respecter au mieux les objectifs à optimiser.

(Sharma and Jain, 2014) étudient de nombreux objectifs (makespan, retard moyen, retard total, …) et développent un modèle de simulation d'un Job-shop dynamique et stochastique. En plus du problème classique, des durées de préparation (setup) des machines sont considérées, ces durées dépendant de la séquence des opérations sur les machines. Dans le problème traité, les jobs arrivent au fur et à mesure, et les durées de traitement des opérations ainsi que les dates d'échéance suivent des lois de probabilité. Un modèle de simulation est développé au sein d'un environnement de simulation dédié. De nombreuses règles de gestion sont évaluées au sein du modèle parmi lesquelles les règles "premier arrivé premier servi", "opération la plus courte en premier", etc… Les auteurs ont pu déterminer grâce à ces simulations que les règles les plus performantes dépendent des objectifs à optimiser. Cependant, deux règles permettent à elles seules d'optimiser l'ensemble des objectifs. La première vise à favoriser dans la file d'attente un job du même type que celui déjà traité par la machine, ou à choisir le job dont la date d'échéance est la plus critique. La seconde consiste à sélectionner en priorité le job pour lequel le temps de configuration de la machine est le plus court.

(Mahdavi et al., 2010) s'intéressent à un Job-shop Flexible stochastique où les opérations ont des durées aléatoires et où les objectifs sont la minimisation du makespan, du temps d'attente et la maximisation de l'utilisation des machines. Ils notent que l'aspect incertain du problème entraîne l'impossibilité de trouver une solution optimale. Selon eux, une méthode efficace pour accroître la performance d'un système de production consiste à développer un modèle de simulation qui correspond aux objectifs fixés. En effet, la simulation est plus adaptée à des environnements dynamiques que des algorithmes d'approximation ou l'utilisation de chaînes de Markov, qui sont moins efficaces dans la manipulation des systèmes flexibles de production et dans le traitement de problèmes de taille importante. Une approche basée sur le couplage simulation et optimisation est proposée. Des échanges sont effectués entre les deux méthodes afin de converger vers un niveau de performance acceptable. Les auteurs proposent également une méthode « temps-réel ». Le module d'aide à la décision définit les règles de gestion à employer lorsque des événements interviennent.

(Al-Hinai and ElMekkawy, 2011) proposent une approche en deux étapes pour un problème de Job-shop Flexible considérant des pannes de machines. L'approche est basée sur un algorithme génétique. Lors de la première étape une population d'individus est construite à l'aide d'un premier algorithme génétique avec pour unique objectif le makespan. Ensuite, un deuxième algorithme a pour objectif d'optimiser le problème biobjectif maximisant la robustesse et la stabilité des solutions de la population. Les auteurs notent que les solutions trouvées par un algorithme minimisant le makespan sont généralement denses et compactes et de ce fait sensibles aux perturbations. Les auteurs cherchent donc des solutions de bonne qualité présentant des durées d'inactivité tout en préservant la structure semi-active des solutions.

(Mokhtari and Dadgar, 2015) étudient un Job-shop Flexible avec considération de tâches de maintenances. Dans le problème étudié, les machines peuvent subir des pannes et les temps de traitement et les dates d'échéance des jobs sont aléatoires. L'objectif à atteindre est la minimisation du nombre de jobs en retard en incluant dans l'ordonnancement des tâches de maintenances préventives. Ce type de maintenance est très important pour éviter de se retrouver avec un parc de machines où certaines ressources sont indisponibles et où les jobs s'accumulent dans les files d'attente. Ils développent un outil d'optimisation et de simulation basé sur une métaheuristique de type recuit

simulé, les métaheuristiques étant les méthodes d'optimisation les plus adaptées pour un couplage avec les outils de simulation. Dans leur approche, le recuit simulé propose une solution au module de simulation qui retourne la fonction objectif probable associé à la solution. La métaheuristique utilise l'information obtenue afin de rechercher des solutions plus pertinentes.

(Gholami and Zandieh, 2009) s'intéressent à un Job-shop Flexible dynamique. Leur objectif consiste à minimiser le makespan et le retard moyen tout en considérant des pannes pouvant arriver sur les machines. Les auteurs considèrent que le temps moyen entre les pannes ainsi que le temps moyen de réparation suivent une loi exponentielle. Une fois une machine réparée, l'opération qui était exécutée dessus peut être poursuivie. Du fait de la complexité inhérente au problème, une métaheuristique reposant sur un algorithme génétique est utilisée. La prise en compte des événements aléatoires est faite par l'intégration d'un simulateur au sein de la métaheuristique. L'algorithme génétique est arrêté après deux cents générations de populations sans amélioration. Le simulateur évalue la valeur d'une solution en reconstruisant l'ordonnancement après avoir pris en compte les différentes pannes. Plus précisément, il effectue le calcul de la date de fin de traitement des jobs basé sur l'ordre de leur passage sur les machines et en prenant en compte les temps moyen inter-pannes. Cette procédure est appliquée vingt fois pour obtenir une estimation fiable de la fonction objectif.

Il est possible de remarquer, à partir des articles présentés ci-dessus, que de nombreux travaux sont faits sur les problèmes d'ordonnancements stochastiques et/ou dynamiques. Pour la définition d'ordonnancements de qualité, un nombre important d'heuristiques et de métaheuristiques sont utilisées conjointement avec des modules de simulation pour déterminer de bonnes solutions aux problèmes dont l'environnement est incertain. Dans ce contexte l'utilisation d'une métaheuristique, le GRASP×ELS, dont le déroulement diffère des métaheuristiques observées dans la littérature sur les problèmes stochastiques est proposée. Cette métaheuristique est utilisée conjointement à un module de simulation. L'objectif est d'explorer l'espace des solutions déterministes et de réduire le nombre de simulations qui permettent d'obtenir une estimation de la qualité d'une solution pour le problème de Job-shop stochastique avec durée aléatoire de traitement des opérations. Ce travail vise donc à proposer une approche intégrée, combinant optimisation et simulation, ce type d'approche étant largement répandu dans la littérature. Une telle approche appartient à la classe des approches d'« optimisation avec itérations basées sur la simulation » (Optimization with Simulation-based Iterations - OSI). Pour une taxonomie plus détaillée sur les couplages optimisation simulation, on peut se référer à l'article de (Figueira and Almada-Lobo, 2014) d'où est issue l'appellation OSI.

Dans l'approche proposée, l'algorithme d'optimisation est conçu pour explorer l'espace des solutions déterministes, alors qu'un modèle de simulation fournit des informations sur la robustesse des ordonnancements explorés. A la différence de (Juan et al., 2014) le nombre de réplications de la simulation est constant et le choix est fait d'appliquer la simulation à la fin de chaque ELS de la métaheuristique. Le choix s'est porté vers l'utilisation d'un module de simulation reposant sur un langage ayant fait ses preuves : SIMAN, et l'environnement de simulation implémentant ce langage : ARENA. L'utilisation de logiciels commerciaux en lieu et place d'applications « maisons » présente évidemment des avantages et des inconvénients (Cardin, 2007). La réduction de la durée de développement et la représentation de ces logiciels dans l'industrie comptent parmi les principaux avantages. En contrepartie, les coûts d'acquisition et de maintenance de la solution sont élevés et la personne qui développe un modèle de simulation par le biais d'outils externes perd la main sur le système qui devient en partie une « boîte noire ». La gestion des événements simultanés peut conduire à des solutions différentes en fonction de l'outil de simulation utilisé. Un autre avantage d'une telle approche, lié à la rapidité de développement, repose sur la possibilité de faire évoluer le modèle de

simulation indépendamment du module d'optimisation, en ayant la possibilité d'apporter des modifications au modèle de simulation sans modifier ou devoir adapter l'outil d'optimisation. Par exemple, il est possible d'ajouter des transporteurs qui ne seraient pas pris en compte dans le module d'optimisation afin d'obtenir des informations sur le système (il est en effet courant d'intégrer les durées de transport dans les temps de traitement en optimisation).

En d'autres termes, il est proposé de résoudre une relaxation du problème au sein d'un module d'optimisation et d'enrichir la qualité de la solution en utilisant un module de simulation en support qui va retourner la valeur simulée de la solution déterministe obtenue. Une structure de donnée qui permet de mettre en œuvre une mémoire à long terme est considérée. Elle garde une trace, au sein du module d'optimisation, des solutions déjà simulées. Les simulations sont effectuées sur des solutions qui sont ensuite utilisées pour comparaison dans le module d'optimisation dont le rôle est d'utiliser la connaissance acquise afin de déterminer de nouveaux ordonnancements à simuler. En faisant l'hypothèse que les solutions déterministes fournies au module de simulation peuvent être de bonnes solutions du problème général, les approches d'optimisation/simulation prennent tout leur sens.

3. Proposition d'une approche pour le Job-shop stochastique (Kemmoé-Tchomté et al., 2015b)

Dans cette section une méthode de résolution couplant optimisation et simulation pour le Job-shop stochastique est proposée. Les temps de traitement des opérations suivent une loi aléatoire. L'objectif est de définir des ordonnancements prédictifs à même de résister aux différents aléas sur les temps de traitement des opérations. Après un rappel du problème, la méthode de résolution est présentée et des résultats expérimentaux sont donnés.

3.1. Présentation du problème

Le problème de Job-shop stochastique est une extension du problème classique qu'est le Job-shop. Pour rappel, il a été vu que le problème de Job-shop consiste à ordonnancer un ensemble de jobs qui doivent être séquencés sur des machines. Chaque job est composé d'un ensemble d'opérations qui doivent être traitées dans un ordre spécifique et chaque opération doit être traitée par une machine donnée. Dans le Job-shop classique le temps de traitement de chaque opération est connu avant tout processus d'optimisation. Un des objectifs les plus courants pour le problème de Job-shop consiste à trouver un ordonnancement minimisant la durée totale de traitement de tous les jobs. Cet objectif, le makespan, est notamment une mesure de la productivité du système de production.

Dans sa forme classique, le Job-shop est un problème déterministe. Cependant, comme il a été montré dans la section précédente, les systèmes de production sont soumis à des aléas qui peuvent les affecter. Un problème de Job-shop prenant en compte ces aléas est appelé Job-shop stochastique. Ces événements incertains peuvent toucher entre autres les machines, les temps de transport ou les temps de traitement des opérations. Dans cette étude, des durées de traitement qui suivent une loi de probabilité sont considérées. Cependant, la résolution est divisée en deux parties afin de considérer également une résolution du problème déterministe. La première partie correspond à la phase d'optimisation. Cette phase est effectuée au sein d'un module d'optimisation dédié qui traite des temps de traitement déterministes. La seconde partie consiste en une phase de simulation qui utilise des temps de traitement aléatoires pour les opérations. Les temps de traitement suivent une distribution

normale dans le module de simulation. Ainsi, si les temps de traitement aléatoire sont notés p_i, leurs valeurs peuvent être exprimées comme suit dans le modèle de simulation proposé dans ce travail :

$$p_i^r \approx Normal(p_i, \frac{p_i}{2})$$

Dans cette formule, l'espérance modélisant la durée moyenne d'une opération a une valeur égale à la durée déterministe utilisée dans les jeux de données de la littérature. La variance est égale à la moitié de la durée. Ce choix vise à considérer des opérations avec une forte variabilité dans les durées de traitement. Dans la section suivante, le cadre global de résolution mis en place, consistant à combiner l'optimisation et la simulation pour proposer des solutions pour le Job-shop stochastique est présenté. L'objectif est de déterminer un ordonnancement robuste, c'est-à-dire capable de résister aux aléas susceptibles d'apparaître au cours de la production, en alternant entre les phases d'optimisation et de simulation.

3.2. Approche Optimisation-Simulation

Il est important de noter que la plupart des problèmes industriels, comme la planification ou le transport, ne peuvent bien souvent pas être résolus analytiquement en raison de leur complexité. Si les modèles mathématiques sont utilisés pour obtenir des solutions exactes, leur efficacité diminue avec la taille des problèmes à traiter. En plus de ces problèmes, les systèmes semblent ne pas être déterministes, et sont soumis à la présence de facteurs aléatoires tels que des temps de traitement incertains ou des pannes de machines. Ainsi, la simulation s'est propagée pour évaluer les performances de ces systèmes complexes, où la recherche de solutions exactes pourrait être impossible et prend beaucoup de temps, alors que des solutions de bonne qualité peuvent être suffisantes. La simulation peut être utilisée à différents stades de la conception d'un système de production comme souligné dans (Pritsker et al., 1997). En effet, elle peut être utilisée pour comprendre un système ou un problème, analyser et déterminer des éléments, des composants ou des problèmes critiques, synthétiser et évaluer les solutions proposées, prévoir et planifier les développements futurs, ou être intégrée dans un environnement dont le comportement doit être surveillé et faire des prédictions pour les décideurs. En outre, la simulation a l'avantage de pouvoir être utilisée sur des systèmes sans les construire dans le cas de propositions ; ou sans les déranger dans le cas de systèmes coûteux ou pour lesquels des expériences pourraient être dangereuses ; et sans les détruire si le but de l'expérience est d'évaluer les limites du système. Compte tenu de ces avantages, la simulation est largement utilisée dans l'industrie, en ce qui concerne les opérations de planification et d'ordonnancement ainsi que pour la planification des ressources humaines ou des transports, dans le but d'obtenir un aperçu de la performance de ces systèmes. Cependant, la simulation classique est appliquée aux scenarii et n'est donc pas adaptée à la meilleure conception du système (Mokhtari and Dadgar, 2015). Par conséquent, la combinaison de la simulation avec l'optimisation permet d'optimiser une fonction objectif soumise à des événements aléatoires.

Dans le type d'approche proposée, le module d'optimisation cherche les meilleurs paramètres utilisés, l'ordre de passage des opérations sur les machines dans le cas présent, comme valeurs d'entrée dans le module de simulation, tandis que le module de simulation évalue les performances du système conçu avec ces paramètres et fournit des valeurs de sortie estimées.

Dans cette section, un cadre de résolution pour déterminer un ordonnancement robuste basé sur une approche en deux étapes est introduit. L'approche repose sur un couplage entre une métaheuristique et un modèle de simulation. Alors que la métaheuristique vise à trouver des solutions déterministes de bonne qualité, le modèle de simulation renvoie la qualité estimée de la solution fournie par la métaheuristique. Ce critère de performance consiste en une estimation du makespan moyen lorsque l'on considère des temps de traitement aléatoires pour les opérations. Ainsi, la solution est sauvegardée par la mémoire à long terme. Enfin, après itérations successives de la phase d'optimisation et de simulation, l'outil renvoie la solution la plus robuste explorée au cours du processus. Une représentation graphique de l'approche est présentée sur la Figure 3.

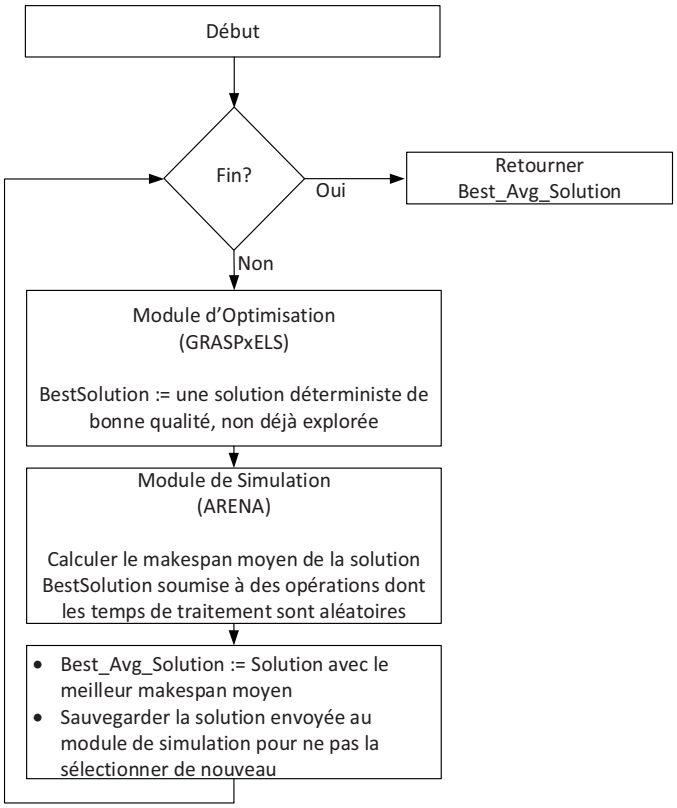

Figure 3 Vue d'ensemble du processus de résolution

Il est possible de voir sur la Figure 3 que la première étape consiste à trouver une solution déterministe (module d'optimisation) de bonne qualité à l'aide d'une métaheuristique. Cette solution est ensuite considérée comme une entrée du modèle de simulation qui renvoie le makespan moyen prévu pour l'ordonnancement proposé. La meilleure solution après la phase de simulation est stockée si elle présente une meilleure robustesse (un makespan moyen plus petit que la solution précédemment sauvegardée). Aussi, chaque solution simulée est sauvegardée afin de ne pas être réutilisée par le module de simulation. L'algorithme s'arrête après avoir atteint un nombre prédéfini d'itérations ou une limite dans le temps de calcul (test <Fin ?>).

Comme la structure de l'algorithme est clairement divisée en deux phases, les sous-sections suivantes suivent cette décomposition: la métaheuristique adaptée au couplage est présentée en premier, puis le modèle de simulation est détaillé.

3.2.1. Module d'optimisation

Au vu des performances du GRASP×ELS sur le problème de Job-shop et sur l'extension traitée dans les chapitres précédents, le choix est fait de continuer avec cette métaheuristique pour le module d'optimisation, l'objectif étant davantage orienté vers le principe de résolution. L'algorithme proposé pour le GRASP×ELS adapté au problème stochastique est donné dans l'Algorithme 1. Dans cet algorithme, quelques différences notables sont observées par rapport au GRASP×ELS classique énoncé dans le chapitre I (§5.2.4.5). Par exemple, la meilleure solution rencontrée au cours du processus d'exploration de l'espace des solutions n'est pas automatiquement conservée lors de la génération des voisins (ligne 12). Une nouvelle solution temporaire est considérée : S'''. Elle permet de conserver la meilleure solution actuellement trouvée lors de la phase d'ELS. De plus, les solutions obtenues au cours de la génération de voisins ne sont conservées que si elles n'ont pas déjà été utilisées au sein du module de simulation. Toute solution simulée au cours du processus de résolution (ligne 24) est donc sauvegardée. L'objectif est de ne pas simuler des ordonnancements déjà évalués et préserver ainsi le temps de calcul alloué aux phases de simulation.

Algorithme 1 : GRASP×ELS_Stochastique

Sortie
S^* Meilleure solution rencontrée par l'algorithme
Variables
S, S', S'', S''' Solutions temporaires

Début
1. $f_{AVG}(S^*) \coloneqq \infty$;
2. **TANT QUE** critère d'arrêt non satisfait **FAIRE**
3. $S \coloneqq$ solution produite par une heuristique de construction randomisée ;
4. $S \coloneqq$ solution retournée par une recherche locale ;
5. $f(S''') \coloneqq \infty$;
6. **POUR** $iterELS \coloneqq 1$ **A** $maxELS$ **FAIRE** // boucle 1
7. $f(S'') \coloneqq \infty$;
8. **POUR** $iterVoisin \coloneqq 1$ **A** $maxVoisin$ **FAIRE** // boucle 2
9. $S' \coloneqq$ solution dans le voisinage V de S;
10. $S' \coloneqq$ solution retournée par une recherche locale ;
11. **SI** $f(S') < f(S'')$ **ET** S' non déjà simulée **ALORS**
12. $S'' \coloneqq S'$;
13. **FIN SI**
14. **FIN POUR**
15. $S \coloneqq S''$;
16. **SI** $f(S) < f(S''')$ **ALORS**
17. $S''' \coloneqq S$;
18. **FIN SI**
19. **FIN POUR**
20. Simuler(S''') ; // simulation faite avec ARENA (SIMAN)
21. **SI** $f_{AVG}(S''') < f_{AVG}(S^*)$ **ALORS**
22. $S^* \coloneqq S'''$;
23. **FIN SI**
24. Sauvegarder S''' ;
25. **FIN TQ**
26. Retourner S^* ;

Fin

Les phases classiques que sont la construction d'une solution, la recherche locale, et la génération des voisins sont retrouvées. Une description rapide de ces points est faite dans les parties suivantes.

3.2.1.1. Phase de construction

L'approche suggérée dans le chapitre II (§5.1.1) visant à construire une séquence d'opérations en insérant une opération à chaque itération est reprise.

3.2.1.2. Phase de recherche locale

Comme la phase de construction produit rarement un optimum local, le rôle de la recherche locale est d'augmenter la qualité d'une solution donnée en explorant le voisinage de cette solution. La recherche locale développée au chapitre II (§5.1.2) pour le problème de Job-shop déterministe est utilisée. Elle consiste à remonter le chemin critique et à effectuer des permutations sur les arcs disjonctifs.

3.2.1.3. Phase de mutation

La procédure de mutation consiste à permuter deux opérations liées à différents jobs dans la séquence par répétition utilisée pour coder une solution. Cette procédure simple mais ayant prouvé son efficacité est préservée au sein de la métaheuristique utilisée.

3.2.1.4. Mémoire à long terme

La particularité du GRASP×ELS développé réside dans l'ajout d'une mémoire à long terme qui a pour objectif de garder une trace des solutions simulées. Il existe plusieurs manières de sauvegarder le fait qu'une solution a déjà été visitée. La première consiste à calculer une signature de la solution évaluée et d'avoir une table de hachage qui permet de savoir en un accès si la signature d'une solution a déjà été observée. Une façon simpliste consiste à sommer les dates de début au carré des opérations du problème, et à effectuer une congruence modulo le nombre de cellules de la table de hachage. Cependant, une telle approche conduit bien souvent à des collisions qui peuvent nuire au comportement attendu, à savoir simuler des solutions différentes. Le choix se porte donc sur l'utilisation d'une *map* qui stocke directement le vecteur contenant les dates de début des opérations. Pour ce faire l'opérateur de comparaison utilisé par la *map* est redéfini. Le choix du vecteur des dates en lieu et place du vecteur par répétition se justifie par le mapping associé à la représentation indirecte d'une solution (cf. chapitre I, §4.3.2) qui implique que plusieurs vecteurs peuvent représenter la même solution. Comme l'objectif est de simuler des solutions explicitement différentes du point de vue de leurs structures, le vecteur des dates est plus approprié. Ainsi, si une solution a déjà été évaluée par le module de simulation alors le vecteur des dates de début (déterministes) des opérations est contenu dans la *map*.

3.2.2. Module de simulation

Dans cette étude, le modèle de simulation est construit à l'aide du langage de programmation SIMUL ANALYSIS (SIMAN) proposé par (Pegden, 1983). Le langage SIMAN est connu pour faire partie du logiciel Arena édité par la société Rockwell Automation. Un modèle de simulation sous Arena requiert deux fichiers : un fichier '*experiment*' et un autre '*model*'. Le premier contient

l'information complète sur le système modélisé : nombre de machines, nombre de files d'attentes, nombre de réplications à effectuer, etc. … Le second fichier contient la description précise du système avec les différentes interactions entre les éléments. Ces fichiers sont propres à chaque problème simulé. Ainsi, si un problème contient cinq machines et un autre dix, les fichiers seront différents. Étant donné que, dans cette étude, le modèle est utilisé en dehors de l'environnement Arena, puisqu'il va être appelé par le module d'optimisation, les fichiers *experiment* et *model* qui sont obligatoires pour l'exécution de la simulation sont générés automatiquement en fonction de la taille des instances. Cette procédure n'est pas décrite ici. Cependant, la simulation en dehors d'Arena nécessite un fichier spécifique qui s'obtient après compilation successive du fichier *experiment* et *model*. L'obtention de ce fichier, d'extension « .p » est décrit dans la Figure 4.

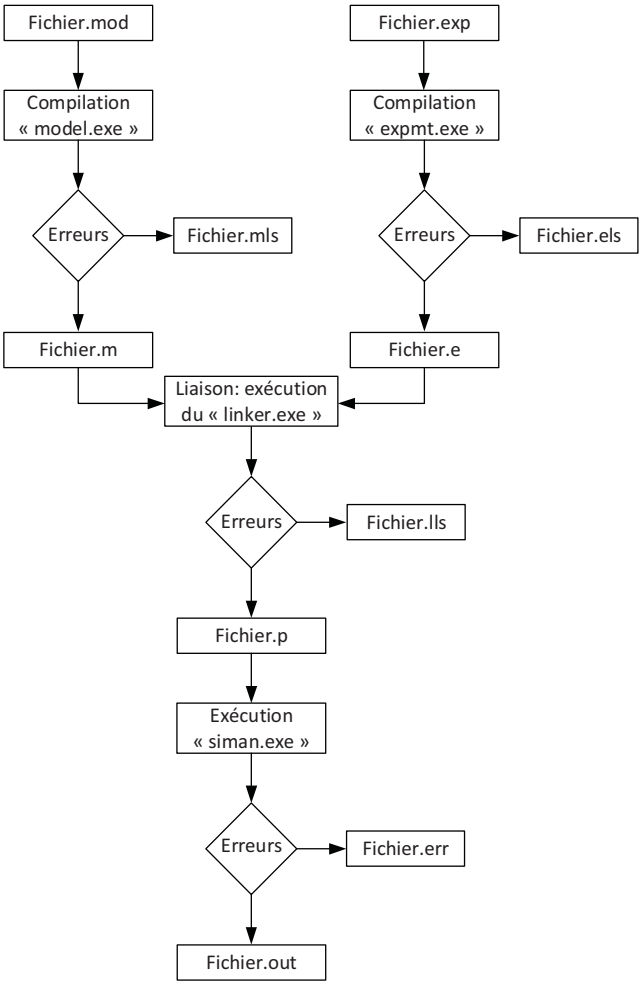

Figure 4 Diagramme d'obtention d'un fichier simulable "Fichier.p"

La Figure 4 fait apparaître l'application successive de plusieurs exécutables ('model.exe', 'expmt.exe', 'linker.exe') en vue d'obtenir le fichier terminal simulable. A toutes les étapes des fichiers d'erreurs peuvent être générés, il faut donc n'avoir aucune erreur pour pouvoir lancer la simulation. Aussi, la génération du fichier Fichier.p peut nécessiter un certain temps. Cependant, comme ce fichier ne dépend que des données du problème, une fois généré, aucune intervention supplémentaire n'est requise, il est donc créé une unique fois avant le lancement de toute procédure d'optimisation. Une

fois que le fichier Fichier.p existe, il est possible de lancer la simulation à l'aide de l'exécutable 'siman.exe'.

Le bon fonctionnement du module de simulation repose sur les informations associées à la solution simulée qui lui sont envoyées. Le principe généralement appliqué lors de la définition de modèles de simulation est l'utilisation de règles de gestion prédéfinies (Nawara and Hassanein, 2013; Sharma and Jain, 2014) qui affectent les entités (les jobs) aux ressources (les machines), telle que « premier arrivé premier servi », etc… Afin de déterminer la robustesse d'une solution, l'objectif est de simuler spécifiquement la solution obtenue lors de la phase d'optimisation, en respectant l'ordre de passage des opérations sur les machines. Un ensemble d'informations contenant le numéro du job, la machine (aussi appelée station) et la durée déterministe de chaque opération est donc transmis au module de simulation, ainsi que l'ordre de passage de chaque opération sur la machine. L'ordonnancement de chaque job est envoyé au module de simulation sous la forme d'une structure de données comme indiqué dans la Figure 5.

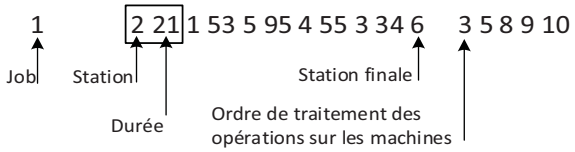

Figure 5 Exemple d'information envoyée au module de simulation

Dans la Figure 5, on visualise l'ensemble des informations transmises. Une station supplémentaire, modélisant le nœud de sortie (machine fictive numéro 6) est également ajoutée. C'est la dernière partie de la ligne qui est plus particulièrement importante. En effet, il peut être vu ici que la première opération du premier job (1) doit être traitée en troisième sur la station 2 (i.e. la deuxième machine), ce qui suggère que deux autres jobs vont être traités avant. Le modèle de simulation doit donc empêcher le traitement d'une opération dont le « ticket de passage » ne correspondrait pas à l'opération qui doit passer sur la machine.

Les fichiers *model* et *experiment* générés et permettant la simulation ne sont pas présentés. La construction graphique du modèle sous ARENA, correspondant aux fichiers générés, permet une meilleure compréhension et est présentée par la suite. Le modèle développé pour le Job-shop stochastique est donné dans la Figure 6. Cette figure montre les différents blocs et éléments utilisés pour modéliser un système de production de Job-shop. Dans la partie Elements, l'élément Projet (*Project*) définit les propriétés du projet telles que son nom ou les statistiques collectées, tandis que l'élément Réplication (*Replicate*) est utilisé pour définir le nombre de réplications pour la simulation. L'élément Entités (*Entities*) est utilisé pour définir le type d'entités créées tout au long de la simulation (un seul type ici) et l'élément Fichiers (*Files*) définit le nom du fichier d'entrée lu, ainsi que le fichier de sortie dans lequel peuvent être stockées certaines valeurs précises (temps moyen de sortie du système pour chaque job par exemple). L'élément Attributs (*Attributes*) définit les différentes valeurs attachées à une entité tel que *JobType* qui stocke le numéro du job de chaque ligne dans le fichier d'entrée et l'*opStage* qui correspond à l'étape de traitement d'une opération. Les Variables (*Variables*) définissent les différentes tables utilisées pour stocker les données accessibles par tout objet du système, on y trouve notamment le tableau utilisé pour compter le nombre de jobs traités par chaque machine (*status_M*) et le tableau permettant de stocker le numéro de passage, pour une opération donnée, sur la machine affectée. Les ressources (*Resources*) sont les différentes machines nécessaires au problème. L'élément « *Queues* » définit les files d'attentes situées devant les machines, tandis que l'élément

« *Stations* » permet de concevoir des routages permettant de passer d'une station à une autre. Enfin, l'élément « *Sets* » permet de représenter plusieurs stations du système étudié sans avoir besoin de les représenter toutes, ce qui apporte de la généricité au modèle de simulation. En effet, comme le montre la partie « Blocks », le nombre de stations définies est limité car la station modélisant la machine *i* permet de représenter toutes les stations. Les éléments sont définis dans le fichier *experiment* mentionné plus haut.

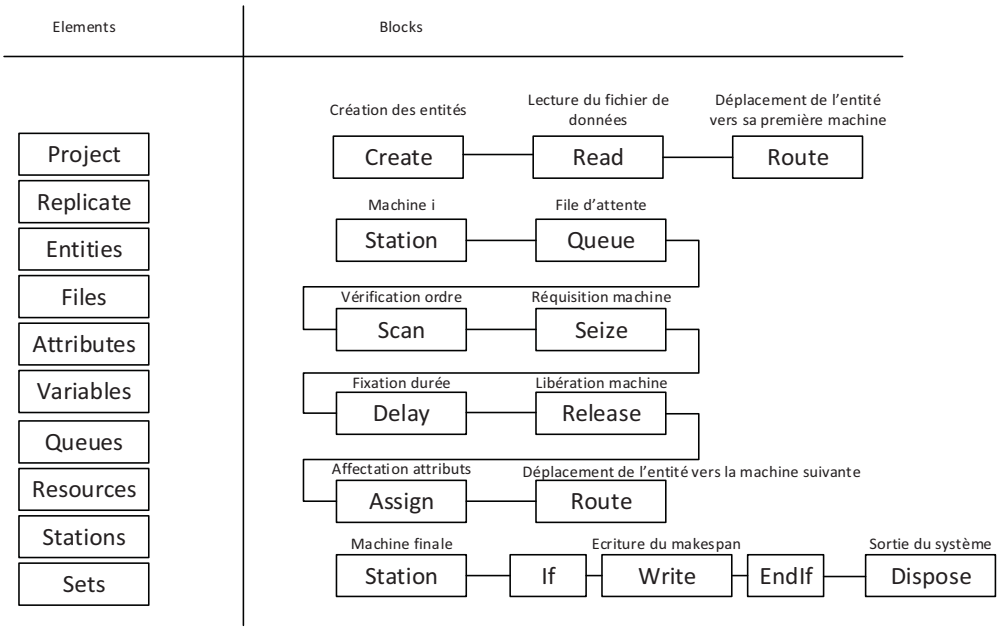

Figure 6 Représentation graphique sous Arena d'un système de type Job-shop

Le modèle lui-même (c'est-à-dire la partie Blocks de la Figure 6) peut être lu ainsi : les premières entités sont créées grâce au bloc *Create*. Tous les jobs sont disponibles à la date 0, donc toutes les entités sont générées au début de la simulation. Chaque fois qu'une entité passe par le bloc *Read*, les données relatives à cette entité sont stockées dans les *variables* et les *attributs* déclarés à cet effet. Ensuite, l'entité arrive au bloc *Route* qui lui permet de se rendre à la station correspondant à la machine devant effectuer sa première opération. L'entité passe par la file d'attente (Queue) attenante à la machine. Si le numéro de passage de l'entité correspond au nombre d'opérations déjà traitées par la machine (Bloc *Scan*), et si la machine est disponible, le job/entité commence à être traité, sinon il doit attendre dans la file d'attente jusqu'à ce qu'il puisse être traité, ce qui implique d'attendre qu'une autre entité prioritaire arrive dans la file d'attente et soit traitée par la machine. Lorsqu'un job quitte une machine, il passe à la machine suivante (Bloc *Route*) conformément à la séquence d'opérations qui lui est propre. Dans le cas où l'entité quitte la dernière machine requise pour être complétement traitée, elle est acheminée vers la station de sortie. Si cette entité est la dernière à sortir du système, sa date de sortie (qui correspond au makespan pour une réplication) est sauvegardée. Lorsqu'une réplication est achevée, l'ensemble de la simulation est exécuté de nouveau jusqu'à atteindre un nombre suffisant d'informations pour obtenir un makespan moyen associé à l'ordonnancement simulé. L'ensemble du processus et de la logique du système sont donnés par le fichier *model*.

Deux blocs permettent de simuler le fonctionnement attendu du système : le bloc *Scan* et le bloc *Delay*. Le premier va autoriser la sortie d'une entité de la file d'attente à condition qu'elle respecte un

critère sur le nombre d'opérations déjà traitées par la machine. Ce critère est donné dans l'équation ci-dessous :

$$jobPriority(jobType, opStage).eq.status\_M(M)$$

En d'autres termes, si la priorité de l'entité (par rapport à son numéro en tant que job, et à l'opération qui doit être effectuée) correspond au numéro autorisé par la machine M, alors l'entité sort de la file d'attente pour être placée sur la machine. Le bloc *Scan* assure que l'ordonnancement obtenu lors de la phase d'optimisation est simulé. Le deuxième point important, à savoir le bloc *Delay*, va définir la durée pendant laquelle l'entité est traitée par la machine ; l'expression calculée lors de l'affectation est la suivante :

$$NORMAL(jobDuration(jobType, opStage), jobDuration(jobType, opStage)/2)$$

Dans cette équation, la partie de gauche correspond à l'espérance de la loi normale, qui correspond à la durée déterministe de l'opération, tandis que la partie droite correspond à la variance.

Comme indiqué précédemment, la Figure 6 est la représentation graphique construite à l'aide du logiciel ARENA. Il convient de mentionner que d'autres modèles pourraient convenir au Job-shop, mais celui-ci a le mérite d'être simple et facilement compréhensible par un utilisateur tiers. Dans cette étude, le but principal de la simulation est d'évaluer la robustesse des séquences sélectionnées pendant la phase d'optimisation. Ainsi, la simulation est utilisée en dehors du logiciel ARENA en utilisant les outils fournis dans le package de l'application et mentionnés dans la Figure 4. Ces outils consistent à construire le fichier final utilisé pour la simulation à partir des fichiers *model* et *experiment*.

Un point important consiste à déterminer le nombre de réplications à effectuer pour que la valeur moyenne du makespan obtenue après les simulations soit pertinente et permette à l'outil d'optimisation de définir de nouvelles solutions de bonne qualité, sans écarter une solution qui aurait pu être bonne mais dont les premières réplications ne conduisent pas à de bonnes valeurs du makespan.

3.3. Définition du nombre de réplications

Lors de la conception d'un modèle de simulation, il est essentiel de déterminer le nombre de réplications (NREP dans Arena) qui doivent être effectuées pour avoir des résultats pertinents. Dans ce travail, une étude a été menée sur un ensemble d'instances représentatives en taille des problèmes utilisés comme jeux de donnés. Ainsi, les instances de Lawrence sur lesquels le GRASP×ELS a été évalué dans le chapitre II sont utilisées. Plus spécifiquement, ce jeu de données comporte différents lots d'instances en fonction de leurs tailles. Le choix fait consiste à prendre la première instance de chaque lot afin de former un échantillon représentatif pour calibrer le nombre de réplications à effectuer, à savoir les instances : La01, La06, La11, La16, La21, La26, La31 et La36. Pour chacune de ces instances, cinq bonnes solutions différentes et dont les temps de traitement des opérations sont déterministes ont été produites à partir de la métaheuristique. Chaque solution est alors simulée sur une base de dix mille réplications. Pour chaque réplication la valeur du makespan obtenu est stockée. La Figure 7 et la Figure 8 montrent la courbe de distribution du makespan pour une exécution classique de chacune des instances.

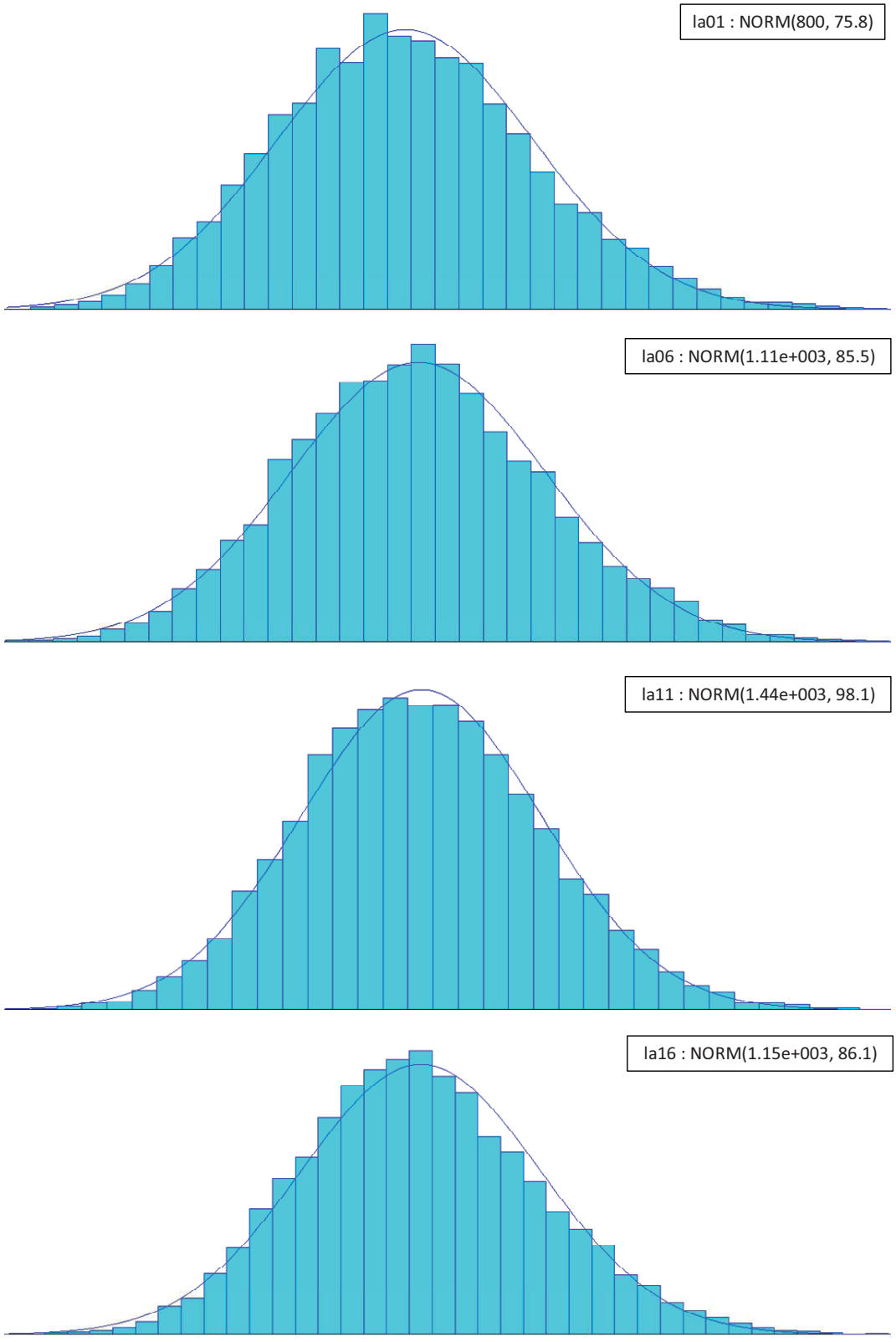

Figure 7 Distributions du makespan pour les instances la01-16

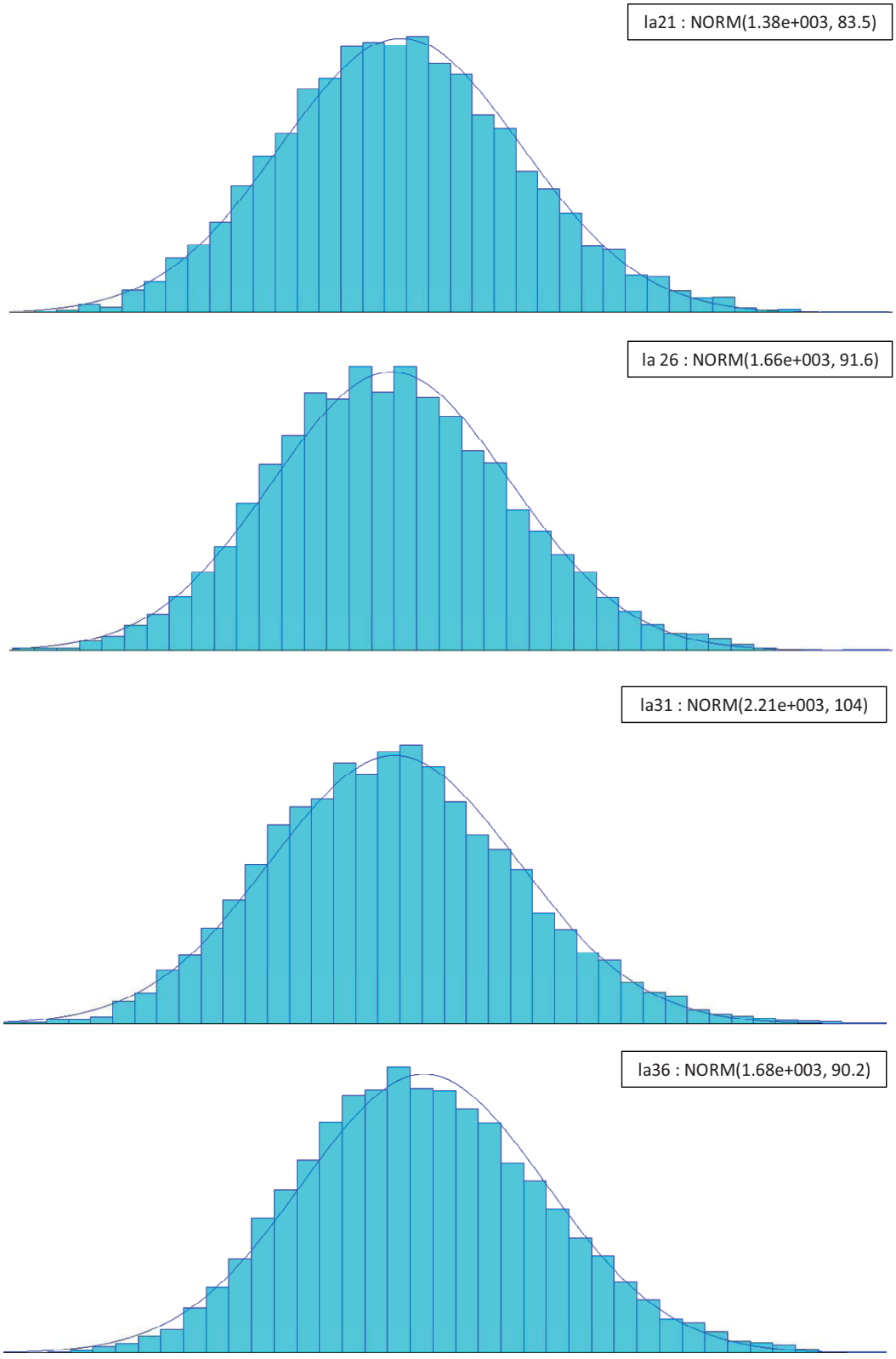

Figure 8 Distributions du makespan pour les instances la21-36

Au vu des distributions observées, il est possible d'approximer les lois que suivent le makespan des instances par des lois normales de paramètre respectif $\mathcal{N}$(800,75.8), $\mathcal{N}$(1110,85.5), $\mathcal{N}$(1440, 98.1), $\mathcal{N}$(1150, 86.1), $\mathcal{N}$(1380, 83.5), $\mathcal{N}$(1660, 91.6), $\mathcal{N}$(2210, 104), $\mathcal{N}$(1680, 90.2). Du fait que les opérations suivent elles-mêmes une loi normale, il n'est pas surprenant d'obtenir une telle distribution pour l'expression du makespan. Ensuite, le meilleur nombre de réplications pour chacune de ces instances est extrait afin de respecter un niveau de confiance (95% dans cette étude) et pour proposer des résultats simulés compris dans un intervalle de confiance. Si l'on considère que les variables (i.e. les valeurs du makespan) sont indépendantes et identiquement distribuées alors on a les équations suivantes :

$$\frac{1}{2}W = z_{\alpha/2} * \frac{\sigma}{\sqrt{n}} \quad (1)$$

$$P\left(Z \leq z_{\alpha/2}\right) = 1 - \frac{\alpha}{2} \quad (2)$$

Dans les équations (1) et (2), $z_{\alpha/2}$ est le $\left(1 - \frac{\alpha}{2}\right)$ quantile de la distribution considérée. W est la largeur de l'intervalle de confiance, σ correspond à l'écart type de l'échantillon, n la taille de l'échantillon et α est le niveau de confiance désiré. Les résultats correspondant à ces tests sont présentés dans le Tableau 1 ci-dessous.

Tableau 1 Valeurs pour le makespan en fonction du nombre de réplications

Problème	Makespan Moyen	W	DEVIATION	NREP
la01	803,43	8,11	1,010	1420
la06	1098,07	11,04	1,005	940
la11	1432,85	14,36	1,002	700
la16	1174,43	11,78	1,003	860
la21	1380,78	13,87	1,005	550
la26	1613,10	16,15	1,001	432
la31	2199,73	22,17	1,008	336
la36	1674,51	16,81	1,004	414

Il apparaît dans le Tableau 1 que, pour respecter un écart approximativement égal à 1%, le nombre de réplications varie d'une instance à l'autre. Il semble également que plus l'instance est petite, plus de réplications sont nécessaires pour obtenir des résultats pertinents. Compte tenu de la variation de NREP sur ces instances, il a été choisi de prendre les valeurs présentées dans le Tableau 2 pour les instances testées dans la suite.

Tableau 2 Nombre de réplications par lot d'instances

Problème	NREP
la01 à la05	1425
la06 à la10	950
la11 à la15	700
la16 à la20	875
la21 à la25	550
la26 à la30	450
la31 à la35	350
la36 à la40	425

Avant de présenter les résultats obtenus sur les 40 instances précédemment testées dans le cas déterministe, une comparaison des résultats obtenus sur la première instance de Lawrence est fournie dans la section suivante.

3.4. Comportement sur une instance test

Comme le processus est itératif, différentes solutions sont explorées. Lorsqu'une phase d'ELS est terminée, la simulation est exécutée sur la séquence obtenue. Une fois la simulation terminée, la séquence présentant le meilleur makespan moyen est sauvegardée. Dans certains cas, des résultats intéressants apparaissent, surtout lorsque différentes solutions semi-actives optimales existent pour un problème. Dans la Figure 9, deux diagrammes de Gantt différents sont présentés, qui sont tous les deux optimaux du point de vue déterministe, mais leurs qualités sont bien différentes si l'on observe le makespan moyen obtenu après simulation. On remarque notamment sur la Figure 9 que la structure des solutions est clairement différente d'un diagramme de Gantt à un autre. Même les chemins critiques (encadrés en noir sur la figure) ne sont pas structurés de la même manière, car, par exemple, la deuxième opération du job J_2 sur la machine M_5 est programmée à la quatrième position dans le premier diagramme Gantt (en haut) alors qu'elle est ordonnancée en huitième position dans le second diagramme. De plus, si ces diagrammes présentent des solutions déterministes qui sont égales, les valeurs moyennes du makespan après simulation ont une différence approximative de 4,4% ce qui suggère que le second ordonnancement est plus robuste que le premier.

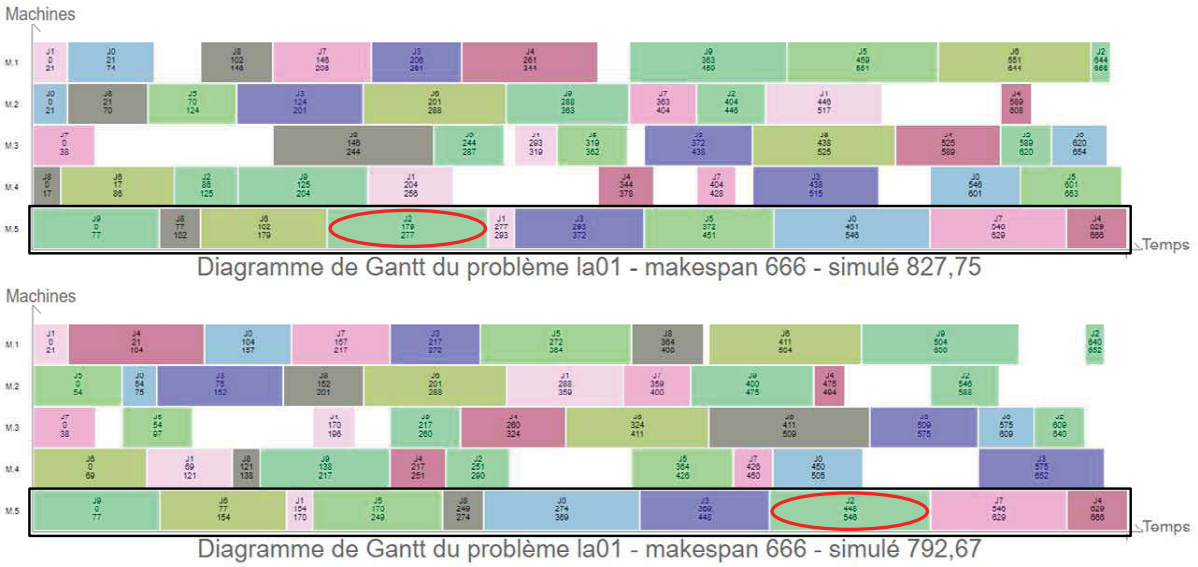

Figure 9 Deux diagrammes de Gantt aux caractéristiques différentes

3.5. Résultats expérimentaux

Au début des expériences menées sur toutes les instances, les paramètres du GRASP×ELS ont été maintenus identiques à ceux utilisés dans le chapitre II (§5.2). Cependant, il est rapidement apparu que la simulation consomme beaucoup de temps, même avec un faible nombre de réplications. Puisque la simulation est traitée juste après la phase ELS, il a été décidé de réduire le nombre de

redémarrages du GRASP à 100 afin de réduire le temps de calcul des simulations. Cependant, une telle démarche implique que les meilleures solutions atteignables par le GRASP×ELS sont modifiées. Ainsi, avec 100 itérations autorisées pour le GRASP (nb_g), le meilleur paramétrage de l'ELS consiste à sélectionner nb_els=180 et nb_n=45. Les écarts entre la meilleure solution retournée par le GRASP×ELS avec ce paramétrage par rapport à la meilleure solution de la littérature augmentent à 0,19%, et à 0,33% concernant les solutions moyennes, ce qui reste acceptable.

Tableau 3 Résultats de l'approche par Optimisation/Simulation

Problème	$\bar{S}$	$\overline{S_{Stoch}}$	$\overline{S_{CPU}}$	S^+	S^+_{Stoch}	$S^+_{Stoch\_CPU}$
la01	666	793.39	103.12	666*	792.67	53.31
la02	655	780.12	113.53	655*	779.95	79.86
la03	597	721.55	114.45	597*	721.17	62.78
la04	590	714.73	110.29	590*	714.73	12.59
la05	593	685.26	95.55	593*	685.26	49.64
la06	926	1082.71	88.39	926*	1082.71	36.81
la07	890	1016.71	99.35	890*	1014.63	22.56
la08	863	1017.46	92.27	863*	1017.46	16.82
la09	951	1108.25	87.39	951*	1097.49	55.47
la10	958	1121.09	93.33	958*	1121.09	3.50
la11	1222	1402.48	92.15	1222*	1400.95	34.58
la12	1039	1216.99	89.85	1039*	1212.77	1.08
la13	1150	1350.42	87.04	1150*	1350.42	49.14
la14	1292	1477.83	90.73	1292*	1470.83	67.47
la15	1207	1408.09	92.60	1207*	1408.09	29.08
la16	946	1141.05	119.58	946	1141.05	59.14
la17	784	964.76	103.86	784*	963.64	102.85
la18	861	1046.10	105.46	861	1046.10	3.17
la19	855	1060.84	111.09	858	1049.77	93.01
la20	915	1116.82	111.06	915	1116.82	89.52
la21	1057.6	1339.93	118.86	1058	1329.75	79.47
la22	936.6	1184.28	117.86	937	1182.75	36.88
la23	1032	1275.37	117.65	1032*	1272.89	42.03
la24	942	1209.87	121.12	939	1207.58	113.06
la25	986.8	1256.88	116.81	984	1250.01	68.02
la26	1219	1538.77	135.85	1218*	1534.26	40.57
la27	1269	1600.15	144.82	1269	1599.71	146.58
la28	1223	1554.69	138.94	1223	1550.97	25.33
la29	1212	1542.19	140.42	1212	1542.19	29.63
la30	1355	1683.52	138.89	1355*	1674.95	120.82
la31	1784	2160.85	138.83	1784*	2160.85	94.89
la32	1850	2274.77	154.31	1850*	2267.63	104.80
la33	1719	2093.54	156.79	1719*	2093.54	9.95
la34	1721	2116.14	154.92	1721*	2116.14	24.34
la35	1888	2252.39	132.64	1888*	2249.25	86.23
la36	1305.2	1620.46	136.73	1331	1619.94	65.65
la37	1444.2	1793.84	139.94	1425	1781.51	106.79
la38	1227	1571.59	144.08	1215	1555.19	20.75
la39	1261	1605.45	146.33	1261	1605.45	30.18
la40	1237.8	1582.00	144.30	1243	1574.22	116.67
Moyenne :			119			57

* = solutions déterministes optimales

Les résultats obtenus dans cette étude sont visibles dans le Tableau 3. Dans ce tableau sont présentés : la solution moyenne déterministe ($\bar{\mathbf{S}}$), la solution moyenne après simulation ($\overline{\mathbf{S_{Stoch}}}$), et le temps total d'exécution en moyenne ($\overline{\mathbf{S_{CPU}}}$). Viennent ensuite les colonnes $\boldsymbol{S}^{+}$ qui est la solution déterministe associée à la meilleure solution stochastique ($\mathbf{S^{+}_{Stoch}}$), et la durée nécessaire pour obtenir cette meilleure solution stochastique ($\mathbf{S^{+}_{Stoch\_CPU}}$).

Les résultats montrent que, la plupart du temps, la meilleure solution stochastique obtenue est aussi la meilleure solution déterministe, ce qui confirme l'approche de (Juan et al., 2014). Cependant, dans certains cas, les ordonnancements optimaux qui ont été retournés pour le cas déterministe (même après avoir réduit le nombre d'itérations du GRASP dans la métaheuristique) ne sont pas considérés comme étant bons du point de vue stochastique. C'est le cas pour les instances la16, la18, la19, et la20. Concernant les instances difficiles, il est délicat d'affirmer que les solutions optimales du cas déterministe auraient été bonnes une fois simulées car les meilleures solutions ne sont pas toujours atteintes lors de l'exécution de la métaheuristique sans prise en compte du module de simulation. Cependant, les solutions choisies comme « bonnes » dans le cas déterministe ne sont plus choisies en tant que bonnes valeurs stochastiques, ce qui est une information intéressante concernant les ordonnancements obtenus dans un tel contexte.

Le seul inconvénient de ce type de procédure proposée concerne les temps de calcul qui sont plus importants que ceux présentés dans le cas déterministe. Ceci est évidemment dû aux phases de simulations qui sont effectuées. Le choix de définir un nombre réduit d'appels à la procédure de simulation par le biais d'un nombre d'itérations du GRASP restreint est donc validé par les résultats. Ces temps de calcul ne sont pas comparables avec ceux obtenus dans le cas déterministe, car ils sont impactés par la durée des simulations et le fait que les différentes bonnes solutions déterministes doivent être trouvées avant d'atteindre la meilleure solution stochastique.

Cependant, les temps de calcul sont inférieurs à deux minutes en moyenne, ce qui est acceptable compte tenu de la difficulté des problèmes à l'étude. Comme il n'est pas possible, à moins de garder une trace de toutes les solutions visitées, de savoir si les solutions optimales ont été trouvées et rejetées par la métaheuristique, un comparatif entre les solutions présentées dans le Tableau 3 et les valeurs des makespan obtenus après simulations des meilleures solutions obtenues dans le chapitre II (§5.2) est proposé. Ce comparatif est proposé dans le Tableau 4.

Il apparaît dans le Tableau 4 que certaines solutions optimales, ou de meilleure qualité d'un point de vue déterministe, sont de moins bonne qualité une fois simulées, ce qui suggère que les solutions optimales ne sont pas toujours les plus à même d'apporter de la robustesse au sein d'un système de production. Cependant, dans certaines situations la meilleure solution obtenue lors de l'exécution de la métaheuristique appliquée dans le chapitre II (§5.2) est également meilleure une fois simulée (la21, la37). Ce résultat est à mettre en perspective avec la différence de qualité entre les solutions déterministes obtenues entre les deux métaheuristiques. En effet, si l'instance la21 montre une meilleure valeur du makespan simulée (de près de 11 points), la solution déterministe est également de 12 points inférieure. Or il est possible de constater sur d'autres instances qu'une valeur du makespan supérieure dans le cas déterministe peut conduire à de meilleures solutions simulées, il n'est donc pas à exclure qu'une solution dont le makespan est entre les valeurs présentées dans le tableau puisse être de bien meilleure qualité. Ce tableau montre également la présence de plusieurs ordonnancements optimaux pour une même instance et présentant des caractéristiques différentes dans la structure des solutions (la01, la02, la09, la33…) ce qui était un des résultats attendus par cette étude.

Tableau 4 Comparatif de la qualité des solutions entre l'optimisation seule et le couplage

	GRASP×ELS (optimisation seule)		GRASP×ELS (couplage)	
Problème	S^+	S^+_{Stoch}	S^+	S^+_{Stoch}
la01	666*	827,75	666*	792,67
la02	655*	789,31	655*	779,95
la03	597*	732,43	597*	721,17
la04	590*	722,65	590*	714,73
la05	593*	702,74	593*	685,26
la06	926*	1102,31	926*	1082,71
la07	890*	1021,62	890*	1014,63
la08	863*	1056,95	863*	1017,46
la09	951*	1131,04	951*	1097,49
la10	958*	1111,43	958*	1121,09
la11	1222*	1460,45	1222*	1400,95
la12	1039*	1229,83	1039*	1212,77
la13	1150*	1371,44	1150*	1350,42
la14	1292*	1516,58	1292*	1470,83
la15	1207*	1422,21	1207*	1408,09
la16	945*	1180,49	946	1141,05
la17	784*	968,72	784*	963,64
la18	848*	1062,87	861	1046,10
la19	842*	1071,84	858	1049,77
la20	902*	1134,17	915	1116,82
la21	1046*	1318,39	1058	1329,75
la22	927*	1188,56	937	1182,75
la23	1032*	1308,03	1032*	1272,89
la24	935*	1213,96	939	1207,58
la25	977*	1254,58	984	1250,01
la26	1218*	1547,29	1218*	1534,26
la27	1248	1595,51	1269	1599,71
la28	1216*	1568,57	1223	1550,97
la29	1185	1536,96	1212	1542,19
la30	1355*	1716,77	1355*	1674,95
la31	1784*	2179,57	1784*	2160,85
la32	1850*	2297,46	1850*	2267,63
la33	1719*	2130,08	1719*	2093,54
la34	1721*	2123,40	1721*	2116,14
la35	1888*	2320,28	1888*	2249,25
la36	1278	1641,80	1331	1619,94
la37	1397*	1778,16	1425	1781,51
la38	1196*	1551,94	1215	1555,19
la39	1240	1611,33	1261	1605,45
la40	1226	1581,63	1243	1574,22

* = solutions déterministes optimales

3.6. Conclusion

Cette partie a proposé un couplage optimisation/simulation où l'objectif est de trouver des ordonnancements robustes pour un problème de Job-shop. La phase d'optimisation se fait à l'aide d'une métaheuristique de type GRASP×ELS, tandis que la simulation est traitée par le langage SIMAN. Les temps de traitement des opérations suivent une distribution normale dans le module de simulation avec une variance qui équivaut à la moitié de la durée déterministe. La durée déterministe

de chaque opération est utilisée en tant qu'espérance dans la loi de distribution. Cette variance est relativement importante, ce qui permet de modéliser des opérations avec une forte incertitude quant à leurs durées. À la fin de chaque application de l'ELS, la robustesse de la meilleure solution trouvée est testée et la plus robuste est gardée comme étant la meilleure solution globale. Les résultats montrent la pertinence de cette étude en plusieurs points. Tout d'abord, comme prévu, il est apparu que la meilleure solution déterministe n'est pas toujours la plus adaptée à un cadre stochastique. Ensuite, en fonction de la structure des instances, la phase d'optimisation peut découvrir différentes solutions optimales du point de vue déterministe qui ne sont pas de même qualité une fois que la robustesse a été évaluée. D'autres études pourraient être conduites sur l'impact des données sur la pertinence des solutions retournées. Il est également important de noter que l'ensemble des résultats est obtenu à partir d'un unique paramétrage, il pourrait être intéressant de définir des paramétrages différents en fonction des tailles d'instances afin d'observer les différences dans la qualité des solutions ce qui pourrait être un moyen d'améliorer les résultats sur des problèmes de grande taille.

4. Proposition d'une approche pour le Job-shop Flexible réactif

Comme il a été montré dans la section présentant les différents travaux sur les couplages entre simulation et optimisation, la définition de systèmes réactifs présente également un enjeu important pour les industriels (Sharma and Jain, 2014). D'après (Lin et al., 2012), la littérature sur le Job-shop Flexible réactif est cependant peu abondante, bien que de nombreuses études aient été menées sur ce type de système au cours des dix dernières années. Comme l'ont noté (Ouelhadj and Petrovic, 2009) peu d'articles utilisent des métaheuristiques dans le cadre d'un problème d'ordonnancement dynamique. De ce fait, une recherche rapide sur la base de données Elsevier montre que moins de vingt articles présentent les termes « réactif » et « Job-shop » dans leur titre. Ce nombre monte à cinquante si l'on remplace « réactif » par « dynamique », et à moins de 250 si l'on considère les titres, résumés et mots-clés, la majorité des 250 articles étant des problèmes de Job-shop et non de Job-shop Flexible. En outre, ces travaux reposent beaucoup sur l'implémentation de règles de gestion capables de prendre des décisions rapidement lors de l'apparition d'aléas sur le système de production. Aussi, la plupart des approches observées dans la littérature ne font pas intervenir un outil de simulation dédié. Il est donc proposé d'étudier l'apport et la pertinence d'un module d'optimisation au sein d'un modèle de simulation. Comparativement à l'approche OSI appliquée dans la section précédente, le type d'approche employée dans cette section se situe dans la classe « simulation avec des itérations basées sur l'optimisation » (Simulation with optimization-based iterations – SOI).

4.1. Présentation du problème

Dans cette section, un problème de Job-shop Flexible réactif est étudié. Ce problème est une extension du problème de Job-shop Flexible présenté dans le chapitre précédent (chapitre IV, §2), et qui consiste en un problème d'ordonnancement et un problème d'affectation. Chaque job devant être traité est composé d'un ensemble d'opérations qui doivent être réalisées dans un ordre spécifique. Chacune de ces opérations peut être affectée à une machine parmi un ensemble de machines compatibles. Dans le Job-shop Flexible classique, l'ensemble des informations est connu avant tout processus d'optimisation. Cependant, dans le problème étudié dans cette section, les jobs peuvent arriver à n'importe quel moment. Ainsi, un problème de Job-shop Flexible réactif, ou dynamique, est considéré. Si dans la section précédente des durées aléatoires étaient considérées lors du traitement de chaque opération, sans perte de généralité les durées des opérations sont considérées déterministes

dans cette section. L'objectif du problème consiste à trouver un ordonnancement minimisant la durée totale de traitement de tous les jobs au fur et à mesure que l'information arrive dans le système afin d'optimiser la productivité de l'atelier.

De la même manière que (Rahmani and Ramezanian, 2016) qui proposent une approche réactive pour un problème de Flow-shop, l'objectif de cette section est de proposer un module d'optimisation efficace permettant de réorganiser un ordonnancement préétabli lorsque de nouveaux jobs sont à intégrer au sein d'une production déjà en cours. La problématique consiste donc à saisir l'état du système à un instant donné (celui où un nouvel entrant est à considérer dans le système de production) et à redéfinir un ordonnancement et une affectation des opérations qui ne sont pas en cours de traitement au moment de la prise de décision. Si des événements aléatoires supplémentaires pourraient être considérés, le principe mis en œuvre dans cette section resterait inchangé, il est donc fait le choix de ne considérer que de nouveaux jobs à ordonnancer.

Dans la suite, l'approche mise en œuvre est présentée, en détaillant le principe du couplage et les éléments permettant son fonctionnement. Un exemple permet de présenter le fonctionnement du module d'optimisation lorsqu'un nouveau job entre dans le système. Les résultats sur des jeux de données sont également exposés.

4.2. Approche Simulation-Optimisation

L'intégration d'un module d'optimisation au sein d'un modèle de simulation vise à supplanter les règles de gestion qui peuvent être appliquées afin de définir les priorités dans le traitement des opérations dans le système de production. Dans ce type d'approche, le module d'optimisation est appelé en support lors de l'apparition d'événements qui nécessitent une prise de décision, plutôt que de laisser les règles de gestion définir l'ordre de traitement des opérations de manière dynamique. L'avantage d'une telle approche est de pouvoir proposer un modèle de simulation évolutif tout en optimisant la productivité du système de production à l'aide d'un module d'optimisation consacré à l'ordonnancement et à l'affectation des opérations sur les machines les plus prometteuses vis-à-vis du critère de performance évalué.

Dans cette section, le cadre de résolution vise donc à déterminer une solution partielle de bonne qualité lorsqu'un événement aléatoire intervient (l'arrivée de nouveaux jobs dans le cas étudié). L'approche repose sur l'intégration au sein du modèle de simulation d'un module d'optimisation basé sur une métaheuristique. En effectuant plusieurs réplications, il est alors possible de mesurer la pertinence de la métaheuristique en tant qu'outil d'aide à la décision dans un environnement où des aléas interviennent. Ceci pose une question importante sur la rapidité à laquelle la métaheuristique doit retourner une solution applicable sur le système de production. En effet, sur un système où les durées des opérations sont très courtes, la prise de décision doit être quasiment instantanée. Cependant, sur des systèmes où les opérations peuvent durer plusieurs minutes, voire heures, il est possible d'allouer plus de temps aux calculs. Dans tous les cas, évaluer la performance d'une métaheuristique dans un environnement dynamique reste un objectif intéressant. Une représentation graphique de l'approche est donnée sur la Figure 10.

La Figure 10 montre que la première étape consiste à gérer les arrivées des entités au sein du système. Lorsqu'une nouvelle entité est générée, le modèle de simulation fait appel au module d'optimisation afin de définir un ordre de passage des opérations sur les machines pour l'ensemble

des entités à traiter. Le module d'optimisation, à l'instant où il est appelé, va donc saisir l'état du système. Ceci consiste à obtenir des informations sur les jobs en production et par conséquent sur les opérations restant à produire pour chaque job déjà présent dans le système de production, tout en ajoutant le nouveau job arrivé en entrée du système. Lorsque le module d'optimisation a défini un nouvel ordonnancement et une nouvelle affectation pour toutes les opérations, le module de simulation reprend la main et applique les nouvelles priorités sur les machines jusqu'à ce qu'une nouvelle entité (job) soit générée. Ce type de procédé visant à passer de l'un des modules à l'autre est également appelé « simulation-optimisation alternée ». Lorsque l'ensemble des jobs est traité pour une réplication donnée, le module de simulation stocke la valeur mesurée du makespan, puis une nouvelle réplication est effectuée jusqu'à ce que l'ensemble des réplications soit achevé. Dans ce cas le module de simulation retourne la valeur moyenne du makespan.

Dans la suite le module de simulation adapté au problème est présenté. Puis, le module d'optimisation est introduit. Ce dernier repose sur la métaheuristique GRASP-mELS définie lors du chapitre précédent (chapitre IV, §5).

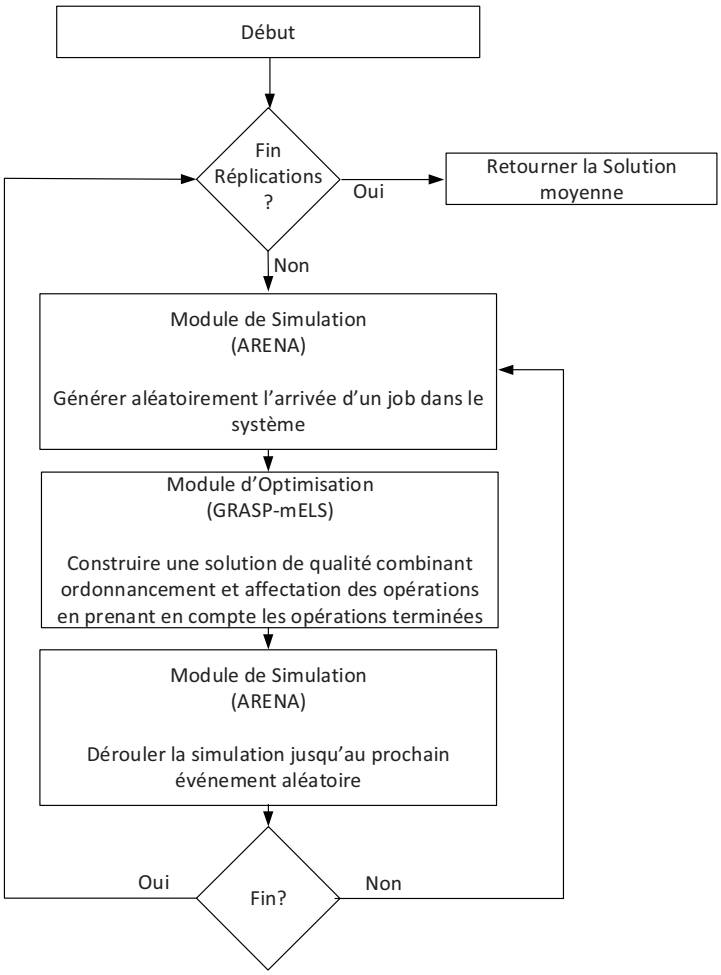

Figure 10 Vue d'ensemble du processus de résolution

4.2.1. Module de simulation

Si des éléments (blocks) sont récurrents entre le modèle de simulation utilisé pour le Job-shop stochastique et celui développé dans cette section, de nombreuses parties diffèrent.

En effet, si un problème d'ordonnancement est commun entre le Job-shop stochastique et le Job-shop Flexible, ce dernier inclut également un problème d'affectation. Si le routage des entités peut être effectué simplement à partir du modèle présenté en Figure 6, la principale difficulté réside dans l'aspect dynamique associé au problème traité dans cette section. En effet, comme énoncé plus haut (§4.1), il est prévu que des entités arrivent dans le système de manière aléatoire, sans que leur ordre ne soit connu à l'avance. L'objectif est alors de redéfinir les affectations prévisionnelles et l'ordonnancement de l'ensemble des entités présentes dans le système au moment de l'appel à la procédure d'optimisation. Un certain nombre de difficultés techniques sont alors à prendre en compte par rapport au modèle de la Figure 6. Le problème principal concerne le déclenchement d'événements pour empêcher tout blocage sur le système. En effet, si l'approche consiste à garder dans une file d'attente une entité dont le numéro de passage ne correspond pas à celui qui est accepté par la machine, l'entité doit demeurer dans la file d'attente jusqu'à ce qu'elle soit autorisée à en sortir. Or, si une entité a un numéro de passage inadapté à un instant donné, rien ne permet d'assurer que ce numéro de passage n'évolue pas dans le temps du fait des appels successifs au module d'optimisation. Ainsi, la gestion des entités dans les files d'attente représente un point essentiel du modèle de simulation proposé. Celui-ci est présenté en Figure 11. Pour des raisons d'espace la partie *Elements* est omise, celle-ci pouvant être gérée depuis les menus d'Arena (ce qui n'était pas le cas du modèle présenté dans la section précédente). Lorsque des intersections sont présentes des pointillés sont utilisés pour distinguer les bonnes directions. Des flèches sont également ajoutées afin de faciliter la lecture du modèle.

Comparativement au modèle présenté en Figure 6 (section précédente), deux groupes de blocs sont ajoutés : le Groupe1 et le Groupe3. Le modèle de simulation débute par un bloc *Create* qui va générer une entité Ent. Ensuite le bloc *Assign* (Affectation_1) génère un type de job choisi aléatoirement parmi les types de Jobs disponibles et l'associe à l'entité courante Ent. L'entité déclenche alors un événement à l'aide du bloc *Event* qui permet un appel au module d'optimisation. La liaison entre le module de simulation et le module d'optimisation se fait par une DLL. La structure de base de cette DLL est fournie avec le logiciel Arena, et il revient à l'utilisateur de la faire évoluer et d'y intégrer les composants souhaités. Lorsque l'entité sort du bloc *Event*, une solution du problème d'ordonnancement a été trouvée. L'entité passe alors dans un bloc *Duplicate* qui va générer une entité temporaire Ent_2, tandis que l'entité déclencheuse Ent est envoyée vers la première machine qui lui est affectée (bloc *Station*). L'entité temporaire Ent_2entre dans le Groupe1, où elle a pour objectif d'examiner les différentes files d'attente pour en sortir les entités pouvant être traitées par des machines. La procédure est la suivante : le bloc *Branch* envoie l'entité vers le bloc *FindJ* si la file d'attente associée à la station testée est non vide, sinon l'entité examine une autre station. Le bloc *FindJ* effectue une recherche dans la file d'attente et retourne l'indice d'une entité si elle respecte le critère de la recherche : que le numéro de passage sur la machine corresponde au numéro attendu par celle-ci. Si aucune entité dans la file d'attente ne respecte la condition alors l'entité Ent_2 passe à la station suivante (mise à jour à l'aide du bloc *Assign* Affectation_2), sinon elle est envoyée vers le bloc *Remove*. Ce bloc retire l'entité concernée de la file d'attente et l'envoie vers la machine devant la recevoir. L'entité Ent_2 continue alors d'explorer chaque station jusqu'à être détruite (bloc *Dispose*).

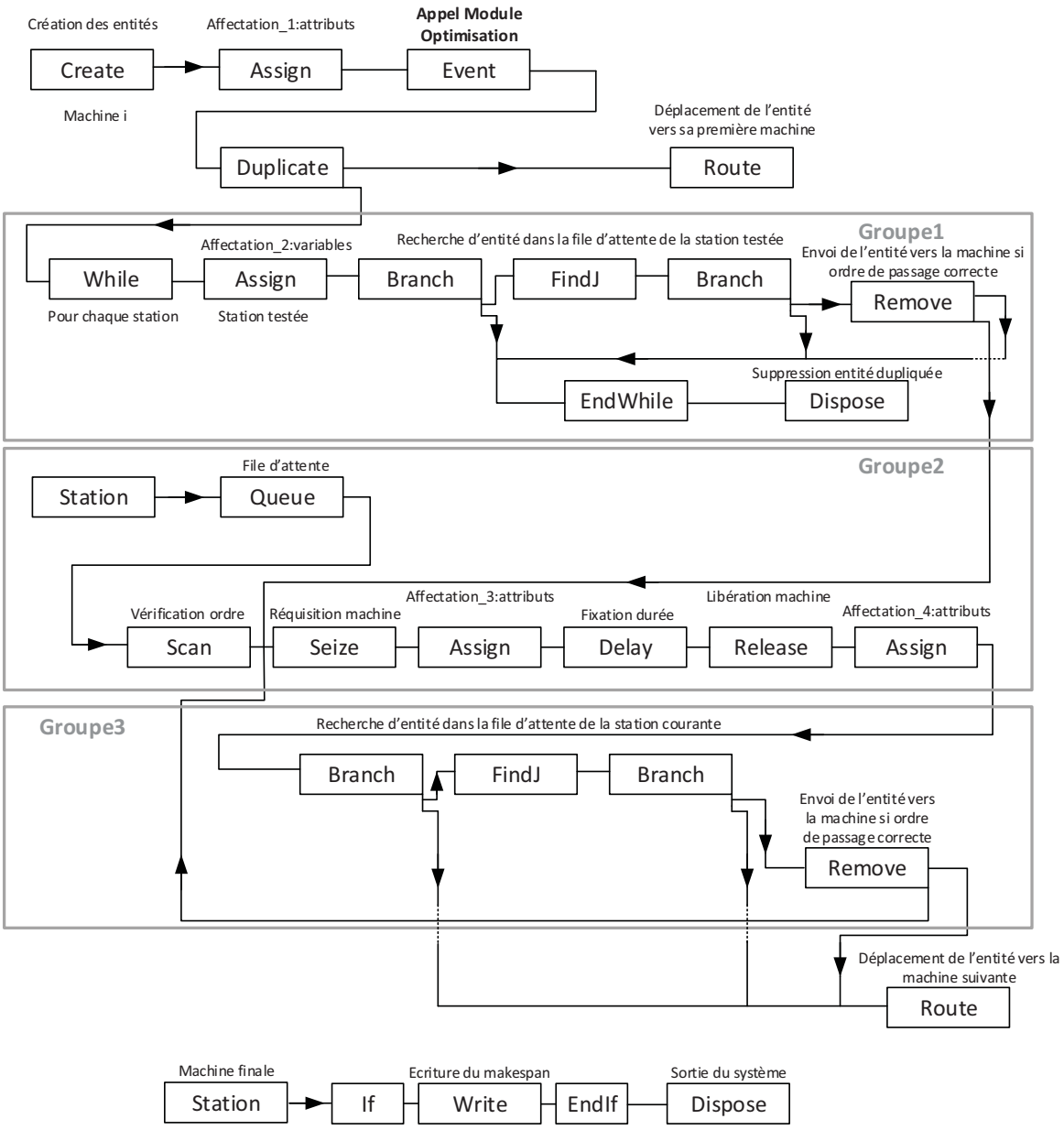

Figure 11 Représentation graphique sous Arena d'un système de type Job-shop Flexible

Concernant l'entité ***Ent*** envoyée vers la station, elle commence par entrer dans la file d'attente associée à la station. Si elle est seule ou que son numéro de passage sur la machine est inférieur à celui des autres entités dans la file d'attente alors elle est envoyée vers le bloc Scan. Si son numéro de passage correspond au numéro attendu par la machine, alors l'entité continue son chemin vers la machine devant la traiter. Un bloc *Assign* (Affectation_3) est ajouté entre le *Seize* et le *Delay* afin de sauvegarder des informations sur le système pour l'application du module d'optimisation. Lorsque ***Ent*** a terminé son traitement sur la machine, elle atteint un bloc *Assign* (Affectation_4) qui permet de mettre à jour ses attributs en vue des prochains traitements à effectuer, ce bloc *Assign* incrémente également le numéro de passage attendu sur la machine.

Lorsque l'entité quitte le Groupe2, elle rejoint le Groupe3 afin de déclencher une procédure similaire au Groupe1 ; l'absence de bloc *While* et la présence d'un bloc *Route* justifie l'ajout de ce dernier groupe. Une fois la vérification sur la file d'attente actuelle effectuée, l'entité est envoyée vers la prochaine station. Si elle a subi tous les traitements, elle est dirigée vers la sortie du système (*Dispose*), où la valeur du makespan est sauvegardée s'il s'agit de la dernière entité à partir du système.

Le problème étudié dans cette section consiste à vérifier la pertinence du module d'optimisation dans la définition d'ordonnancements de bonne qualité lorsque de nouvelles entités intègrent le système. Les temps de traitement des opérations sur les machines sont déterministes. Le module d'optimisation est appelé lorsque l'entité passe dans le bloc *Event*. Dans la suite le principe de fonctionnement de ce module est présenté.

4.2.2. Module d'optimisation

Dans le chapitre précédent, un GRASP-mELS a été utilisé pour résoudre le problème de Job-shop Flexible. Les performances de cette métaheuristique ayant été validées, c'est donc naturellement qu'elle est choisie en tant qu'outil d'aide à la décision pour le problème du Job-shop Flexible réactif.

Lorsqu'un appel au module d'optimisation est effectué, une première étape consiste à extraire du modèle de simulation les informations permettant la résolution d'un problème de Job-shop Flexible. En effet, lorsqu'une nouvelle entité arrive dans le système, d'autres entités sont déjà présentes depuis plus ou moins longtemps et ont donc déjà subi des traitements par les machines. Il paraît évident qu'une opération déjà effectuée n'a pas lieu d'être intégrée dans le module d'optimisation. Ainsi lors de l'appel au module d'optimisation, les entités pour lesquelles des opérations ont déjà été effectuées sont transformées en jobs comportant uniquement le nombre d'opérations restant à traiter. Afin d'extraire l'ensemble des informations, le module contient une procédure qui examine les différentes variables permettant de savoir quelle entité est actuellement en traitement par une machine. Aussi, chaque file d'attente est analysée, et il est considéré que toute entité présente dans une file d'attente, et qui attend donc d'être traitée par une machine, ne peut être réaffectée à une autre machine. Seule la priorité de passage qui est associée à une entité présente dans une file d'attente peut être modifiée ; les opérations suivantes ne sont pas touchées par ce choix et peuvent donc être réaffectées à n'importe quelle machine parmi l'ensemble des machines pouvant traiter l'opération. Avec les informations de l'état courant du système un nouveau graphe disjonctif est construit. Ce graphe comporte toutes les opérations qui doivent toujours être effectuées (les opérations qui appartiennent aux jobs déjà présents dans le système), plus le nouveau job à intégrer dans le système. Le processus d'optimisation est ensuite exécuté. Pour le problème étudié, la métaheuristique reste inchangée dans sa structure par rapport au Job-shop Flexible classique. Le GRASP-mELS n'est donc pas présenté dans cette section, le lecteur pouvant se référer au chapitre IV (§5) présentant cette métaheuristique.

Lorsque le processus d'optimisation est terminé, une procédure réinjecte dans le module de simulation les informations permettant de continuer le déroulement de la simulation. Ainsi, les priorités de passage des entités sur les machines sont mises à jour afin de ne simuler que l'ordonnancement choisi par la métaheuristique. Par conséquent le routage de chaque entité au sein du système de production est également transmis au module de simulation. Le mécanisme appliqué au sein du module d'optimisation est présenté dans la Figure 12.

Dans la suite un exemple succinct du fonctionnement du couplage mis en œuvre est présenté.

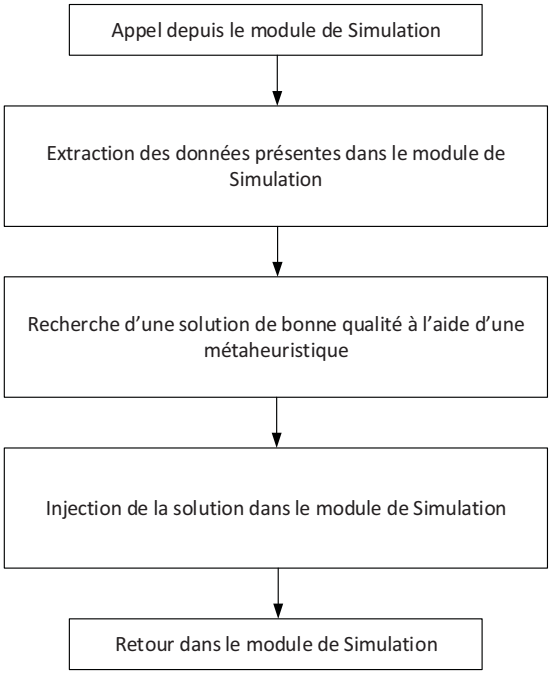

Figure 12 Fonctionnement du module d'optimisation

4.3. Exemple d'application

Supposons qu'un problème soit composé de deux types de jobs tel qu'énoncé dans le Tableau 5.

Tableau 5 Exemple de deux types de jobs possibles dans le module de simulation

	O_1	**O_2**	**O_3**
J_1	$(M_1–D_4^1)(M_2–D_8^2)$	$(M_3–D_5^3)$	$(M_1–D_6^1)(M_2–D_3^2)(M_3–D_4^3)$
J_2	$(M_1–D_7^1)(M_2–D_3^3)$	$(M_1–D_4^1)(M_2–D_{12}^2)(M_3–D_9^3)$	$(M_2–D_4^2)(M_3–D_8^3)$

Dans le Tableau 5, deux jobs avec chacun trois opérations sont présentés. Les opérations peuvent être traitées par trois machines différentes, avec des durées potentiellement différentes (e.g. l'opération O_1 du job J_1 peut passer 4 unités de temps sur la machine M_1, ou 8 unités de temps sur la machine M_2). A la date 0, une première entité entre dans le système. Cette entité est de type 1 (job J_1) et son identifiant est Ent_1. L'entité entre alors dans le bloc *Event* et un appel au module d'optimisation est effectué. Ce module construit le graphe suivant :

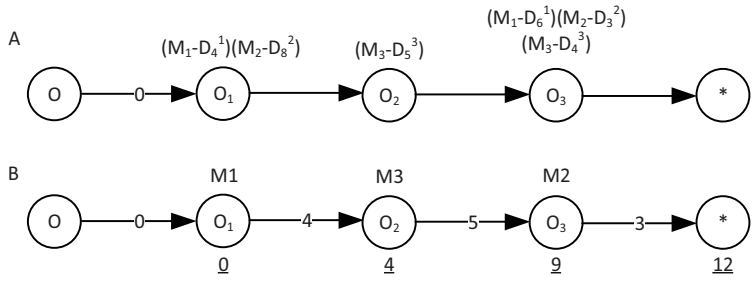

Figure 13 Graphe modélisant le problème à résoudre à la date 0 (A) et solution retournée (B)

Dans la Figure 13(A), un graphe composé d'un seul Job est construit. En sortie du module d'optimisation, l'entité se voit affecter une séquence de passage sur les machines. Cette séquence consiste à passer sur les machines M_1, puis M_3 et enfin M_2. Lorsque l'entité Ent_1 sort du bloc *Event*, le makespan prévisionnel vaut 12. L'entité étant seule dans le système elle est prioritaire sur toutes ces machines. Aussi, elle est directement envoyée à la machine M_1 pour sa première opération, qu'elle commence à la date 0.

A la date 2, inférieure à la date de fin de la première opération de l'entité Ent_1, une nouvelle entité entre dans le système. Cette entité correspond à un job J_2 et son identifiant est Ent_2. Lorsqu'elle arrive au bloc Event, un appel au module d'optimisation est effectué. L'état courant du système est transmis au module d'optimisation. Ce module construit le graphe donné dans la Figure 14 sans inclure la première opération de Ent_1 compte tenu du fait qu'elle est en cours de traitement par la machine M_1 :

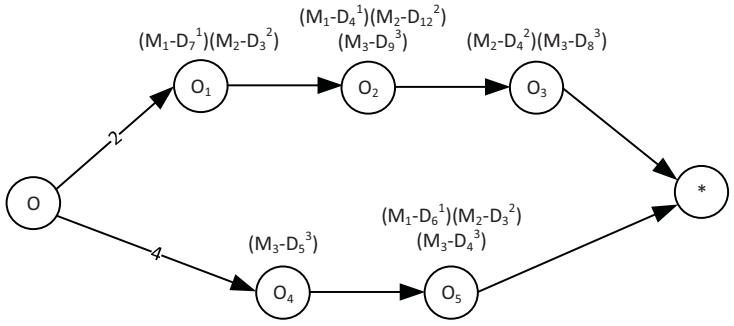

Figure 14 Graphe modélisant le problème à résoudre à la date 2 (A)

Dans la Figure 14 le graphe du nouveau problème à résoudre est présenté. Comme l'entité Ent_1 a déjà commencé son traitement sur la machine M_1, le module d'optimisation construit un job avec deux opérations correspondant aux deux opérations restantes pour cette entité (O_4 et O_5). Le graphe du problème comporte donc cinq opérations (plus les opérations modélisant la source et le puits). Comme l'entité Ent_1 finit son traitement à la date 4 sur la machine M_1, l'arc allant de la source O vers l'opération O_4 est pondéré par la date de disponibilité de l'entité. De la même manière, l'entité Ent_2 arrive dans le système à la date 2, donc l'arc allant de la source à l'opération O_1 est pondéré par cette valeur. Après optimisation, la solution retournée au module de simulation consiste à passer, pour l'entité Ent_1, sur les machines M_3 (avec priorité 1) puis de nouveau sur M_3 (avec priorité 2 car à ce moment-là une entité aura déjà été traitée par la machine). L'entité Ent_2 débute son traitement sur la

machine M_2 en étant prioritaire. Elle est ensuite envoyée pour ses deux opérations restantes sur les machines M_1 et M_2. Avec l'ordonnancement et l'affectation considérés, le makespan prévisionnel vaut désormais 13, tel que montré par la Figure 15.

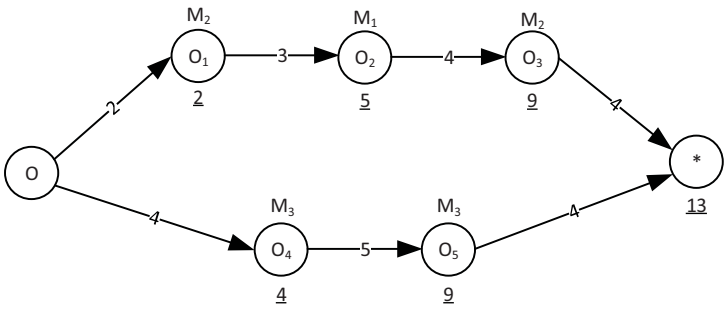

Figure 15 Graphe conjonctif évalué

Il apparaît donc que l'ordonnancement initial a été modifié : la dernière opération de l'entité Ent_1 n'est plus affectée à la machine M_2, mais à la machine M_3. Si l'affectation n'avait pas été modifiée, le makespan aurait valu au mieux 15.

Il reste à évaluer la pertinence de la métaheuristique dans le cadre d'un problème de Job-shop Flexible avec plus de types de jobs. Une expérimentation numérique est donc effectuée sur un ensemble d'instances dans la suite.

4.4. Expérimentations numériques

Dans cette partie les performances de l'approche présentée, à savoir l'inclusion d'un module d'optimisation au sein d'un modèle de simulation développé sous Arena, sont analysées. Ce module d'optimisation est comparé à un modèle de simulation dans lequel deux règles de gestion sont utilisées en lieu et place du module d'optimisation. Les règles testées sont les suivantes :

- Règle 1 : SPT (Shortest Processing Time – durée de traitement la plus courte) pour le choix de la machine et FIFO pour la gestion des files d'attentes. Cette règle est notée **SPT+FIFO** ;

- Règle 2 : Machine pouvant finir le traitement de l'opération le plus tôt (Load Balance) puis règle FIFO pour les files d'attentes. Cette règle est notée **LB+FIFO** ;

Les expérimentations sont effectuées à partir des données présentes dans cinq instances de la littérature : les quatre instances de (Kacem et al., 2002) et l'instance Mk10 de (Brandimarte, 1993). Ces instances constituent la base des types de jobs auxquels une entité peut être rattachée et qui peuvent être créés en entrée du système. Les instances de Kacem (Kacem01-04) comportent de 4 à 15 jobs « types », et de 5 à 10 machines. Chaque job est composé de 2 à 4 opérations, et chaque opération peut être traitée par n'importe quelle machine avec des durées différentes. L'instance Mk10 comporte 20 jobs, 15 machines et chaque job peut subir de 10 à 14 opérations. Chaque opération peut être traitée par un ensemble de 1 à 5 machines. Pour chaque instance testée, un ensemble de 10 réplications est effectué. Ce nombre est choisi de sorte que les temps de calcul pour l'approche combinant

l'optimisation et la simulation ne soient pas prohibitifs compte tenu des appels répétés à la métaheuristique. Pour chaque réplication 50 entités sont générées en entrée du système. Différentes durées d'inter-arrivées (DIA) sont appliquées : une durée égale à zéro (**DIA_0**), ainsi que des lois de probabilités parmi lesquelles des lois exponentielles de paramètres 1 ou 5 (respectivement **$DIA_{EXPO(1)}$** et **$DIA_{EXPO(5)}$**). Les valeurs moyennes des makespan obtenus sont présentées dans le Tableau 6 ci-dessous.

Tableau 6 Makespan moyen pour 10 réplications

	Métaheuristique			LB+FIFO			SPT+FIFO		
	$DIA_{EXPO(5)}$	$DIA_{EXPO(1)}$	DIA_0	$DIA_{EXPO(5)}$	$DIA_{EXPO(1)}$	DIA_0	$DIA_{EXPO(5)}$	$DIA_{EXPO(1)}$	DIA_0
Kacem01	244,43	115,49	113,80	244,70	124,54	126,30	293,54	278,00	281,60
Kacem02	240,88	61,86	51,60	240,92	63,92	57,10	241,27	82,43	73,60
Kacem03	238,38	52,94	27,70	238,39	53,12	32,20	238,63	66,19	59,20
Kacem04	240,32	55,49	36,40	240,32	56,56	41,00	240,43	74,78	70,60
Mk10	572,90	550,00	548,40	596,01	575,35	570,40	890,30	871,85	813,70

Il apparaît dans le Tableau 6 que lorsque les durées inter-arrivées augmentent, les comportements des trois méthodes évaluées sont similaires. Ainsi, concernant la colonne $DIA_{EXPO(5)}$ les valeurs moyennes des makespan pour les instances Kacem02-04 sont pratiquement identiques. La règle SPT+FIFO est néanmoins moins performante pour l'instance Kacem01. Lorsque l'on passe à des durées inter-arrivées plus faibles ($DIA_{EXPO(1)}$ et DIA_0), qui représentent donc un rythme soutenu dans le fonctionnement du système de production, la règle SPT+FIFO semble inadaptée. Si l'on se réfère aux colonnes $DIA_{EXPO(1)}$ il est possible de voir que la règle LB+FIFO présente un excellent comportement, bien qu'un écart existe entre la métaheuristique et la règle de gestion. Cet écart est encore plus visible pour les colonnes DIA_0, avec des distances importantes entre les valeurs obtenues. Un résumé des déviations par rapport aux solutions obtenues par la métaheuristique est donné dans le Tableau 7. Dans ce tableau, il est possible de voir que la métaheuristique est particulièrement adaptée lorsque les durées inter-arrivées sont faibles, avec une déviation de plus de 4% en moyenne entre la règle LB+FIFO et la métaheuristique pour la colonne $DIA_{EXPO(1)}$. Cette déviation est proche de 12% pour la colonne DIA_0 entre la règle LB+FIFO et la métaheuristique. Il n'y a que pour l'instance Mk10 que les déviations mesurées évoluent peu entre les différentes durées inter-arrivées. Le tableau met également en évidence que la règle SPT+FIFO est inutilisable pour des durées inter-arrivées faibles. Il est également important de noter que les makespan obtenus dans la colonne DIA_0 ne représentent pas la résolution d'un problème dont tous les jobs seraient disponibles à la date 0. Dans ce cas précis, la première opération du premier job commence à être traitée par la machine qui lui est affectée sans connaître l'existence des autres entités censées être disponibles à la date 0 également. Dans ce cas précis, la métaheuristique pourrait être appliquée indépendamment du module de simulation en considérant un ensemble d'instances constituées de 50 jobs générés aléatoirement. Ce test n'est pas effectué ici car l'objectif est d'observer la capacité à redéfinir un ordonnancement lorsque des décisions ont déjà été prises et non d'évaluer la capacité de la métaheuristique à définir un ordonnancement en connaissant l'ensemble des informations dès le début.

Les résultats montrent donc que l'utilisation d'une métaheuristique à la place de règles de gestion permet d'obtenir des résultats de meilleure qualité dans un environnement dynamique de production.

Tableau 7 Déviations (%) entre les règles de gestion et la métaheuristique

	LB+FIFO			SPT+FIFO		
	$DIA_{EXPO(5)}$	$DIA_{EXPO(1)}$	DIA_0	$DIA_{EXPO(5)}$	$DIA_{EXPO(1)}$	DIA_0
Kacem01	0,06	9,12	12,17	20,03	143,57	150,09
Kacem02	0,02	5,11	11,31	0,16	35,54	43,47
Kacem03	0,04	0,53	19,70	0,15	25,27	120,07
Kacem04	0	1,65	14,85	0,04	34,40	97,76
Mk10	4,04	4,61	4,01	55,40	58,52	48,38

4.5. Conclusion

Dans cette section, une approche réactive pour le problème de Job-shop Flexible est proposée. Un couplage simulation/optimisation est mis en œuvre, avec l'environnement de simulation Arena et une métaheuristique basée sur le GRASP-mELS proposé dans le chapitre précédent. Dans l'approche étudiée, le module de simulation génère de nouvelles entités (jobs) de manière aléatoire et leur affecte un type. Le type va définir la séquence des opérations pour le nouveau job avec les machines qui peuvent être affectées à chacune d'entre elles. A chaque nouvelle entité entrant dans le système un appel est fait au module d'optimisation qui a pour objectif de prendre en compte les opérations déjà effectuées et qui recréé par conséquent un graphe associé au nouveau problème à résoudre. Lorsqu'une solution est obtenue, elle est réinjectée dans le module de simulation qui applique les modifications proposées par le module d'optimisation. Une comparaison est faite avec deux règles classiques de gestion. Des expérimentations numériques sont proposées sur cinq instances. Les résultats montrent les capacités du module d'optimisation à prendre des décisions pertinentes dans l'affectation et l'ordonnancement des opérations dans un environnement dynamique. Un problème subsiste cependant lorsque la taille des instances augmente, car le module d'optimisation requiert alors un certain temps afin d'obtenir une solution globale de bonne qualité du fait notamment de la recherche locale, et les appels répétés à la métaheuristique au fur et à mesure que les entités arrivent dans le système affectent également les temps de calcul.

Finalement, cette étude préliminaire montre que dans le cadre d'un environnement réactif une métaheuristique permet d'obtenir de meilleures performances lorsque les durées d'inter-arrivées sont courtes, tandis que les règles de gestion, même simples, restent appropriées lorsque les durées d'inter-arrivées sont plus élevées, les deux méthodes convergent vers les mêmes solutions dans ce dernier cas. Des recherches supplémentaires vont être conduites sur le principe d'intégration d'un module d'optimisation au sein d'un module de simulation, en réduisant par exemple le nombre d'appels faits au module d'optimisation en utilisant en complément des règles de gestion. La métaheuristique utilisée a été développée pour le cas statique, et il apparaît qu'elle reste sensible au passage à l'échelle (i.e. lorsque le nombre d'opérations devient très important). La définition d'une nouvelle métaheuristique ou l'application d'une méthode existante pour les problèmes de grande taille constituent également des perspectives d'études.

5. Conclusion

Ce chapitre présente les approches couplant Optimisation et Simulation mises en œuvre au cours de la thèse. En reposant sur la littérature, une première méthode visant à définir des ordonnancements robustes est proposée. Cette méthode basée sur une métaheuristique utilise un module de simulation développé avec le langage SIMAN qui vise à évaluer la performance des ordonnancements explorés par la métaheuristique. Les opérations ont des durées déterministes dans la métaheuristique, tandis que la durée suit une loi normale dans le modèle de simulation. L'objectif vise à explorer les solutions déterministes à l'aide d'une métaheuristique et d'en évaluer la robustesse à l'aide d'un module de simulation. Du fait de la durée inhérente aux modèles de simulation, la métaheuristique bénéficie d'une mémoire à long terme qui permet de ne pas simuler de nouveau les mêmes solutions. Les résultats obtenus à partir d'instances de la littérature montrent que les meilleures solutions du point de vue de la métaheuristique ne sont pas toujours sauvegardées en tant que meilleures solutions simulées. De plus, il apparaît que plusieurs ordonnancements optimaux ne présentent pas les mêmes avantages lorsque des durées aléatoires sont considérées.

Dans une deuxième partie, une première tentative de résolution d'un Job-shop Flexible réactif est proposée. L'objectif est de redéfinir l'ordonnancement et l'affectation des opérations lorsque de nouveaux jobs doivent être intégrés à la production en cours. La littérature ne traite généralement pas du Job-shop Flexible réactif, l'approche proposée vise donc à évaluer la performance d'une métaheuristique pour les prises de décision au sein d'un système dynamique. Si les résultats obtenus à partir des jeux de données testés montrent que la métaheuristique est capable de retourner de bonnes solutions globales, il apparaît que lorsque les durées inter-arrivées entre les entités sont importantes, les écarts entre les solutions moyennes observées avec des règles de gestion ne sont pas suffisamment importants pour rejeter l'utilisation de règles de gestion dynamique. En outre, les temps de calcul observés sont généralement en faveur des règles de gestion, car ces dernières sont appliquées pour des prises de décision locales et directement au sein de l'environnement de simulation et ne sont dès lors pas impactées par les échanges entre le module d'optimisation et le module de simulation.

Les travaux sur le couplage optimisation/simulation ont été appliqués sur un projet industriel, le projet ECOTHER. Dans le cadre de ce couplage Optimisation/Simulation, l'ordonnancement des opérations est obtenu à l'aide d'une métaheuristique et il est ensuite évalué à l'aide du module de simulation. Celui-ci a été développé par l'IRCCyN (Nantes). Il repose sur l'utilisation d'une bibliothèque spécifiquement développée pour les besoins du projet. Elle permet de créer un modèle complet et autonome au sein d'un environnement de Simulation (Arena) et de simuler le fonctionnement d'un atelier faisant intervenir des machines pour lesquelles les phases de production présentent des profils de puissance. La principale force de cette bibliothèque réside dans la possibilité de prendre en compte des profils de puissance précis correspondant aux phases de production grâce à des mesures sur des machines industrielles, ou de définir un profil de puissance mathématiquement. Les travaux sur cette bibliothèque ont fait l'objet d'une publication (Kouki et al., 2017). Dans le cadre de ce couplage des hypothèses sont faites au sein du module d'optimisation afin d'en améliorer les performances, principalement au niveau des temps de calcul. Le fait d'avoir deux outils, l'un orienté vers la définition d'ordonnancements respectant un seuil de puissance tout en considérant des profils de puissance représentés plus simplement, l'autre intégrant plus finement les profils de puissance de chaque ressource sans gérer l'ordre de passage des entités, permet d'avoir un couplage permettant de bénéficier des forces et faiblesses de chacun des outils. Ainsi, les ordonnancements générés pour le cas d'étude permettent d'atteindre des solutions de meilleure qualité lorsque l'on considère le seuil de puissance global, par rapport aux solutions initialement simulées qui ne disposaient pas de l'apport de

l'optimisation. De manière générale, une réduction de 5 à 10% des puissances nécessaires est observée lorsque l'ordonnancement retourné par le module d'optimisation est utilisé. Ces résultats valident donc l'approche et montrent que l'utilisation d'un module d'optimisation, même en relâchant certaines contraintes, permet d'obtenir des résultats meilleurs que la seule utilisation de la simulation.

Parmi les perspectives qui sont apparues avec les travaux réalisés, l'amélioration des performances dans le cadre du Job-shop Flexible réactif semble une direction intéressante. En effet, l'optimisation globale d'un problème conduit généralement à de meilleures solutions que des prises de décision locales, aussi pertinentes soient-elles. Si les résultats retournés par la métaheuristique sont de bonne qualité, les temps de calcul observés dans le cadre du Job-shop Flexible réactif sont prohibitifs pour que l'approche puisse être utilisée dans un contexte industriel. La réduction des temps de calcul est donc un enjeu essentiel. Aussi, une comparaison avec le solveur OptQuest embarqué dans Arena pourrait donner des résultats intéressants. Des procédures d'optimisation basées sur des heuristiques associées à des recherches locales peuvent être également une perspective de recherche, ou encore la combinaison de règles de décision et d'une métaheuristique appelée lorsqu'un nombre donné d'entités a intégré le système. Cette dernière approche peut permettre d'accorder plus de temps à une métaheuristique appelée moins souvent et pourrait conduire à des résultats intéressants.

Par la suite, les approches d'optimisation/simulation présentées dans ce chapitre seront utilisées pour la résolution de problèmes impliquant des enjeux financiers dans le cadre de l'optimisation d'une supply chain. L'objectif étant d'enrichir les travaux présentés dans les conférences META 2014 (Kemmoe et al., 2014) et ICORES 2015 (Kemmoe et al., 2015a) en permettant de redéfinir les dates de paiement des sous-traitants lorsque de nouvelles opérations arrivent et requièrent des besoins en trésorerie pour être effectuées, ou alors de simuler la capacité d'un ordonnancement à respecter les dates d'échéance lorsque des débits et crédits sont pris en compte. Comme mentionné par (Ouelhadj and Petrovic, 2009) l'utilisation d'un système de connaissance fait partie des perspectives de cette thèse dans la définition de systèmes réactifs, une métaheuristique pouvant bénéficier des informations connues pour en améliorer les performances.

Conclusion générale

Au cours de cette thèse, nous nous sommes intéressés aux problèmes d'optimisation des systèmes de production. Ces problèmes sont de différentes natures et vont de la considération de contraintes énergétiques à la conception de procédures couplant optimisation et simulation.

Dans un premier temps, nous avons identifié un problème classique d'ordonnancement sur lequel reposent la plupart des outils que nous mettons en œuvre au sein de cette thèse : le Job-shop. Il est apparu que les systèmes de voisinages sont au cœur de toute procédure d'optimisation efficace pour ces problèmes d'optimisation difficiles. Après une présentation de différentes méthodes de résolution et de représentation des solutions, nous avons proposé une approche basée sur une métaheuristique de type GRASP×ELS. La métaheuristique a montré qu'elle était capable de retourner des solutions pertinentes en un laps de temps relativement court, et qu'elle était aussi performante que des méthodes actuelles.

La thématique principale de cette thèse concerne l'optimisation des systèmes de production soumis à des contraintes énergétiques. Ce sujet a été abordé dans le troisième chapitre. A travers une analyse de la littérature existante et plus particulièrement de celle relative à l'ordonnancement, il est apparu que peu d'articles traitent de l'ordonnancement sous contrainte énergétique, bien que le nombre d'articles soit en constante augmentation depuis les années 2010. Cependant, de nombreux problèmes restent à étudier et l'application systématique des méthodes au sein d'environnements industriels reste à mettre en œuvre. Un de ces problèmes concerne le Job-shop avec seuil de puissance. En effet, la plupart des articles d'ordonnancement des systèmes de production sont principalement orientés vers la réduction de la consommation totale ou des coûts lors de tarifications par tranche horaire. Peu de travaux prennent en compte la puissance instantanée requise par le système de production, alors que c'est un poste de dépense important pour les industriels et tout outil d'optimisation dans ce domaine peut s'avérer être un moyen de négociation durant les phases de contractualisation. Fort de ce constat, un premier travail a été de réaliser un modèle mathématique d'un système de production de type Job-shop. En plus de prendre en compte un seuil de puissance, il a été décidé de modéliser les opérations à ordonnancer comme des opérations présentant des pics de consommation en puissance au début de l'opération, alors que la plupart des modélisations de la littérature représentent des opérations à consommation constante, ou traitent de la consommation énergétique totale, qui est une valeur agrégée. Par la suite, nous avons pris en compte le fait que le seuil de puissance peut être variable afin de modéliser différents aléas sur la puissance injectée au système (maintenance côté fournisseur, intégration d'énergies renouvelables…) ou pour simplement prendre en compte la puissance contractualisée. Des instances ont été développées pour le problème ; cependant, en raison de la difficulté d'obtenir des résultats en un temps raisonnable à l'aide d'un outil de résolution exacte (CPLEX), il a été choisi de proposer une approche par métaheuristiques afin d'obtenir des résultats de qualité en un temps fortement réduit. Les instances générées ont été résolues à l'aide de trois métaheuristiques : un GRASP×ELS, un VNS et un MA. Un autre aspect important dans un contexte industriel où les consommations énergétiques sont prises en compte est de trouver un compromis entre la qualité de la solution d'un point de vue de la productivité, et du point de vue de la puissance instantanée. Un tel objectif revient à proposer un front de Pareto des solutions qui présentent des niveaux de performance différents selon le critère considéré. C'est dans cet objectif qu'a été réalisée une métaheuristique de type NSGA-II. Elle est comparée à une évolution de la métaheuristique GRASP×ELS utilisée dans le cas du seuil de puissance variable. Les résultats montrent la complémentarité des deux approches, avec un NSGA-II globalement plus performant

même si le GRASP×ELS présente un intérêt dans les systèmes où un seuil de puissance élevé est autorisé.

Un autre sujet de recherche traité dans ce travail de doctorat concerne l'ordonnancement des systèmes flexibles de production avec l'étude d'une extension du problème de Job-shop : le Job-shop Flexible (FJSP). Dans ce problème, l'affectation des opérations aux machines est à gérer par le module d'optimisation. La considération du problème d'ordonnancement et de celui d'affectation rendent le Job-shop Flexible plus difficile à résoudre que le Job-shop classique. Pour apporter des solutions à ce problème, une nouvelle métaheuristique a été proposée. Elle consiste en une recherche locale évolutionnaire (ELS) itérée, associée à un GRASP. Une heuristique de construction basée sur une heuristique performante de la littérature pour le Job-shop a également été proposée. Les résultats obtenus sont comparables aux travaux existants sur le sujet, avec un nombre de solutions optimales comparable à l'une des meilleures approches publiées sur le problème, et ce en des temps de calcul raisonnables.

Un des points de la thèse concerne également le couplage simulation/optimisation au sein d'un projet industriel (ECOTHER). La simulation présente un grand intérêt lors de l'étude de systèmes présentant des événements aléatoires (problèmes stochastiques), ou de fortes incertitudes quant à leur fonctionnement. Une étude a été faite sur la résolution d'un problème stochastique reposant sur le Job-shop. Il a été choisi de traiter du problème de robustesse d'un tel système où les durées de traitement des opérations suivent une loi Normale avec un écart-type important représentant des opérations avec un fort degré d'incertitude. L'objectif est de proposer des ordonnancements stables, dont le critère de performance est moins sensible aux aléas sur les temps de traitement. Les résultats démontrent que les ordonnancements valides d'un point de vue déterministe (aucun aléa) peuvent ne pas être les meilleurs ordonnancements une fois que des événements aléatoires sont considérés. Ils permettent aussi de remarquer plus facilement l'existence de plusieurs solutions optimales pour un problème donné dont les performances sont notablement différentes lors de la considération d'événements aléatoires. La deuxième approche consiste à faire appel à un module d'optimisation au sein d'un environnement de simulation lorsque des aléas se produisent. Un cas simple consistant à réordonnancer des jobs au sein d'un système flexible de production a été abordé. Nous montrons également que cette approche permet d'obtenir en moyenne des solutions meilleures que des règles classiques de gestion telles que les règles « premier arrivé premier servi ».

De nombreuses perspectives de recherche sont apparues lors des travaux mentionnés ci-dessus. Ainsi, la résolution de problèmes d'ordonnancement dans un environnement industriel de type FJSP avec considération de puissances et de machines qui peuvent être allumées ou éteintes semble être une direction intéressante. L'objectif est de proposer des ordonnancements plus performants en bénéficiant de la puissance non utilisée par le système de production lorsque des machines sont éteintes. Une autre activité de recherche concerne l'optimisation de systèmes flexibles de production dans un contexte incertain et soumis à des contraintes énergétiques. L'objectif est de proposer un réordonnancement rapide des tâches lorsque des aléas interviennent. Il faut alors prendre en compte les opérations déjà effectuées et celles à venir dans le processus de planification tout en considérant les consommations énergétiques associées aux opérations. Un autre problème particulièrement intéressant concerne le problème biobjectif visant à déterminer un ensemble de solutions pour un problème de Job-shop avec seuil de puissance variable. Si l'on considère également des coûts liés à la consommation énergétique, la résolution d'un problème ainsi considéré peut être particulièrement intéressante.

Parallèlement à ces perspectives, la résolution d'un problème de Group-shop est également une piste à suivre, en cela que ce problème est à la frontière entre le Job-shop et l'Open-shop et qu'il permet de traiter l'un et l'autre. Additionné de contraintes énergétiques, un tel problème participerait davantage à la réduction des consommations énergétiques des systèmes de production.

Bibliographie

Abedinnia, H., Glock, C.H., Grosse, E.H., Schneider, M., 2017. Machine scheduling problems in production: A tertiary study. Comput. Ind. Eng. 111, 403–416. https://doi.org/10.1016/j.cie.2017.06.026

Adibi, M.A., Zandieh, M., Amiri, M., 2010. Multi-objective scheduling of dynamic job shop using variable neighborhood search. Expert Syst. Appl. 37, 282–287. https://doi.org/10.1016/j.eswa.2009.05.001

Ahmadi, E., Zandieh, M., Farrokh, M., Emami, S.M., 2016. A multi objective optimization approach for flexible job shop scheduling problem under random machine breakdown by evolutionary algorithms. Comput. Oper. Res. 73, 56–66. https://doi.org/10.1016/j.cor.2016.03.009

Ahmadi-Javid, A., Hooshangi-Tabrizi, P., 2017. Integrating Employee Timetabling with Scheduling of Machines and Transporters in a Job Shop Environment: A Mathematical Formulation and an Anarchic Society Optimization Algorithm. Comput. Oper. Res.

Al-Hinai, N., ElMekkawy, T.Y., 2011. Robust and stable flexible job shop scheduling with random machine breakdowns using a hybrid genetic algorithm. Int. J. Prod. Econ. 132, 279–291. https://doi.org/10.1016/j.ijpe.2011.04.020

Annual Energy Outlook, 2017. . Energy Information Administration.

Anthony, R.N., 1965. Planning and Control Systems: A Framework for Analysis. Harvard Business School Press.

Avez, G., Lacomme, P., Lamy, D., Tchernev, N., Phan, R., 2016. First experiment of STORM for design of efficient optimization methods: application to the Job-shop with Time-lags, in: 11th International Conference on Modeling, Optimization and Simulation. Presented at the MOSIM, Montréal.

Baker, K.R., 1974. Introduction to sequencing and scheduling. Wiley.

Balas, E., Vazacopoulos, A., 1998. Guided Local Search with Shifting Bottleneck for Job Shop Scheduling. Manag. Sci. 44, 262–275. https://doi.org/10.1287/mnsc.44.2.262

Bechikh, S., Elarbi, M., Ben Said, L., 2017. Many-objective Optimization Using Evolutionary Algorithms: A Survey, in: Bechikh, S., Datta, R., Gupta, A. (Eds.), Recent Advances in Evolutionary Multi-Objective Optimization. Springer International Publishing, Cham, pp. 105–137. https://doi.org/10.1007/978-3-319-42978-6_4

Bellman, R., 1958. On a Routing Problem. Quarterly of Applied Mathematics 16, 87–90.

Ben Hmida, A., Haouari, M., Huguet, M.-J., Lopez, P., 2010. Discrepancy search for the flexible job shop scheduling problem. Comput. Oper. Res. 37, 2192–2201. https://doi.org/10.1016/j.cor.2010.03.009

Biel, K., Glock, C.H., 2016. Systematic literature review of decision support models for energy-efficient production planning. Comput. Ind. Eng. 101, 243–259. https://doi.org/10.1016/j.cie.2016.08.021

Bierwirth, C., 1995. A generalized permutation approach to job shop scheduling with genetic algorithms. Oper.-Res.-Spektrum 17, 87–92.

Bierwirth, C., Mattfeld, D.C., Kopfer, H., 1996. On permutation representations for scheduling problems, in: International Conference on Parallel Problem Solving from Nature. Springer, pp. 310–318.

Binato, S., Hery, W., Loewenstern, D., Resende, M., 2000. A GRASP for job shop scheduling. Essays Surv. Metaheuristics.

Blazewicz, J., Domschke, W., Pesch, E., 1996. The job shop scheduling problem: Conventional and new solution techniques. Eur. J. Oper. Res. 93, 1–33.

Blazewicz, J., Ecker, K.H., Pesch, E., Schmidt, G., Weglarz, J., 1996. Scheduling Computer and Manufacturing Processes. Springer-Verlag Berlin Heidelberg GmbH.

Booch, G., Rumbaugh, J., Jacobson, I., 2000. Le guide de l'utilisateur UML, Technologies objet. ed. Eyrolles.

Borel, E., 1909. Les probabilités dénombrables et leurs applications arithmetiques. Rendiconti del Circolo Matematico di Palermo 27, 247–271.

Boussaïd, I., Lepagnot, J., Siarry, P., 2013. A survey on optimization metaheuristics. Inf. Sci. 237, 82–117. https://doi.org/10.1016/j.ins.2013.02.041

Brah, S.A., Hunsucker, J.L., 1991. Branch and bound algorithm for the flow shop with multiple processors. Eur. J. Oper. Res. 51, 88–99.

Brandimarte, P., 1993. Routing and scheduling in a flexible job shop by tabu search. Ann. Oper. Res. 41, 157–183.

Brucker, P., Schlie, R., 1990. Job-shop scheduling with multi-purpose machines. Computing 45, 369–375.

Bruzzone, A.A.G., Anghinolfi, D., Paolucci, M., Tonelli, F., 2012. Energy-aware scheduling for improving manufacturing process sustainability: A mathematical model for flexible flow shops. CIRP Ann. - Manuf. Technol. 61, 459–462. https://doi.org/10.1016/j.cirp.2012.03.084

Cardin, O., 2007. Apport de la simulation en ligne dans l'aide à la décision pour le pilotage des systèmes de production–application à un système flexible de production. Université de Nantes.

Carlier, J., Chrétienne, P., 1988. Problèmes d'ordonnancement: modélisation, complexité, algorithmes.

Caumond, A., 2006. Le problème de jobshop avec contraintes: modélisation et optimisation. Université Blaise Pascal-Clermont-Ferrand II.

Caumond, A., Lacomme, P., Tchernev, N., 2008. A memetic algorithm for the job-shop with time-lags. Comput. Oper. Res. 35, 2331–2356. https://doi.org/10.1016/j.cor.2006.11.007

Černỳ, V., 1985. Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm. J. Optim. Theory Appl. 45, 41–51.

Chambers, J., Barnes, J.W., 1998. Reactive search for flexible job shop scheduling. Grad. Program Oper. Res. Ind. Eng. Univ. Tex. Austin Tech. Rep. Ser. ORP98-04.

Chassaing, M., Fontanel, J., Lacomme, P., Ren, L., Tchernev, N., Villechenon, P., 2014. A GRASP×ELS approach for the job-shop with a web service paradigm packaging. Expert Syst. Appl. 41, 544–562. https://doi.org/10.1016/j.eswa.2013.07.080

Chaudhry, I.A., Khan, A.A., 2016. A research survey: review of flexible job shop scheduling techniques. Int. Trans. Oper. Res. 23, 551–591. https://doi.org/10.1111/itor.12199

Che, A., Zeng, Y., Lyu, K., 2016. An efficient greedy insertion heuristic for energy-conscious single machine scheduling problem under time-of-use electricity tariffs. J. Clean. Prod. 129, 565–577. https://doi.org/10.1016/j.jclepro.2016.03.150

Chen, P.P.-S., 1976. The entity-relationship model—toward a unified view of data. ACM Trans. Database Syst. TODS 1, 9–36.

Cheng, R., Gen, M., Tsujimura, Y., 1996. A tutorial survey of job-shop scheduling problems using genetic algorithms - I. Representation. Comput. Ind. Eng. 30, 983–997.

Cheng, T.C.E., Peng, B., Lü, Z., 2016. A hybrid evolutionary algorithm to solve the job shop scheduling problem. Ann. Oper. Res. 242, 223–237. https://doi.org/10.1007/s10479-013-1332-5

Collette, Y., Siarry, P., 2002. Optimisation multiobjectif. Eyrolles, Paris.

Dai, M., Tang, D., Giret, A., Salido, M.A., Li, W.D., 2013. Energy-efficient scheduling for a flexible flow shop using an improved genetic-simulated annealing algorithm. Robot. Comput.-Integr. Manuf. 29, 418–429. https://doi.org/10.1016/j.rcim.2013.04.001

Dauzère-Pérès, S., Paulli, J., 1997. An integrated approach for modeling and solving the general multiprocessor job-shop scheduling problem using tabu search. Ann. Oper. Res. 70, 281–306.
Deb, K., Agrawal, R.B., 1995. Simulated Binary Crossover for Continuous Search Space. Complex Syst. 9.
Deb, K., Pratap, A., Agarwal, S., Meyarivan, T., 2002. A fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE Trans. Evol. Comput. 6, 182–197.
Deming, W.E., 1986. Out of the Crisis. The MIT Press.
Ding, J.-Y., Song, S., Wu, C., 2016. Carbon-efficient scheduling of flow shops by multi-objective optimization. Eur. J. Oper. Res. 248, 758–771. https://doi.org/10.1016/j.ejor.2015.05.019
Dongarra, J.J., 2014. Performance of various computers using standard linear equations software. University of Tennessee. Computer Science Department.
Duflou, J.R., Sutherland, J.W., Dornfeld, D., Herrmann, C., Jeswiet, J., Kara, S., Hauschild, M., Kellens, K., 2012. Towards energy and resource efficient manufacturing: A processes and systems approach. CIRP Ann. - Manuf. Technol. 61, 587–609. https://doi.org/10.1016/j.cirp.2012.05.002
Duhamel, C., Lacomme, P., Prins, C., Prodhon, C., 2010. A GRASP×ELS approach for the capacitated location-routing problem. Comput. Oper. Res. 37, 1912–1923. https://doi.org/10.1016/j.cor.2009.07.004
Fang, K., Uhan, N., Zhao, F., Sutherland, J.W., 2011. A new approach to scheduling in manufacturing for power consumption and carbon footprint reduction. J. Manuf. Syst. 30, 234–240. https://doi.org/10.1016/j.jmsy.2011.08.004
Feo, T.A., Resende, M.G., 1995. Greedy randomized adaptive search procedures. J. Glob. Optim. 6, 109–133.
Figueira, G., Almada-Lobo, B., 2014. Hybrid simulation–optimization methods: A taxonomy and discussion. Simul. Model. Pract. Theory 46, 118–134. https://doi.org/10.1016/j.simpat.2014.03.007
Fisher, H., Thompson, G.L., 1963. Probabilistic learning combinations of local job shop scheduling rules, in: Industrial Scheduling. Prentice Hall, pp. 225–251.
Gabel, T., Riedmiller, M., 2008. Adaptive reactive job-shop scheduling with reinforcement learning agents. Int. J. Inf. Technol. Intell. Comput. 24.
Gao, J., Sun, L., Gen, M., 2008. A hybrid genetic and variable neighborhood descent algorithm for flexible job shop scheduling problems. Comput. Oper. Res. 35, 2892–2907. https://doi.org/10.1016/j.cor.2007.01.001
Gao, L., Li, X., Wen, X., Lu, C., Wen, F., 2015. A hybrid algorithm based on a new neighborhood structure evaluation method for job shop scheduling problem. Comput. Ind. Eng. 88, 417–429. https://doi.org/10.1016/j.cie.2015.08.002
Garcia-Santiago, C.A., Del Ser, J., Upton, C., Quilligan, F., Gil-Lopez, S., Salcedo-Sanz, S., 2015. A random-key encoded harmony search approach for energy-efficient production scheduling with shared resources. Eng. Optim. 47, 1481–1496. https://doi.org/10.1080/0305215X.2014.971778
Garey, M.R., Johnson, D.S., 1979. Computers and intractability: a guide to the theory of NP-completeness, A series of books in the mathematical sciences. W. H. Freeman & Co, New York, NY, USA.
Garey, M.R., Johnson, D.S., Sethi, R., 1976. The Complexity of Flowshop and Jobshop Scheduling. Math. Oper. Res. 1, 117–129. https://doi.org/10.1287/moor.1.2.117
Gholami, M., Zandieh, M., 2009. Integrating simulation and genetic algorithm to schedule a dynamic flexible job shop. J. Intell. Manuf. 20, 481–498. https://doi.org/10.1007/s10845-008-0150-0

Giret, A., Trentesaux, D., Prabhu, V., 2015. Sustainability in manufacturing operations scheduling: A state of the art review. J. Manuf. Syst. 37, 126–140. https://doi.org/10.1016/j.jmsy.2015.08.002

Glover, F., 1986. Future paths for integer programming and links to artificial intelligence. Comput. Oper. Res. 13, 533–549.

Golenko-Ginzburg, D., Gonik, A., 2002. Optimal job-shop scheduling with random operations and cost objectives. Int. J. Prod. Econ. 76, 147–157.

Gonçalves, J.F., de Magalhães Mendes, J.J., Resende, M.G.C., 2005. A hybrid genetic algorithm for the job shop scheduling problem. Eur. J. Oper. Res. 167, 77–95. https://doi.org/10.1016/j.ejor.2004.03.012

González Fernández, M.Á., Rodríguez Vela, M. del C., Varela Arias, J.R., 2013. An efficient memetic algorithm for the flexible job shop with setup times, in: Proceedings of the Twenty-Third International Conference on Automated Planning and Scheduling. Association for the Advancement of Artificial Intelligence (AAAI).

González, M.A., Vela, C.R., Varela, R., 2015. Scatter search with path relinking for the flexible job shop scheduling problem. Eur. J. Oper. Res. 245, 35–45. https://doi.org/10.1016/j.ejor.2015.02.052

Gonzalez, T., Sahni, S., 1976. Open shop scheduling to minimize finish time. J. ACM JACM 23, 665–679.

Gourgand, M., 1984. Outils logiciels pour l'évaluation des performances des systèmes informatiques. Université Blaise Pascal-Clermont-Ferrand II.

Grabowski, J., Nowicki, E., Zdrzalka, S., 1986. A block approach for single-machine scheduling with release dates and due dates. Eur. J. Oper. Res. 26, 278–285.

Graham, R.L., Lawler, E.L., Lenstra, J.K., Rinnooy Kan, A.H.., 1979. Optimization and approximation in deterministic sequencing and scheduling: a survey. Ann. Discrete Math. 5, 287–326.

Grangeon, N., 2001. Métaheuristiques et modèles d'évaluation pour le problème du flow shop hybride hiérarchisé. Université Blaise Pascal-Clermont-Ferrand II.

Grimes, D., Ifrim, G., O'Sullivan, B., Simonis, H., 2014. Analyzing the impact of electricity price forecasting on energy cost-aware scheduling. Sustain. Comput. Inform. Syst. 4, 276–291. https://doi.org/10.1016/j.suscom.2014.08.009

Gu, J., Gu, M., Cao, C., Gu, X., 2010. A novel competitive co-evolutionary quantum genetic algorithm for stochastic job shop scheduling problem. Comput. Oper. Res. 37, 927–937. https://doi.org/10.1016/j.cor.2009.07.002

Gu, J., Gu, X., Gu, M., 2009. A novel parallel quantum genetic algorithm for stochastic job shop scheduling. J. Math. Anal. Appl. 355, 63–81. https://doi.org/10.1016/j.jmaa.2008.12.065

Gupta, A., Venkataraman, S., 2013. Reducing Price Volatility of Electricity Consumption for a Firm's Energy Risk Management. Electr. J. 26, 89–105. https://doi.org/10.1016/j.tej.2013.03.005

Gutowski, T., Dahmus, J., Thiriez, A., 2006. Electrical energy requirements for manufacturing processes, in: 13th CIRP International Conference on Life Cycle Engineering. pp. 623–638.

Gutowski, T., Murphy, C., Allen, D., Bauer, D., Bras, B., Piwonka, T., Sheng, P., Sutherland, J., Thurston, D., Wolff, E., 2005. Environmentally benign manufacturing: Observations from Japan, Europe and the United States. J. Clean. Prod. 13, 1–17. https://doi.org/10.1016/j.jclepro.2003.10.004

Haït, A., Artigues, C., 2011. On electrical load tracking scheduling for a steel plant. Comput. Chem. Eng. 35, 3044–3047. https://doi.org/10.1016/j.compchemeng.2011.03.006

Hao, X., Gen, M., Lin, L., Suer, G.A., 2015. Effective multiobjective EDA for bi-criteria stochastic job-shop scheduling problem. J. Intell. Manuf. https://doi.org/10.1007/s10845-014-1026-0

Hasan, S.M.K., Sarker, R., Essam, D., Cornforth, D., 2009. Memetic algorithms for solving job-shop scheduling problems. Memetic Comput. 1, 69–83. https://doi.org/10.1007/s12293-008-0004-5

He, Y., Li, Y., Wu, T., Sutherland, J.W., 2015. An energy-responsive optimization method for machine tool selection and operation sequence in flexible machining job shops. J. Clean. Prod. 87, 245–254. https://doi.org/10.1016/j.jclepro.2014.10.006

Holland, J.H., 1975. Adaptation in Natural and Artificial Systems. University of Michigan Press, Ann Arbor, MI, USA.

Hoos, H.H., Stützle, T., 2005. Stochastic local search: foundations and applications. Morgan Kaufmann Publishers, San Francisco, CA.

Horn, W.A., 1973. Technical Note - Minimizing average flow time with parallel machines. Oper. Res. 21, 846–847.

Horng, S.-C., Lin, S.-S., Yang, F.-Y., 2012. Evolutionary algorithm for stochastic job shop scheduling with random processing time. Expert Syst. Appl. 39, 3603–3610. https://doi.org/10.1016/j.eswa.2011.09.050

Hu, Q., Lim, A., 2014. An iterative three-component heuristic for the team orienteering problem with time windows. Eur. J. Oper. Res. 232, 276–286. https://doi.org/10.1016/j.ejor.2013.06.011

Hurink, J., Jurisch, B., Thole, M., 1994. Tabu search for the job-shop scheduling problem with multi-purpose machines. Oper.-Res.-Spektrum 15, 205–215.

Jain, A.S., Meeran, S., 1999. Deterministic job-shop scheduling: Past, present and future. Eur. J. Oper. Res. 113, 390–434.

Juan, A.A., Barrios, B.B., Vallada, E., Riera, D., Jorba, J., 2014. A simheuristic algorithm for solving the permutation flow shop problem with stochastic processing times. Simul. Model. Pract. Theory 46, 101–117. https://doi.org/10.1016/j.simpat.2014.02.005

Kacem, I., Hammadi, S., Borne, P., 2002. Approach by localization and multiobjective evolutionary optimization for flexible job-shop scheduling problems. IEEE Trans. Syst. Man Cybern. Part C Appl. Rev. 32, 1–13.

Kellert, P., Tchernev, N., Force, C., 1997. Object-oriented methodology for FMS modelling and simulation. Int. J. Comput. Integr. Manuf. 10, 405–434. https://doi.org/10.1080/095119297131002

Kemmoe, S., Lamy, D., Tchernev, N., 2017. Job-shop like manufacturing system with variable power threshold and operations with power requirements. Int. J. Prod. Res. 1–22. https://doi.org/10.1080/00207543.2017.1321801

Kemmoe, S., Lamy, D., Tchernev, N., 2016. A GRASP embedding a bi-level ELS for solving Flexible Job-shop Problems. IFAC, Troyes, pp. 1749–1754.

Kemmoe, S., Lamy, D., Tchernev, N., 2015a. An Optimization Approach for Job-shop with Financial Constraints - In the Context of Supply Chain Scheduling Considering Payment Delay between Members. SCITEPRESS - Science and and Technology Publications, Lisbonne, pp. 190–198. https://doi.org/10.5220/0005271301900198

Kemmoe, S., Lamy, D., Tchernev, N., 2015b. A Job-shop with an Energy Threshold Issue Considering Operations with Consumption Peaks. Presented at the INCOM, Ottawa, pp. 788–793.

Kemmoe, S., Lamy, D., Tchernev, N., 2015c. A job-shop with an energy threshold issue considering operations with consumption peaks. IFAC-Pap. 48, 788–793.

Kemmoe, S., Lamy, D., Tchernev, N., 2014. A GRASPxELS for supply chain optimization considering payment delay between members. Presented at the META, Morocco.

Kemmoé-Tchomté, S., Lamy, D., Tchernev, N., 2017. An effective multi-start multi-level evolutionary local search for the flexible job-shop problem. Eng. Appl. Artif. Intell. 62, 80–95. https://doi.org/10.1016/j.engappai.2017.04.002

Kemmoé-Tchomté, S., Lamy, D., Tchernev, N., 2015a. Job-Shop like Manufacturing System with Time Dependent Energy Threshold and Operations with Peak Consumption, in: Umeda, S., Nakano, M., Mizuyama, H., Hibino, N., Kiritsis, D., von Cieminski, G. (Eds.), Advances in Production Management Systems: Innovative Production Management Towards Sustainable Growth. Springer International Publishing, Cham, pp. 617–624. https://doi.org/10.1007/978-3-319-22756-6_75

Kemmoé-Tchomté, S., Lamy, D., Tchernev, N., 2015b. A metaheuristic based on simulation for stochastic Job-shop optimization, in: Industrial Engineering and Systems Management (IESM), 2015 International Conference On. IEEE, Seville, pp. 108–116.

Kirkpatrick, S., Gelatt, C.D., Vecchi, M.P., others, 1983. Optimization by simulated annealing. science 220, 671–680.

Kouki, M., Cardin, O., Castagna, P., Cornardeau, C., 2017. Input data management for energy related discrete event simulation modelling. J. Clean. Prod. 141, 194–207. https://doi.org/10.1016/j.jclepro.2016.09.061

Kruskal, J.B., 1956. On the shortest spanning subtree of a graph and the traveling salesman problem. Proc. Am. Math. Soc. 7, 48–50.

Kuhpfahl, J., Bierwirth, C., 2016. A study on local search neighborhoods for the job shop scheduling problem with total weighted tardiness objective. Comput. Oper. Res. 66, 44–57. https://doi.org/10.1016/j.cor.2015.07.011

Kurdi, M., 2015. A new hybrid island model genetic algorithm for job shop scheduling problem. Comput. Ind. Eng. 88, 273–283. https://doi.org/10.1016/j.cie.2015.07.015

Lacomme, P., Larabi, M., Tchernev, N., 2013a. Job-shop based framework for simultaneous scheduling of machines and automated guided vehicles. Int. J. Prod. Econ. 143, 24–34. https://doi.org/10.1016/j.ijpe.2010.07.012

Lacomme, P., Prins, C., Sevaux, M., 2006. A genetic algorithm for a bi-objective capacitated arc routing problem. Comput. Oper. Res. 33, 3473–3493. https://doi.org/10.1016/j.cor.2005.02.017

Lacomme, P., Toussaint, H., Duhamel, C., 2013b. A GRASP×ELS for the vehicle routing problem with basic three-dimensional loading constraints. Eng. Appl. Artif. Intell. 26, 1795–1810. https://doi.org/10.1016/j.engappai.2013.03.012

Lawrence, S., 1984. Resource constrained project scheduling: an experimental investigation of heuristic scheduling techniques (Supplement).

Le Moigne, J.-L., 1991. La modélisation des systèmes complexes.

Lee, K., Leung, J.Y.-T., Pinedo, M.L., 2010. Makespan minimization in online scheduling with machine eligibility. 4OR 8, 331–364. https://doi.org/10.1007/s10288-010-0149-1

Lei, D., 2011. Scheduling stochastic job shop subject to random breakdown to minimize makespan. Int. J. Adv. Manuf. Technol. 55, 1183–1192. https://doi.org/10.1007/s00170-010-3151-z

Lei, D., Zheng, Y., Guo, X., 2016. A shuffled frog-leaping algorithm for flexible job shop scheduling with the consideration of energy consumption. Int. J. Prod. Res. 1–15. https://doi.org/10.1080/00207543.2016.1262082

Li, H.C., Cao, H.J., 2015. An Optimization Model for Carbon Efficiency of a Job-shop Manufacturing System. Procedia CIRP 28, 113–118. https://doi.org/10.1016/j.procir.2015.04.019

Li, X., Gao, L., 2016. An effective hybrid genetic algorithm and tabu search for flexible job shop scheduling problem. Int. J. Prod. Econ. 174, 93–110. https://doi.org/10.1016/j.ijpe.2016.01.016

Lin, L., Gen, M., Liang, Y., Ohno, K., 2012. A Hybrid EA for Reactive Flexible Job-shop Scheduling. Procedia Comput. Sci. 12, 110–115. https://doi.org/10.1016/j.procs.2012.09.039

Lin, W., Yu, D.Y., Zhang, C., Liu, X., Zhang, S., Tian, Y., Liu, S., Xie, Z., 2015. A multi-objective teaching−learning-based optimization algorithm to scheduling in turning processes for minimizing makespan and carbon footprint. J. Clean. Prod. 101, 337–347. https://doi.org/10.1016/j.jclepro.2015.03.099

Lin, Y.-K., Yeh, C.-T., 2012. Multi-objective optimization for stochastic computer networks using NSGA-II and TOPSIS. Eur. J. Oper. Res. 218, 735–746. https://doi.org/10.1016/j.ejor.2011.11.028

Linn, R., Zhang, W., 1999. Hybrid flow shop scheduling: a survey. Comput. Ind. Eng. 37, 57–61.

Liu, C.-H., 2016. Mathematical programming formulations for single-machine scheduling problems while considering renewable energy uncertainty. Int. J. Prod. Res. 54, 1122–1133. https://doi.org/10.1080/00207543.2015.1048380

Liu, C.-H., Huang, D.-H., 2014. Reduction of power consumption and carbon footprints by applying multi-objective optimisation via genetic algorithms. Int. J. Prod. Res. 52, 337–352. https://doi.org/10.1080/00207543.2013.825740

Liu, G.-S., Yang, H.-D., Cheng, M.-B., 2016. A three-stage decomposition approach for energy-aware scheduling with processing-time-dependent product quality. Int. J. Prod. Res. 1–19. https://doi.org/10.1080/00207543.2016.1241446

Liu, Y., Dong, H., Lohse, N., Petrovic, S., 2015. Reducing environmental impact of production during a Rolling Blackout policy – A multi-objective schedule optimisation approach. J. Clean. Prod. 102, 418–427. https://doi.org/10.1016/j.jclepro.2015.04.038

Liu, Y., Dong, H., Lohse, N., Petrovic, S., Gindy, N., 2014. An investigation into minimising total energy consumption and total weighted tardiness in job shops. J. Clean. Prod. 65, 87–96. https://doi.org/10.1016/j.jclepro.2013.07.060

Liu, Y., Tiwari, A., 2015. An Investigation into Minimising Total Energy Consumption and Total Completion Time in a Flexible Job Shop for Recycling Carbon Fiber Reinforced Polymer. Procedia CIRP 29, 722–727. https://doi.org/10.1016/j.procir.2015.01.063

Lu, C., Gao, L., Li, X., Pan, Q., Wang, Q., 2017. Energy-efficient permutation flow shop scheduling problem using a hybrid multi-objective backtracking search algorithm. J. Clean. Prod. 144, 228–238. https://doi.org/10.1016/j.jclepro.2017.01.011

Luo, H., Du, B., Huang, G.Q., Chen, H., Li, X., 2013. Hybrid flow shop scheduling considering machine electricity consumption cost. Int. J. Prod. Econ. 146, 423–439. https://doi.org/10.1016/j.ijpe.2013.01.028

Mahdavi, I., Shirazi, B., Solimanpur, M., 2010. Development of a simulation-based decision support system for controlling stochastic flexible job shop manufacturing systems. Simul. Model. Pract. Theory 18, 768–786. https://doi.org/10.1016/j.simpat.2010.01.015

Manne, M., 1960. On the Job-shop schediuling problem. Oper. Res. 8, 219–223.

Mastrolilli, M., Gambardella, L.M., 2000. Effective neighbourhood functions for the flexible job shop problem. J. Sched. 3, 3–20.

May, G., Barletta, I., Stahl, B., Taisch, M., 2015a. Energy management in production: A novel method to develop key performance indicators for improving energy efficiency. Appl. Energy 149, 46–61. https://doi.org/10.1016/j.apenergy.2015.03.065

May, G., Stahl, B., Taisch, M., Prabhu, V., 2015b. Multi-objective genetic algorithm for energy-efficient job shop scheduling. Int. J. Prod. Res. 53, 7071–7089. https://doi.org/10.1080/00207543.2015.1005248

Merkert, L., Harjunkoski, I., Isaksson, A., Säynevirta, S., Saarela, A., Sand, G., 2015. Scheduling and energy – Industrial challenges and opportunities. Comput. Chem. Eng. 72, 183–198. https://doi.org/10.1016/j.compchemeng.2014.05.024

Mirshekarian, S., Šormaz, D.N., 2016. Correlation of job-shop scheduling problem features with scheduling efficiency. Expert Syst. Appl. 62, 131–147. https://doi.org/10.1016/j.eswa.2016.06.014

Mladenović, N., Hansen, P., 1997. Variable neighborhood search. Comput. Oper. Res. 24, 1097–1100.

Mokhtari, H., Dadgar, M., 2015. Scheduling optimization of a stochastic flexible job-shop system with time-varying machine failure rate. Comput. Oper. Res. 61, 31–45. https://doi.org/10.1016/j.cor.2015.02.014

Moon, J.-Y., Park, J., 2014. Smart production scheduling with time-dependent and machine-dependent electricity cost by considering distributed energy resources and energy storage. Int. J. Prod. Res. 52, 3922–3939. https://doi.org/10.1080/00207543.2013.860251

Moon, J.-Y., Shin, K., Park, J., 2013. Optimization of production scheduling with time-dependent and machine-dependent electricity cost for industrial energy efficiency. Int. J. Adv. Manuf. Technol. 68, 523–535. https://doi.org/10.1007/s00170-013-4749-8

Moscato, P., 1989. On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms. Caltech Concurr. Comput. Program C3P Rep. 826, 1989.

Mouzon, G., Yildirim, M.B., 2008. A framework to minimise total energy consumption and total tardiness on a single machine. Int. J. Sustain. Eng. 1, 105–116. https://doi.org/10.1080/19397030802257236

Mouzon, G., Yildirim, M.B., Twomey, J., 2007. Operational methods for minimization of energy consumption of manufacturing equipment. Int. J. Prod. Res. 45, 4247–4271. https://doi.org/10.1080/00207540701450013

Nance, R.E., 1994. The conical methodology and the evolution of simulation model development. Ann. Oper. Res. 53, 1–45.

Nawara, G.M., Hassanein, W.S., 2013. Solving the job-shop scheduling problem by Arena simulation software. Int. J. Eng. Innov. Res. 2, 161.

Nolde, K., Morari, M., 2010. Electrical load tracking scheduling of a steel plant. Comput. Chem. Eng. 34, 1899–1903. https://doi.org/10.1016/j.compchemeng.2010.01.011

Nowicki, E., Smutnicki, C., 2005. An advanced tabu search algorithm for the job shop problem. J. Sched. 8, 145–159.

Nowicki, E., Smutnicki, C., 1996. A Fast Taboo Search for the Job Shop Problem. Manag. Sci. 42, 797–813.

O'Rielly, K., Jeswiet, J., 2014. Strategies to Improve Industrial Energy Efficiency. Procedia CIRP 15, 325–330. https://doi.org/10.1016/j.procir.2014.06.074

Ouelhadj, D., Petrovic, S., 2009. A survey of dynamic scheduling in manufacturing systems. J. Sched. 12, 417–431. https://doi.org/10.1007/s10951-008-0090-8

Pach, C., Berger, T., Sallez, Y., Bonte, T., Adam, E., Trentesaux, D., 2014. Reactive and energy-aware scheduling of flexible manufacturing systems using potential fields. Comput. Ind. 65, 434–448. https://doi.org/10.1016/j.compind.2013.11.008

Palacios, J.J., González, M.A., Vela, C.R., González-Rodríguez, I., Puente, J., 2015. Genetic tabu search for the fuzzy flexible job shop problem. Comput. Oper. Res. 54, 74–89. https://doi.org/10.1016/j.cor.2014.08.023

Papadimitriou, C.H., Steiglitz, K., 1998. Combinatorial Optimization : Algorithms and Complexity. Courier Corporation.

Pegden, C.D., 1983. Introduction To SIMAN. Presented at the Winter Simulation Conference, pp. 231–241.

Pinedo, M.L., 2012. Scheduling. Springer US, Boston, MA. https://doi.org/10.1007/978-1-4614-2361-4

Piroozfard, H., Wong, K.Y., Wong, W.P., n.d. Minimizing total carbon footprint and total late work criterion in flexible job shop scheduling by using an improved multi-objective genetic algorithm. Resour. Conserv. Recycl. https://doi.org/10.1016/j.resconrec.2016.12.001

Prins, C., 2009. A GRASP x evolutionary local search hybrid for the vehicle routing problem, in: Bio-Inspired Algorithms for the Vehicle Routing Problem. Springer, pp. 35–53.

Pritsker, A.A.B., O'Rielly, J.J., Laval, D.K., 1997. Simulation with Visual SLAM and AweSim. John Wiley & Sons, New York.

Qing-dao-er-ji, R., Wang, Y., 2012. A new hybrid genetic algorithm for job shop scheduling problem. Comput. Oper. Res. 39, 2291–2299. https://doi.org/10.1016/j.cor.2011.12.005

Rager, M., Gahm, C., Denz, F., 2015. Energy-oriented scheduling based on Evolutionary Algorithms. Comput. Oper. Res. 54, 218–231. https://doi.org/10.1016/j.cor.2014.05.002

Rahmani, D., Ramezanian, R., 2016. A stable reactive approach in dynamic flexible flow shop scheduling with unexpected disruptions: A case study. Comput. Ind. Eng. 98, 360–372. https://doi.org/10.1016/j.cie.2016.06.018

Rajkumar, M., Asokan, P., Anilkumar, N., Page, T., 2011. A GRASP algorithm for flexible job-shop scheduling problem with limited resource constraints. Int. J. Prod. Res. 49, 2409–2423. https://doi.org/10.1080/00207541003709544

Roy, B., Sussmann, B., 1964. Les problemes d'ordonnancement avec contraintes disjonctives. SEMA, Rapport de recherche n°9.

Salido, M.A., Escamilla, J., Barber, F., Giret, A., Tang, D., Dai, M., 2016a. Energy efficiency, robustness, and makespan optimality in job-shop scheduling problems. Artif. Intell. Eng. Des. Anal. Manuf. 30, 300–312. https://doi.org/10.1017/S0890060415000335

Salido, M.A., Escamilla, J., Giret, A., Barber, F., 2016b. A genetic algorithm for energy-efficiency in job-shop scheduling. Int. J. Adv. Manuf. Technol. 85, 1303–1314. https://doi.org/10.1007/s00170-015-7987-0

Sampels, M., Blum, C., Mastrolilli, M., Rossi-Doria, O., 2002. Metaheuristics for group shop scheduling, in: International Conference on Parallel Problem Solving from Nature. Springer, pp. 631–640.

Sarramia, D., 2002. ASCI-mi: une méthodologie de modélisation multiple et incrémentielle. Application aux systèmes de trafic urbain. Université Blaise Pascal-Clermont-Ferrand II.

Sevaux, M., Sörensen, K., 2004. A genetic algorithm for robust schedules in a one-machine environment with ready times and due dates. Q. J. Belg. Fr. Ital. Oper. Res. Soc. 2. https://doi.org/10.1007/s10288-003-0028-0

Sharma, A., Zhao, F., Sutherland, J.W., 2015. Econological scheduling of a manufacturing enterprise operating under a time-of-use electricity tariff. J. Clean. Prod. 108, 256–270. https://doi.org/10.1016/j.jclepro.2015.06.002

Sharma, P., Jain, A., 2014. Analysis of dispatching rules in a stochastic dynamic job shop manufacturing system with sequence-dependent setup times. Front. Mech. Eng. 9, 380–389. https://doi.org/10.1007/s11465-014-0315-9

Shewhart, W.A., 1939. Statistical Method from the Viewpoint of Quality Control. Courier Corporation, New York.

Shrouf, F., Ordieres-Meré, J., García-Sánchez, A., Ortega-Mier, M., 2014. Optimizing the production scheduling of a single machine to minimize total energy consumption costs. J. Clean. Prod. 67, 197–207. https://doi.org/10.1016/j.jclepro.2013.12.024

Sörensen, K., 2012. Metaheuristics—the metaphor exposed. Int. Trans. Oper. Res. 22, 3–18.

Sprecher, A., 1994. Resource-Constrained Project Scheduling, Lecture Notes in Economics and Mathematical Systems. Springer Berlin Heidelberg, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-48397-4

Stock, T., Seliger, G., 2015. Multi-objective Shop Floor Scheduling Using Monitored Energy Data. Procedia CIRP 26, 510–515. https://doi.org/10.1016/j.procir.2014.07.178

Taguchi, G., 1987. System of experimental design: engineering methods to optimize quality and minimize costs.

Tan, K.C., Goh, C.K., Yang, Y.J., Lee, T.H., 2006. Evolving better population distribution and exploration in evolutionary multi-objective optimization. Eur. J. Oper. Res. 171, 463–495. https://doi.org/10.1016/j.ejor.2004.08.038

Tang, D., Dai, M., 2015. Energy-efficient approach to minimizing the energy consumption in an extended job-shop scheduling problem. Chin. J. Mech. Eng. 28, 1048–1055. https://doi.org/10.3901/CJME.2015.0617.082

Tang, D., Dai, M., Salido, M.A., Giret, A., 2016. Energy-efficient dynamic scheduling for a flexible flow shop using an improved particle swarm optimization. Comput. Ind. 81, 82–95. https://doi.org/10.1016/j.compind.2015.10.001

Tardieu, H., Rochfeld, A., Colletti, R., 1994. La méthode Merise: Principes et outils. Les éditions d'Organisation.

Tavakkoli-Moghaddam, R., Jolai, F., Vaziri, F., Ahmed, P.K., Azaron, A., 2005. A hybrid method for solving stochastic job shop scheduling problems. Appl. Math. Comput. 170, 185–206. https://doi.org/10.1016/j.amc.2004.11.036

Tchernev, N., 1997. Modélisation du processus logistique dans les systèmes flexibles de production. Université de Clermont-Ferrand II.

Tonelli, F., Bruzzone, A.A.G., Paolucci, M., Carpanzano, E., Nicolò, G., Giret, A., Salido, M.A., Trentesaux, D., 2016. Assessment of mathematical programming and agent-based modelling for off-line scheduling: Application to energy aware manufacturing. CIRP Ann. - Manuf. Technol. 65, 405–408. https://doi.org/10.1016/j.cirp.2016.04.119

Toussaint, H., 2010. Algorithmique rapide pour les problèmes de tournées et d'ordonnancement. Université Blaise Pascal-Clermont-Ferrand II.

Tsai, J.-T., Liu, T.-K., Ho, W.-H., Chou, J.-H., 2008. An improved genetic algorithm for job-shop scheduling problems using Taguchi-based crossover. Int. J. Adv. Manuf. Technol. 38, 987–994. https://doi.org/10.1007/s00170-007-1142-5

Turing, A.M., 1937. On computable numbers, with an application to the Entscheidungsproblem. Proc. Lond. Math. Soc. 2, 230–265.

Van Laarhoven, P.J.M., Aarts, E.H.L., Lenstra, J.K., 1992. Job Shop Scheduling by Simulated Annealing. Oper. Res. 40, 113–125.

Van Looveren, A., Gelders, L., Van Wassenhove, L., 1986. A review of FMS planning models, in: Modelling and Design of Flexible Manufacturing Systems. Elsevier Science, Amsterdam.

Wang, S., Liu, M., Chu, F., Chu, C., 2016. Bi-objective optimization of a single machine batch scheduling problem with energy cost consideration. J. Clean. Prod. 137, 1205–1215. https://doi.org/10.1016/j.jclepro.2016.07.206

Weinert, N., Chiotellis, S., Seliger, G., 2011. Methodology for planning and operating energy-efficient production systems. CIRP Ann. - Manuf. Technol. 60, 41–44. https://doi.org/10.1016/j.cirp.2011.03.015

Wolf, S., Merz, P., 2007. Evolutionary Local Search for the Super-Peer Selection Problem and the p-Hub Median Problem. Lect. Notes Comput. Sci., Lecture notes in computer science 4771, 1–15.

Wolpert, D.H., Macready, W.G., 1997. No free lunch theorems for optimization. IEEE Trans. Evol. Comput. 1, 67–82.

Wright, S.J., 2015. Coordinate descent algorithms. Math. Program. 151, 3–34. https://doi.org/10.1007/s10107-015-0892-3

Yin, L., Li, X., Gao, L., Lu, C., Zhang, Z., 2016. A novel mathematical model and multi-objective method for the low-carbon flexible job shop scheduling problem. Sustain. Comput. Inform. Syst. https://doi.org/10.1016/j.suscom.2016.11.002

Yuan, Y., Xu, H., 2013a. Flexible job shop scheduling using hybrid differential evolution algorithms. Comput. Ind. Eng. 65, 246–260. https://doi.org/10.1016/j.cie.2013.02.022

Yuan, Y., Xu, H., 2013b. An integrated search heuristic for large-scale flexible job shop scheduling problems. Comput. Oper. Res. 40, 2864–2877. https://doi.org/10.1016/j.cor.2013.06.010

Yuan, Y., Xu, H., Yang, J., 2013. A hybrid harmony search algorithm for the flexible job shop scheduling problem. Appl. Soft Comput. 13, 3259–3272. https://doi.org/10.1016/j.asoc.2013.02.013

Zdamar, L., Ulusoy, G., 1995. A survey on the resource-constrained project scheduling problem. IIE Trans. 27, 574–586. https://doi.org/10.1080/07408179508936773

Zhang, C., Li, P., Guan, Z., Rao, Y., 2007. A tabu search algorithm with a new neighborhood structure for the job shop scheduling problem. Comput. Oper. Res. 34, 3229–3242. https://doi.org/10.1016/j.cor.2005.12.002

Zhang, C.Y., Li, P., Rao, Y., Guan, Z., 2008. A very fast TS/SA algorithm for the job shop scheduling problem. Comput. Oper. Res. 35, 282–294. https://doi.org/10.1016/j.cor.2006.02.024

Zhang, H., Zhao, F., Fang, K., Sutherland, J.W., 2014. Energy-conscious flow shop scheduling under time-of-use electricity tariffs. CIRP Ann. - Manuf. Technol. 63, 37–40. https://doi.org/10.1016/j.cirp.2014.03.011

Zhang, H., Zhao, F., Sutherland, J.W., 2015. Energy-efficient scheduling of multiple manufacturing factories under real-time electricity pricing. CIRP Ann. - Manuf. Technol. 64, 41–44. https://doi.org/10.1016/j.cirp.2015.04.049

Zhang, L., Li, X., Gao, L., Zhang, G., 2016. Dynamic rescheduling in FMS that is simultaneously considering energy consumption and schedule efficiency. Int. J. Adv. Manuf. Technol. 87, 1387–1399. https://doi.org/10.1007/s00170-013-4867-3

Zhang, L., Luo, Y., Zhang, Y., Song, G., 2015. Production Scheduling Oriented to Energy Consumption Optimization for Process Industry Based on Self-adaptive DE Algorithm. Int. J. Control Autom. 8, 31–42. https://doi.org/10.14257/ijca.2015.8.2.04

Zhang, R., Chiong, R., 2016. Solving the energy-efficient job shop scheduling problem: a multi-objective genetic algorithm with enhanced local search for minimizing the total weighted tardiness and total energy consumption. J. Clean. Prod. 112, 3361–3375. https://doi.org/10.1016/j.jclepro.2015.09.097

Zhang, R., Wu, C., 2011. An Artificial Bee Colony Algorithm for the Job Shop Scheduling Problem with Random Processing Times. Entropy 13, 1708–1729. https://doi.org/10.3390/e13091708

Zhou, A., Qu, B.-Y., Li, H., Zhao, S.-Z., Suganthan, P.N., Zhang, Q., 2011. Multiobjective evolutionary algorithms: A survey of the state of the art. Swarm Evol. Comput. 1, 32–49. https://doi.org/10.1016/j.swevo.2011.03.001

Zitzler, E., Laumanns, M., Thiele, L., 2001. SPEA2: Improving the strength Pareto evolutionary algorithm.

Annexes

Annexe 1 : Formalisation mathématique pour le Bi-JSPPR

Notations

H : un nombre entier très grand ;

M : ensemble de machines ;

J : ensemble de jobs ;

V : ensemble de toutes les opérations ($|V| = |M|.|J|$) ;

i, j : indices pour les différentes opérations ($i = O_i$) ;

J_i : job de l'opération i ;

SO_i : ensemble des sous-opérations d'une opération i ;

k, l : indices représentant les différentes sous-opérations d'une opération ;

mi : machine requise pour la réalisation de l'opération i, $m_i \in M$;

$P_{i,k}$: durée de la $k^{ième}$ sous-opération de l'opération i ;

$W_{i,k}$: puissance requise pour réaliser la $k^{ième}$ sous-opération de l'opération i ;

w_{max} : seuil de puissance maximal à ne pas dépasser (objectif) ;

c_{max} : temps total de traitement de toutes les opérations (objectif) ;

$s_{i,k}$: date de début de la $k^{ième}$ sous-opération de l'opération i ;

$x_{i,k,j,l}$: variable binaire égale à 1 si la $k^{ième}$ sous-opération de l'opération i est réalisée avant la $l^{ième}$ sous-opération de l'opération j, et égale à 0 sinon ;

$y_{i,k,j,l}$: variable binaire égale à 1 s'il y a un flot de puissance non nul allant de l'opération i vers l'opération j, et égale à 0 sinon ;

$\varphi_{i,k,j,l}$: variable entière représentant le nombre d'unités de puissance allant de la $k^{ième}$ sous-opération de l'opération i vers la $l^{ième}$ sous-opération de l'opération j. $\varphi_{0,0,j,l}$ correspond aux unités de puissance allant de l'origine (nœud O) aux autres opérations ;

Formalisation mathématique

La première équation (1) de la formalisation mathématique correspond à l'objectif du problème : la minimisation du temps total de traitement (makespan) et de la puissance autorisée (seuil de puissance).

$$Min\ c_{max}$$
$$Min\ w_{max} \quad (1)$$

Modèle pour le Job-shop

$$s_{i,|SO_i|} - c_{max} \leq -P_{i,|SO_i|}, \forall i \in V \quad (2)$$

$$x_{i,|SO_i|,j,1} + x_{j,|SO_j|,i,1} = 1, \forall (i,j) \in V, m_i = m_j \quad (3)$$

$$s_{j,1} - s_{i,|SO_i|} \geq P_{i,|SO_i|}, \forall (i,j) \in V, i < j, J_i = J_j \quad (4)$$

$$s_{i,k} - s_{i,k-1} = P_{i,k-1}, \forall i \in V, \forall k \in SO_i, k > 1 \quad (5)$$

$$s_{j,1} - s_{i,|SO_i|} - Hx_{i,|SO_i|,j,1} \geq P_{i,|SO_i|} - H, \forall (i,j) \in V, m_i = m_j \quad (6)$$

Le premier ensemble de contraintes (2) donne l'expression du makespan, qui doit être plus grand ou égal à la date de fin de toutes les opérations (comme elles sont découpées en sous-opérations, le calcul est effectué à partir de la dernière sous-opération, d'où l'utilisation du cardinal de l'ensemble SO_i). Les contraintes (3) représentent les disjonctions entre les opérations traitées par la même machine. Dans ces contraintes, si deux opérations i et j appartenant à différents jobs doivent être ordonnancées sur la même machine, alors la dernière sous-opération de l'opération i est traitée avant la première sous-opération de l'opération j, ou vice versa. Les contraintes (4) définissent les dates de début des opérations en fonction des opérations précédentes dans la gamme opératoire du job. Les contraintes (5) assurent la non-préemption des opérations. Les contraintes (6) définissent les dates de début des opérations qui doivent être traitées par la même machine, car elles ne peuvent être effectuées simultanément.

Modèle pour la puissance consommée

$$\sum_{j \in V} \sum_{k \in SO_j} \varphi_{0,0,j,k} - w_{max} \leq 0 \quad (7)$$

$$\varphi_{0,0,j,l} + \sum_{i \in V \setminus j} \sum_{k \in SO_i} \varphi_{i,k,j,l} + \sum_{k=1}^{l-1} \varphi_{j,k,j,l} = W_{j,l}, \forall j \in V, \forall l \in SO_j \quad (8)$$

$$\sum_{j \in V \setminus i} \sum_{l \in SO_j} \varphi_{i,k,j,l} + \sum_{l=k+1}^{|SO_i|} \varphi_{i,k,i,l} \leq W_{i,k}, \forall i \in V, \forall k \in SO_i \quad (9)$$

$$\varphi_{i,k,j,l} - Hy_{i,k,j,l} \leq 0, \forall (i,j) \in V, \forall k \in SO_i, \forall l \in SO_j \quad (10)$$

$$y_{i,k,j,l} - \varphi_{i,k,j,l} \leq 0, \forall (i,j) \in V, \forall k \in SO_i, \forall l \in SO_j \quad (11)$$

$$s_{j,l} - s_{i,k} - Hy_{i,k,j,l} \geq P_{i,k} - H, \forall (i,j) \in V, \forall (k,l) \in \left(SO_i, SO_j\right), J_i \neq J_j \quad (12)$$

$$\varphi_{i,k,j,l} = 0, \forall (i,j) \in V, \forall (k,l) \in (SO_i, SO_j), (j < i, J_i = J_j) \vee (i = j, l < k) \quad (13)$$

La contrainte (7) empêche de dépasser le seuil de puissance autorisé. Les contraintes (8) assurent que la somme des flots allant de l'origine et de chaque sous-opération vers la $l^{ième}$ sous-opération de l'opération j correspond aux besoins en puissance de cette sous-opération. Les contraintes (9) assurent que la somme des flots sortant de la $k^{ième}$ sous-opération de l'opération i n'excède pas la puissance consommé par cette sous-opération. Les contraintes (10) fixent $y_{i,k,j,l}$ à 1 si un flot existe entre la $k^{ième}$ sous-opération de l'opération i vers la $l^{ième}$ sous-opération de l'opération j. Si $y_{i,k,j,l} = 0$ alors aucun flot n'est possible entre les deux sous-opérations. Les contraintes (11) fixent $y_{i,k,j,l}$ à 0 si aucun flot n'existe entre les sous-opérations ; si $y_{i,k,j,l} = 1$ alors un flot de valeur positive existe entre les sous-opérations ($\varphi_{i,k,j,l} \geq 1$). Les contraintes (12) ajustent les dates de début des sous-opérations qui doivent attendre la fin d'autres opérations afin de ne pas dépasser le seuil de puissance autorisé. Les contraintes (13) assurent que si $O_{j,l}$ et $O_{j,l}$ sont deux opérations d'un même job, alors aucun flot ne peut exister entre ces sous-opérations si $O_{j,l}$ est réalisée après $O_{i,k}$.