Université Abdelhamid Ibn Badis Mostaganem
Faculté des Sciences Exactes et de l'Informatique
Département d'Informatique

Thèse de Doctorat

Présentée par :

Mohamed HABIB ZAHMANI

Contribution à l'ordonnancement dynamique : proposition d'une approche guidée par effet de Simulation/Datamining

Spécialité : Informatique

Option : Apprentissage Automatique et Web Intelligence

Soutenue le 06 Mars 2018 devant un jury composé de :

Président du jury	**Pr. Omar BELHAMITI**	(Université de Mostaganem)
Examinateurs	**Dr. Karim SEHABA**	(Université de Mostaganem)
	Pr. Zakaria ELBERRICHI	(Université de Sidi-Bel-Abbès)
Directeur de thèse	**Pr. Baghdad ATMANI**	(Université d'Oran 1)
Invité	**Pr. Ghalem BELALEM**	(Université d'Oran 1)

Résumé

Dans un environnement ouvert et mondialisé où la satisfaction des attentes des clients est à la base d'une concurrence extrême, l'amélioration de la compétitivité des entreprises devient une question de survie. Cette compétitivité implique la nécessité d'améliorer continuellement la productivité et de réduire encore et toujours les coûts de production. Parmi les réponses les plus immédiates on trouve l'ordonnancement dans les ateliers de fabrication.

L'ordonnancement, étape de planification et d'organisation du fonctionnement des ateliers de production, optimise l'affection des ressources humaines et matérielles aux tâches et opérations de production. Il améliore l'utilisation des ressources et réduit de ce fait les temps de production et diminue les coûts de production.

Pour les besoins de cette optimisation, il est possible de faire appel à une panoplie de méthodes et techniques présentées dans la littérature, chacune ayant ses points forts et ses points faibles et pouvant s'adapter à tel ou tel problème d'ordonnancement en fonction de ses particularités. Parmi les techniques qui ont fait leurs preuves pour la résolution de problèmes d'ordonnancement on peut citer les règles de priorité. Les règles de priorité sont capables d'affecter des priorités aux tâches en temps-réel, permettant ainsi de résoudre des problèmes d'ordonnancement fortement complexes mais aussi de renforcer la réactivité du système en cas de perturbation. Cependant, elles présentent certaines faiblesses liées au fait qu'aucune règle de priorité n'est à priori meilleure que les autres dans des environnements de production différents, poussant ainsi les opérateurs humains à recourir à la simulation, à leur expérience, ou aux données issues de productions antérieures.

En pratique, les systèmes de production génèrent de gros volumes de données, qui ne sont souvent pas exploitées en dépit de leur fort potentiel et l'hypothèse est de mettre à profit ces données et d'y découvrir des connaissances implicites afin d'améliorer les pratiques d'ordonnancement actuelles. La solution serait de recourir à la fouille des données pour l'extraction de cette connaissance.

Dans cette thèse nous proposons une nouvelle approche combinant simulation et fouille de données pour automatiser la sélection et l'identification des règles de priorité, et leur affectation, en temps-réel, aux machines d'un atelier Job Shop en vue d'en optimiser l'ordonnancement. Cette méthodologie repose sur l'hypothèse selon laquelle l'allocation de règles de priorité différentes pour chacune des machines de l'atelier permet de trouver de meilleures solutions.

Mise à part l'automatisation de la sélection des règles de priorité, la fouille de données peut aussi être utilisée pour l'extraction de nouvelles règles en s'inspirant d'autres méthodes de résolution plus efficaces. Dans cette perspective nous proposons l'utilisation de la fouille de données pour la génération de nouvelles règles de priorité à partir de solutions obtenues par un algorithme génétique. Nous nous focalisons aussi sur l'utilisation de ces nouvelles règles dans les problèmes d'ordonnancement Single Machine.

Dans chacune des approches proposées dans cette thèse, des expérimentations intensives ont été menées afin de montrer l'efficacité et l'intérêt à intégrer la fouille de données pour l'amélioration et l'optimisation des problèmes d'ordonnancement, leur résolution en temps-réel, ainsi que son utilisation comme outil d'aide à la décision dans la sélection des règles de priorité.

Mots-clés : Systèmes de Production · Ordonnancement en Temps-Réel · Optimisation · Simulation · Règles de Priorité · Fouille de Données · Aide à la Décision · Extraction de Connaissances · Arbres de Décision

مُلَخَّصٌ

في البيئة المفتوحة والعالمية الحالية أين إرضاء الزبائن هو أساس لتنافس شرس، أصبح تحسين قدرات الشركات التنافسية مسألة ضمان للبقاء في هذه البيئة. وترجع هذه القدرة التنافسية إلى ضرورة مواصلة تحسين القدرة الإنتاجية و كذلك تخفيض تكاليفها. من بين الحلول الممكنة في هذا المجال الجدولة في ورشات التصنيع.

إن الجدولة، وهي مرحلة تخطيط وتنظيم في ورشات التصنيع، تحسن عملية تخصيص الموارد البشرية والمادية للاشغال و العمليات. مما يؤدي إلى تثمين استخدام الموارد، وبالتالي تقليل أوقات الإنتاج و تكاليفها.

لحل هذه المشاكل، نجد مجموعة واسعة من الأساليب والتقنيات المقترحة من قبل الباحثين في هذا المجال، ولكل منها نقاط قوة وضعف خاصة بها، حيث من الممكن تكييفها لجدولة الأعمال لحالات معينة وفقا للخصائص التي تميزها عن غيرها. ومن بين التقنيات التي أثبتت جدارتها في حل مشاكل الجدولة نجد قواعد الأولوية. إن قواعد الأولوية قادرة على تحديد الأولويات للإشغال في الوقت الأني، مما يسمح لحل مشاكل جدولة صعبة وتعزيز استجابية النظام في حالة حدوث اضطرابات. ومع ذلك، هذه التقنيات لديها بعض نقاط الضعف التي ترجع إلى عدم وجود أي قاعدة أولوية أفضل من غيرها في بيئات إنتاج مختلفة، مما يدفع المسؤولين للجوء إلى تقنية المحاكاة، أو إلى خبراتهم المكتسبة، أو إلى البيانات المجمعة من عمليات الإنتاج السابقة.

ومن الناحية العملية، تولد نظم الإنتاج كميات كبيرة من البيانات، التي غالبا لا تستغل رغم فائدتها المحتملة والفرص التي تتيحها لاكتشاف معارف مفيدة من هذه البيانات لغرض أجل تحسين عمليات الجدولة الحالية. الحل سيكون باستخدام تقنيات استخراج البيانات للتحصل على هذه المعرفة.

في هذه الأطروحة، نقترح منهجية جديدة تجمع بين المحاكاة و تقنيات إستخراج البيانات بهدف التشغيل الآلي لعمليات الاختيار، التحديد، و التخصيص في الوقت الأني لقواعد الأولوية لكل الآلات ضمن ورشة العمل متعددة التوجيهات. تستند هذه المنهجية على افتراض أن تخصيص قاعدة أولوية مختلفة لكل آلة في ورشة العمل يسمح بإيجاد حلول جيدة.

وبصرف النظر عن الإختيار الآلي لقواعد الأولوية، من الممكن أيضا إستعمال تقنيات إستخراج البيانات للحصول على قواعد جديدة من خلال تقليد أساليب حل أخرى أكثر كفاءة. وبالتالي نقترح استخدام تقنيات إستخراج البيانات لتشكيل قواعد أولوية جديدة من خلال الحلول المتحصل عليها اعتمادا على خوارزمية وراثية. و نركز كذلك على كيفية إستخدام هذه القواعد الجديدة من أجل حل مشاكل جدولة ذات آلة واحدة.

في كل المناهج المقترحة في هاته الأطروحة، أجريت عدة تجارب واسعة النطاق لإثبات فعالية وفائدة هاته المناهج، وللتأكيد على ضرورة دمج تقنيات استخراج البيانات لتحسين حلول مشاكل جدولة الأعمال، حلها في الوقت الأني، وكذلك كأداة لدعم القرارات عند إختيار قواعد الأولوية.

الكلمات المفتاحية : أنظمة الإنتاج · جدولة الأعمال في الوقت الأني · تحسين · محاكاة · قواعد الأولوية · استخراج البيانات · إتخاذ القرار · استخراج المعرفة · شجرة القرار

Abstract

In an open and globalized environment where customer satisfaction is at the root of extreme competition, the competitiveness of companies becomes a matter of survival. This competitiveness implies the need to continuously improve productivity and to reduce production costs. Among the most immediate answers is scheduling in manufacturing workshops.

Scheduling, a stage in planning and organizing the operation of workshops, optimizes the allocation of human and material resources to jobs and operations, improving the use of resources and thereby reducing production times as well as costs.

To solve this optimization problem, a wide variety of well-established methods and techniques presented in literature is available, each with its own strengths and weaknesses, all of which can be adapted to a specific scheduling problem and its peculiarities. Examples of proven techniques for solving scheduling problems include dispatching rules; Dispatching rules are capable of assigning priorities to jobs in real time, thus allowing to solve highly complex scheduling problems and enhancing the system's responsiveness in the event of a disruption. However, they have certain weaknesses related to the fact that no dispatching rule is a priori better than the others in different manufacturing environments, forcing human operators to resort to simulation, their experience, or data from earlier productions.

In practice, manufacturing systems generate large volumes of data, which are often not exploited despite their potential usefulness and a good chance of discovering useful knowledge from this data to improve the current scheduling practices. The solution would be to use data mining to extract this knowledge.

In this thesis, we propose a new methodology combining simulation and data mining for the automation of the selection, identification, and real-time allocation of dispatching rules to the machines of a Job Shop to optimize the scheduling performance measures. This methodology is based on the assumption that the allocation of a different dispatching rule for each machine in the workshop allows to reach better solutions.

Aside from automating the selection of dispatching rules, data mining can also be used to extract new rules by mimicking other, more efficient solving methods. Given this we propose the use of data mining for the generation of new dispatching rules from solutions obtained by a genetic algorithm. We also focus on the use these new rules in order to solve Single Machine scheduling problems.

For each approach presented below, intensive experiments were conducted to highlight their efficiency and emphasize, should data mining be integrated in the process, the improvements made in the optimization of scheduling problems, their resolution in real time, as well as a decision-making tool for the selection of dispatching rules.

Keywords: Manufacturing Systems · Scheduling in Real-Time · Optimization · Simulation · Dispatching Rules · Data Mining · Decision Support · Knowledge Extraction · Decision Trees

Remerciements

La rédaction de cette partie n'est pas une chose facile. En pensant aux personnes qui ont contribué d'une manière ou d'une autre à l'élaboration de ce travail de thèse, de nombreux noms me viennent à l'esprit. Seulement, je n'arrive pas à trouver les bons mots pour exprimer ma gratitude et ma reconnaissance envers eux. Je m'excuse auprès des personnes qui j'ai oublié de mentionner, merci pour tout ce que vous avez fait pour moi.

Tout d'abord j'exprime ma gratitude à mon directeur de thèse, Pr. Baghdad ATMANI, pour son encadrement continu, son suivi, ses orientations, et ses efforts pour l'aboutissement de ce travail. Je le remercie également pour la confiance qu'il m'a toujours témoigné, mais aussi ses précieux conseils et pas uniquement sur le plan professionnel. Ma gratitude pour sa qualité humaine, son comportement envers moi, son hospitalité, et son soutien tout au long de ces cinq dernières années.

J'adresse mes remerciements à Messieurs Omar BELHAMITI, Karim SEHABA, Zakaria ELBERRICHI, ainsi que Ghalem BELALEM de m'avoir fait l'honneur d'examiner ce travail de thèse et d'avoir accepté de faire partie des membres du jury. J'apprécie l'intérêt qu'ils ont porté à mes travaux.

Je tiens également à remercier le staff de l'Université de Mostaganem. Je pense notamment au Pr. Omar BELHAMITI pour la possibilité qu'il m'a offerte pour continuer mes études et m'améliorer, mais aussi pour ses précieux conseils, son soutien, sa disponibilité. Je remercie également Dr. Fouad HENNI pour son soutien et ses orientations, il a toujours fait de son mieux pour me faciliter mon travail. Au Prof. Houari BENMEKKI qui a mis tous les moyens à sa disposition pour me faciliter et m'aider à réaliser mon travail de thèse en m'offrant la possibilité de réaliser une partie de ma thèse à l'Université de Valenciennes, mais aussi pour m'avoir permis de participer aux conférences internationales pour exposer et discuter mon travail.

Je remercie également les membres du laboratoire LIO de l'Université d'Oran 1 que j'ai eu l'honneur de côtoyer pour leur amitié et leurs mots de soutien. Je pense notamment au Pr. Ghalem BELALEM et le Pr. Bouziane BELDJILALI. Merci à Sidahmed, Amine, Laïd, Hichem, Fatima, Wassim, et Yasmine.

Mes remerciements vont également au Dr. Abdelghani BEKRAR membre du laboratoire LAMIH de l'Université de Valenciennes et du Hainaut-Cambrésis

pour m'avoir accompagné lors de mes stages au sein de son laboratoire de recherche. Pour ses précieux conseils et orientations qui m'ont toujours permis de me remettre sur le droit chemin. Je pense aussi au Dr. Nassima AISSANI de l'Université d'Oran 1 d'avoir fait de son mieux pour m'aider à achever mon travail de thèse, en écoutant mes préoccupations, et en m'aidant à résoudre mes problèmes.

Je tiens aussi à exprimer ma gratitude envers les membres du laboratoire LAMIH, notamment le Pr. Damien TRENTESAUX qui était toujours disponible pour évaluer mon travail et de prodiguer de précieux conseils pour l'améliorer. Je remercie également Dr. Farid KADRI et Dr. Sondes CHAABANE du LAMIH pour leurs conseils, orientations, et encouragements. À José pour son soutien et son aide pour l'achèvement de mon travail mais aussi pour son amitié et ses encouragements. À Faïza et Marwa avec lesquels j'ai fait connaissance lors de mes stages, merci à vous pour votre gentillesse et votre aide.

À tous mes précieux amis de l'Université de Mostaganem, en particulier : Amine, Saliha, Asma, Djamila, Amel, Amin, Walid, Djalil, Maghnia, Naïma, Leila, et Houria avec lesquels j'ai partagé des moments inoubliables.

À mes vieux amis et compagnons, Mohamed El Amine, Habib, Yassine, Mohamed, Zaki, Amine, Sofiane, et Daho pour leurs encouragements et soutien, pour leur respect et leur gentillesse. Vous êtes les meilleurs !

Je ne saurais finir sans exprimer mes remerciements aux personnes qui me sont les plus proches et les plus chères. À mes parents pour lesquels je suis éternellement reconnaissant pour m'avoir constamment soutenu, pour avoir cru en moi pour mener à bien ce travail, pour avoir mis à ma disposition tous les moyens dont ils disposaient, mais surtout pour leur amour. Merci infiniment !

À mon frère Samir qui a beaucoup contribué dans cette thèse, pour les nombreuses heures passées à lire, relire, et corriger mes manuscrits. À mes sœurs Soraya et Amina pour leur soutien indéfectible et leurs encouragements. Un remerciement particulier à mon oncle Ali pour son support et son soutien.

Et enfin, merci à tous ceux qui ont participé de près ou de loin à l'aboutissement de ce travail.

Je remercie *Dieu*, le Tout Puissant.

Je dédie ce travail à toutes les personnes qui sont chères à mon cœur,

À mes parents,

À mon frère, mes sœurs, à ma nièce, et ma grand-mère,

À mes amis.

Table des matières

Liste des figures

Liste des tableaux

Introduction Générale

Aujourd'hui, dans le cadre d'une économie ouverte et de la mondialisation, les entreprises doivent faire face à une concurrence accrue, illustrée par une offre plus large et plus diversifiée, à des exigences fixées par le client-roi en termes de qualité, de prix et d'échéances, à des normes environnementales et règlementaires de plus en plus strictes et bien d'autres contraintes. L'entreprise doit faire face en même temps à tous ces challenges afin de se faire une place dans ce marché ultra-compétitif et ainsi garantir sa survie. L'un des points clés de la compétitivité réside dans un effort continuel d'amélioration de la productivité et de réduction encore et toujours des coûts de production. La productivité est sous-tendue par la maitrise des processus de fabrication, elle-même dépendante du degré de maitrise de la planification des opérations de production et de l'utilisation des ressources humaines, techniques et technologiques de l'entreprise.

Les systèmes de production actuels offrent de nombreuses possibilités telle la maximisation de l'utilisation des ressources, la réduction des temps perdus (machines disponibles et tâches en attente), la réduction du temps de réalisation des produits, ou le respect des délais. Ces systèmes de production sont connus sous le nom de « *Systèmes de Production Flexibles* », en anglais « *Flexible Manufacturing Systems* » (*FMS*) [1], [2]. Afin de tirer pleinement profit de ces systèmes mais aussi de faire face à leurs difficultés, les décisions d'affectations des ressources aux produits, le séquencement des opérations, ou l'arrivée dynamique des tâches doivent être pris en considérations et traités en temps-réel lors d'une phase connue sous le terme *d'ordonnancement*.

L'ordonnancement est une tâche critique du processus de planification, où le but est d'allouer, d'une manière intelligente et efficace, les ressources de production disponibles aux tâches ou aux opérations de fabrication tout en trouvant un compromis entre plusieurs objectifs exprimés en termes de temps, de coût, ou de réactivité, et en respectant différentes contraintes liées à ces mêmes ressources ou tâches. Les problèmes d'ordonnancement font partie de la famille des problèmes d'optimisation et ont fait l'objet d'études minutieuses de la part de chercheurs issus de l'informatique, du management, ou de l'ingénierie de production entre autres [3]–[8]. Un nombre très important de contributions ont été proposées où de nombreuses méthodes de résolution sont avancées pour la résolution des problèmes d'ordonnancement. Ces contributions vont des méthodes exactes jusqu'aux heuristiques et méta-heuristiques [4].

L'ordonnancement dans les ateliers à cheminements multiples ou Job Shop fait partie des problèmes d'ordonnancement les plus complexes et qui a reçu beaucoup d'attention de la part de la communauté de recherche. Le Job Shop est un système de fabrication où des produits spécifiques ou personnalisés sont réalisés par petits lots, ou par petites quantités. Chaque lot est différent du précédent et requière d'autres ressources ou suit un ordre de production différent. Un problème Job Shop est généralement composé de plusieurs machines pouvant opérer plusieurs types d'opérations (Job Shop flexible). Il existe de milliers d'usines pouvant réaliser plusieurs types de produits qui peuvent être classés comme étant du type Job Shop d'où l'engouement des chercheurs.

Ainsi donc, en raison de sa complexité, le Job Shop est considéré comme étant un problème NP-Difficile (*NP-Hard : Non-deterministic Polynomial-time Hard*) [9].

Pour résoudre ce type de problèmes, les chercheurs ont d'abord orienté leurs efforts vers les méthodes exactes. Ces méthodes sont basées sur des techniques énumératives et des méthodes mathématiques [10]. C'est le cas notamment de la fameuse méthode Branch & Bound qui reste la méthode exacte la plus efficace pour ce genre de problèmes [11], [12], mais qui, en contrepartie, requière un temps d'exécution très important pour trouver une solution optimale. Ce temps pouvant évoluer exponentiellement en fonction de la taille du problème, l'utilisation du Branch & Bound devient rapidement impossible. Il s'ensuivit une attention accrue de la part des chercheurs aux heuristiques et méta-heuristiques qui garantissent des bons résultats en un temps relativement raisonnable (polynomial) [13]. Parmi ces méthodes on trouve les « Règles de Priorité » qui sont dédiées à la résolution de problèmes d'ordonnancement difficile comme le Job Shop, le Job Shop dynamique et le Job Shop flexible entre-autres. Ici, l'ordonnancement se base sur la gestion des files d'attentes, c'est-à-dire les tâches qui sont en attente d'une certaine ressource, et sur la définition d'une priorité pour chacune de ces tâches. Ce qui permet de maximiser le taux d'utilisation des ressources et de réduire globalement les temps d'attente, le processus entier se déroulant en temps réel. Les règles de priorité ont intéressé de nombreux chercheurs et ont fait l'objet de nombreux travaux de recherches [14]–[20].

Bien que les règles de priorité ne garantissent pas l'optimalité elles sont tout de même très sollicitées en pratique grâce à leur simplicité, leur faible temps de calcul, et le fait qu'elles peuvent être utilisées dans des environnements dynamiques [21]. Une règle attribue une priorité à chaque tâche en attente sur une file d'une machine quelconque, une fois que la machine est disponible la tâche la plus prioritaire est alors sélectionnée pour être opérée. L'affectation des priorités se fait suivant de simples opérations mathématiques ou via la comparaison d'attributs des tâches, par exemple en fonction de la date d'arrivée ou du temps d'exécution requis contrairement aux méta-heuristiques et autres méthodes exactes qui représentent l'espace de solution d'une manière plus sophistiquée [10]. Cette spécificité des règles de priorité permet aux opérateurs travaillants dans le système de production de mieux comprendre son comportement, de déterminer la ou les tâches qui seront traitées une fois qu'une machine devient libre et d'avoir la possibilité d'interagir et de modifier le séquencement si cela s'avère nécessaire.

Un autre aspect positif des règles de priorité réside dans le fait qu'elles peuvent être modifiées lors d'évènements subits non pris en considération durant la phase de planification, comme l'arrivée de nouvelles tâches, le changement de dates d'échéances de certaines tâches, ou encore des pannes machines, et ainsi réagir efficacement [21], [22]. De plus, les règles de priorité gardent leurs performances et se dégradent peu lors de changement d'échelle ou pour des problèmes complexes, contrairement aux méthodes exactes et méta-heuristiques [18], [21], [23]. Elles peuvent aussi être utilisées dans le but d'accélérer la recherche de solutions par méta-heuristiques en générant un ensemble de solutions initiales offrants de bonnes performances [19], ceci permet une convergence rapide de la méthode d'exploration et la possibilité d'atteindre de meilleures solutions.

Cependant, pour la résolution de problèmes d'ordonnancement à base de règles de priorité, il est nécessaire de tester différentes règles en utilisant la simulation afin de déterminer celle qui peut maximiser/minimiser la fonction objectif du problème. Ce processus de sélection peut s'avérer très lent dans certains cas en raison du nombre de règles considérées ou en raison du temps requis par la simulation dans d'autres cas. D'où la principale motivation de cette thèse où l'objectif est d'intégrer les techniques de l'*Intelligence Artificielle* et de *Data Mining* pour la simplification et l'automatisation du processus de sélection des règles de priorité.

Le Data Mining, ou *Fouille de Données*, est l'étape centrale du processus d'*Extraction de Connaissances à partir de Données*. Elle consiste en un ensemble de techniques basées sur les connaissances qualitatives et quantitatives spécifiques à un problème donné. Son but est de rechercher et d'extraire de l'information inconnue ou cachée et qui peut être utile à partir de gros volumes de données. Ces données ont généralement pour origine une source humaine, la simulation, ou des données historiques. Ces sources sont alors traitées, étudiées, et analysées afin de faire sortir de la connaissance ou des informations qui ne peuvent être obtenues sans une profonde étude statistique et numérique. La fouille de données s'est ainsi imposée comme un outil incontournable pour l'ordonnancement et le réordonnancement en permettant la capture, l'analyse de données issues de ces problèmes, et leur transformation en modèles pour l'assistance, la prise de décision, et la simplification de la résolution des problèmes d'ordonnancement [20], [22], [24].

Dans le cas de notre thèse et dans le but de résoudre le problème de sélection des règles de priorité, nous proposons l'intégration de la fouille de données afin de déduire la ou les meilleures règles de priorité pouvant répondre d'une manière efficace au problème d'ordonnancement sans pour autant avoir recours à la simulation utilisée jusqu'à présent pour la sélection [19], [25], [26]. Le Data Mining permet alors d'identifier la meilleure règle de priorité en temps-réel et ainsi offre la possibilité de réagir, entre-autres, aux perturbations. L'approche que nous proposons se base sur le fait qu'un ensemble de règles de priorité est déjà prédéfini et où le rôle de la fouille de données est de déterminer la meilleure à appliquer en fonction de l'état courant de l'atelier de production.

Mise à part l'aide à la sélection des règles de priorité, la fouille de données peut être aussi utilisée pour générer de nouvelles règles de priorité. Dans cette perspective, nous proposons d'intégrer la fouille de données dans le processus d'extraction des règles. Pour contribuer dans ce domaine nous avons proposé d'utiliser conjointement la fouille de données et l'optimisation pour résoudre efficacement des problèmes d'ordonnancement Single Machine, ou « *atelier à Machine Unique* » considéré aussi NP-Difficile [27]. Les résultats obtenus ont montré l'efficacité des règles extraites comparativement aux règles de priorité issues de la littérature et bien connues pour leurs bonnes performances pour ce genre de problèmes. Le choix porté sur l'atelier Single Machine est motivé par sa complexité moindre que celle du Job Shop, mais dans le but, surtout, de démontrer la capacité de la fouille de données à générer des règles de priorité.

Organisation de la Thèse

Cette thèse est composée de six chapitres organisés selon le plan suivant :

Dans le premier chapitre, nous introduisons les problèmes d'ordonnancement, leurs concepts de base, les éléments clés qui les composent et les caractérisent ainsi que les notations utilisées pour formuler les problèmes. Nous présentons ensuite les différentes classes d'ordonnancement. S'en suit les différents problèmes d'ordonnancement rencontrés dans la littérature avec un état de l'art comprenant plusieurs travaux de recherche ayant porté sur les problèmes Job Shop et Single Machine. Nous nous intéressons ensuite aux notions générales sur la complexité des problèmes d'ordonnancement. Nous détaillons aussi les différentes méthodes de modélisation et de représentation graphiques et mathématiques. Enfin, nous terminons le chapitre avec la présentation des méthodes de résolution, à savoir les méthodes exactes et les méthodes approchées.

Lors du deuxième chapitre, nous nous intéressons aux règles de priorité dont nous donnons une liste exhaustive, à leur classification ainsi qu'aux facteurs affectant leurs performances. Nous présentons ensuite un état de l'art sur les règles de priorité qui est organisé en trois parties en fonction de l'objectif des différents travaux de recherche discutés. En raison du grand nombre de règles de priorité existantes, il est nécessaire de mettre en place un système d'aide à la sélection des meilleures règles.

Dans le troisième chapitre, nous abordons la fouille de données et son application dans l'ordonnancement par règles de priorité. Nous commençons par présenter brièvement le processus d'extraction de connaissances à partir de données ainsi que ses principales composantes. Ensuite, nous détaillons la fouille de données, ses techniques pour l'extraction de connaissances ainsi que ses différents types d'utilisation. Après nous présentons un état de l'art sur l'emploi de la fouille de données dans les problèmes d'ordonnancement et son apport en fonction des objectifs désirés, à savoir, la sélection ou l'extraction de règles de priorité. Enfin, nous détaillons la procédure complète de création et de construction des arbres de décision à l'aide des algorithmes ID3 et C4.5.

Lors du quatrième chapitre, nous présentons une approche qui repose sur la simulation et la fouille de données pour la sélection en temps-réel de règles de priorité pour la résolution du problème Job Shop. Lors de cette approche les résultats de problèmes d'affectation de règles de priorité obtenus par simulation sont repris par la fouille de données. Cette approche est composée de trois principales étapes. Durant la première étape la simulation teste les différentes combinaisons de règles, renvoie et stocke dans une base de données celle qui minimise le temps d'exécution total. Lors de la seconde étape, toutes les données historiques stockées par la simulation sont traitées et analysées et plusieurs arbres de décision sont créés à l'aide de la classification multi-labels. Ces derniers sont à leur tour utilisés lors de la troisième étape pour la résolution de nouveaux problèmes d'affectation de règles de priorité mais en temps-réel cette fois-ci, contrairement à la simulation. Durant ce chapitre, nous présentons une description complète et détaillée du problème d'ordonnancement Job Shop ainsi que ses principales

composantes. Ensuite, en nous inspirant des états de l'art présentés lors des trois premiers chapitres nous motivons le besoin d'une telle approche pour la résolution du Job Shop. Nous présentons enfin les expérimentations et les résultats, et clôturons avec une conclusion et des perspectives.

Dans le cinquième chapitre, nous reprenons les bases de l'approche proposée dans le quatrième chapitre, pour apporter une amélioration significative en termes de temps d'exécution total et de temps de calcul. Cette version améliorée est, elle aussi, basée sur la simulation et la fouille de données, mais intègre un algorithme génétique pour une meilleure exploration de l'espace de recherche. L'approche se base sur trois étapes. La première combine un algorithme génétique et un processus de simulation en vue de résoudre un certain nombre de problèmes Job Shop en affectant différentes règles de priorité aux machines et en stockant la meilleure solution identifiée dans une base de données. La seconde a pour but d'améliorer le processus de prétraitement de données et de construire un arbre de décision. Ce dernier, lors de la dernière étape, peut allouer en temps-réel une règle de priorité à chacune des machines de l'atelier en fonction des tâches en attente sur sa file. Lors de ce chapitre, nous présentons un état de l'art sur l'utilisation de l'algorithme génétique pour la résolution du Job Shop, nous discutons la motivation de l'utilisation de l'algorithme génétique pour l'amélioration de l'approche précédente. Ensuite, nous détaillons l'approche proposée et nous présentons les expérimentations, suivies par les résultats et discussion ainsi qu'une conclusion et des perspectives.

Enfin, dans le sixième et dernier chapitre nous nous focalisons sur l'extraction de règles priorité à base de fouille de données. Nous commençons d'abord par présenter le problème d'ordonnancement à Machine Unique avec la somme des retards pondérés comme objectif. Ensuite, nous présentons différentes approches ayant considéré l'algorithme génétique comme moyen pour résoudre ce problème ainsi que la motivation derrière l'utilisation de la fouille de données pour l'extraction de connaissances sous forme de règles de priorité. Lors de la phase suivante nous présentons l'approche proposée qui est décomposées à son tour de trois principales composantes. Lors de la première l'algorithme génétique proposé est utilisé pour résoudre plusieurs problèmes d'ordonnancement et les solutions obtenues sont stockées par la suite dans une base de données. Ces solutions après plusieurs opérations de prétraitement sont utilisées pour générer un arbre de décision à l'aide de l'algorithme C4.5. L'arbre de décision est enfin combiné avec trois heuristiques suggérées pour résoudre de nouveaux problèmes à Machine Unique en temps-réel. Nous présentons ensuite les résultats de l'algorithme génétique proposé ainsi que l'utilisation de la fouille de données pour l'extraction de règle de priorité, suivie par une conclusion et des perspectives.

Chacun de ces chapitres a donné lieu à un ensemble de communications et de publications dans des conférences internationales.

Publications internationales :

M. Habib Zahmani and B. Atmani, "Extraction of Dispatching Rules for Single Machine Total Weighted Tardiness using a Modified Genetic Algorithm and Data Mining," *Int. J. Manuf. Res.*, vol. 13, no. 1, pp. 1–25, 2018.

M. Habib Zahmani and B. Atmani, "A Data Mining Based Dispatching Rules Selection System for the Job Shop Scheduling Problem," *J. Adv. Manuf. Syst.*, to be published.

Communications internationales :

M. Habib Zahmani, B. Atmani, A. Bekrar, and N. Aissani, "A Real Time Data Mining Rules Selection Model for The Job Shop Scheduling Problem," in *45th International Conference on Computers & Industrial Engineering 2015 (CIE45)*, 2015, pp. 465–472.

M. Habib Zahmani, B. Atmani, A. Bekrar, and N. Aissani, "Multiple priority dispatching rules for the job shop scheduling problem," in *2015 3rd International Conference on Control, Engineering & Information Technology (CEIT)*, 2015, pp. 1–6.

M. Habib Zahmani, B. Atmani, and A. Bekrar, "Efficient Dispatching Rules Based on Data Mining for the Single Machine Scheduling Problem," in *Computer Science & Information Technology (CS & IT)*, 2015, pp. 199–208.

Publications en cours d'évaluation dans des journaux internationaux :

M. Habib Zahmani and B. Atmani, "Multiple Dispatching Rules Allocation in Real-Time using Data Mining, Genetic Algorithms, and Simulation," *Int. J. Manuf. Technol. Manag.*, submitted May 2017.

CHAPITRE

1. L'Ordonnancement dans les Systèmes de Production

Résumé : L'ordonnancement fait partie du domaine de la recherche opérationnelle, et est considéré comme la problématique la plus difficile dans un système de production. Dans la littérature on recense un très grand nombre d'ouvrages ou d'articles qui se sont intéressé à ce problème. Il a attiré de nombreux chercheurs de par sa complexité et les défis et challenges qu'il faut relever, ainsi que les atouts qu'il offre pour mieux maitriser et optimiser les coûts et/ou délais de production et leur impact sur les entreprises. L'ordonnancement consiste à organiser dans le temps le fonctionnement d'un atelier de production afin de maximiser l'utilisation de ses ressources matérielles et humaines disponibles et de confectionner un certain nombre de produits dans les meilleurs délais. Il doit aussi respecter les limites des ressources, comme le nombre d'heures de travails des ressources humaines ou les capacités des traitements des ressources matérielles, mais aussi les gammes des produits et les ordres de production.

Mots-clés : Ordonnancement · Système de Production · Optimisation · Classes d'Ordonnancement · Ateliers · Complexité · Modélisation et Représentation · Méthodes de Résolution · Etat de l'Art

1.1. Introduction

L'ordonnancement relève du domaine la Recherche Opérationnelle ainsi que de la Gestion de la Production. C'est un processus de prise de décision qu'on rencontre aussi dans de nombreux systèmes et/ou domaines tel que l'informatique, l'industrie, ou encore le transport. Son principal but est de réduire les coûts de la production et des délais de livraison améliorant par la même occasion l'efficacité de l'entreprise ainsi que son chiffre d'affaire.

L'ordonnancement est toujours présent dans les systèmes de production. Il y représente l'une des tâches les plus importantes pour planifier et séquencer les produits. Résoudre un problème d'ordonnancement consiste à déterminer des dates de démarrage pour toutes les tâches et prévoir ainsi leur date de fin d'exécution. Mais aussi déterminer l'ensemble des ressources nécessaires, humaines ou matérielles, requises en respectant les contraintes tout en en optimisant un ou plusieurs objectifs.

Cette planification permet aussi à l'entreprise de mettre en place des plans d'approvisionnement en matière première et de définir des calendriers de travail des employés. Il permet de prendre en considération les périodes de maintenance des équipements ainsi que les congés des employés ainsi que d'autres évènements pouvant altérer le calendrier de production. L'ordonnancement des ateliers de production doit nécessairement répondre au mieux aux besoins du client, au meilleur coût et dans les meilleurs délais.

L'ordonnancement passe par trois étapes [28] qui sont :

- La planification, a comme objectif de déterminer les différentes opérations à réaliser, les dates correspondantes ainsi que les moyens matériels et humains nécessaires à y affecter,
- L'exécution, qui met en place les différentes opérations définies dans la partie planification,
- Le contrôle, ce dernier détermine le décalage entre la planification et l'exécution en effectuant une comparaison. Le décalage se trouve soit au niveau des coûts, soit au niveau des dates de réalisation, ou autre.

Donc, les résultats attendus de l'ordonnancement des ateliers de production sont définis en plusieurs points [29] à savoir :

- L'affectation, qui définit l'ensemble des ressources humaines et/ou matérielles nécessaires aux tâches,
- Le séquencement, qui indique l'ordre de passage des tâches sur les ressources,
- Le datage, qui pour chaque tâche définit les dates de début et de fin d'exécution globales et sur les ressources.

On peut distinguer deux catégories de problèmes d'ordonnancement en fonction des informations disponible que l'on a du problème à résoudre :

- Problème statique, lorsque les tâches à ordonnancer ainsi que leurs dates d'arrivées sont connues au préalable,
- Problème dynamique, lorsque les tâches à ordonnancer sur une période donnée ne sont pas connues à l'avance et arrivent au fur et à mesure.

De plus, un ordonnancement se décompose en en deux parties [30] :

- L'ordonnancement prédictif : consiste à prévoir un certain nombre de décisions ou d'évènement à base de données prévisionnelles,
- L'ordonnancement réactif : consiste à adapter et/ou modifier les décisions prévues initialement en fonction de l'état courant du système en prenant en compte les déviations entre la réalité et le modèle. Comme par exemple des pannes de machines ou l'indisponibilité de ressources.

Au niveau des entreprises, l'ordonnancement peut concerner les ventes, la production, ou la maintenance. Son rôle est donc très important, car en plus de trouver des solutions pour bien maitriser la production par exemple, il permet aussi une gestion optimale et efficace.

Dans ce chapitre, nous présentons une vue d'ensemble du problème d'ordonnancement dans les systèmes de production. Nous rappelons les points clés de ces problèmes et les différentes notations utilisées. Nous nous intéressons ensuite aux différentes classes d'ordonnancement et aux types d'atelier existants et nous présentons brièvement l'état de l'art dans ce domaine. Par la suite, nous discutons de la complexité de ces problèmes et des différentes manières existantes pour leur représentativité. Enfin, nous abordons les familles et méthodes de résolution les plus populaires et nous expliquons brièvement leur fonctionnement.

1.2. Formulation d'un problème d'ordonnancement

Il est possible de rencontrer des problèmes d'ordonnancement dans plusieurs domaines tel que l'informatique avec la gestion des tâches dans les processeurs ; dans l'industrie avec la gestion de la production et les chaines de distribution, mais aussi dans d'autres domaines tel que la construction ou l'administration.

Les différentes données d'un problème d'ordonnancement peuvent varier d'un domaine à un autre. Dans les systèmes de production elles sont groupées en tâches, ressources, contraintes, et objectifs ou critères d'optimisation.

1.2.1. Les tâches

C'est l'ensemble des opérations requises pour la fabrication d'un produit. Une tâche, « *Job* » en anglais, J_i est une unité fondamentale présentée dans le temps par une date de début et une date de fin ainsi qu'un temps d'exécution sur une ou plusieurs machines [29].

Chaque tâche i où $i = \{0, 1, 2, \dots, N-1\}$, où N est le nombre de tâches, est composée d'une ou plusieurs d'opérations notées $\{O_{i,0}, O_{i,1}, \dots, O_{i,j}\}$, $j = \{0, 1, \dots, N_i - 1\}$. Où $O_{i,j}$ est la $j^{ème}$ opération du job i et N_i le nombre total d'opérations requises pour achever le job i (peut être

aussi dénotée m_{ij}). Chaque tâche $O_{i,j}$ doit être exécutée sur une machine où $m_{i,l}$ représente son numéro d'identification pendant un certain temps p_{ij}.

Une tâche, comme présenté dans la Figure 1.1, possède une date d'arrivée r_i (aucune opération de la tâche i ne peut être exécutée avant la date r_i), un temps d'exécution p_i (Équation 1.1), et une date d'échéance d_i. La tâche doit se terminer avant cette date sinon une pénalité peut être appliquée. Une tâche possède aussi une date de fin d'exécution C_i, qui correspond au moment de fin de toutes les opérations de cette tâche. Il existe aussi un autre attribut pouvant être associé à une tâche ou à une opération appelé « *setup-time* » ou temps de préparation noté s_{ijk}, i pour la tâche, j pour l'opération si le setup-time concerne celle-ci et k pour la machine. Il représente le temps requis par une machine pour procéder à l'exécution d'un certain type de tâches/opérations.

$$p_i = \sum_{l=0}^{N_i-1} p_{ij} \tag{1.1}$$

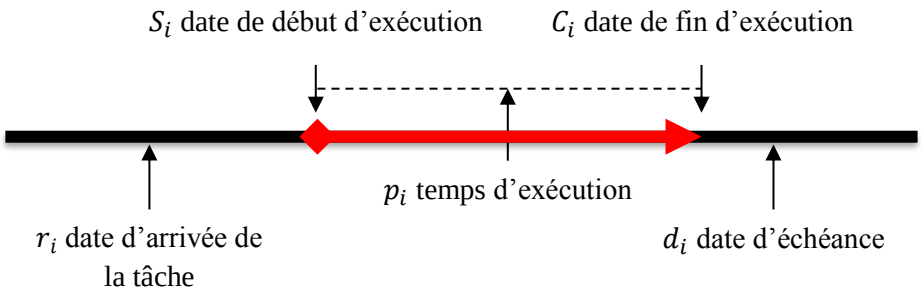

Figure 1.1. Exemple d'une tâche

Il est aussi possible d'associer une priorité d'exécution à chaque tâche afin de répondre aux besoins des clients ou à cause de la nature du produit, si le produit ou la matière première utilisée sont périssables par exemple. A ces tâches on affecte des poids w_i relatifs à leurs degrés de priorité.

On distingue deux types de tâches :

- Les tâches morcelables, ou on parle de préemption, qui peuvent être exécutées en plusieurs fois, les opérations pouvant ainsi être stoppées et reprisent plus tard, ce qui facilite la résolution de certains problèmes,
- Les tâches non morcelables ou indivisibles qui doivent être exécutées en une seule fois et ne peuvent être interrompues qu'une fois une des opérations soient terminées.

Pour certains types de tâches on peut parler de « *gamme de fabrication* ». C'est-à-dire, afin de d'achever un produit (tâche/job) celui-ci doit passer par un certain nombre de machines définis au préalable. Il existe trois types de gammes :

- Gamme linéaire, l'ordre des opérations est déterminé et doit être respecté,
- Gamme libre, l'ordre est totalement libre mais toutes les opérations doivent être achevées,

- Gamme mixte, est un mélange des gammes linéaire et libre. C'est-à-dire que certaines opérations doivent respecter un ordre de passage bien spécifique et d'autres non.

1.2.2. Les ressources

L'exécution des tâches ou opérations nécessite la mise en place et l'affectation d'un ensemble de moyens techniques et/ou de ressources humaines indispensables à la réalisation de ces tâches/opérations durant les intervalles de disponibilité.

Une ressource (machine) M_k où $k = \{0, 1, 2, \dots, M-1\}$, avec M désignant le nombre de machines, peut-être physique ou virtuelle, et d'une capacité limitée ou virtuellement illimitée, permettant l'exécution d'une opération (ou une tâche). On distingue plusieurs types de ressources [28] :

- Ressources renouvelables, sont disponible et peuvent être réutilisées après être allouées à une tâche comme les machines ou le personnel,
- Ressources consommables, peuvent être épuisées après l'exécution d'une tâche ou d'une opération, par exemple la matière première ou les ressources financières,
- Ressources partageables, sont des ressources pouvant être partagées entre plusieurs tâches/opérations.

Une autre classification des ressources existe aussi dans la littérature [31] :

- Ressources disjonctives, ne peuvent exécuter qu'une seule tâche à la fois. Les autres tâches ayant besoin de cette ressource doivent attendre dans une file d'attente la fin de la tâche en cours,
- Ressources cumulatives, contrairement aux disjonctives, peuvent exécuter plusieurs tâches en parallèle (deux ou plus).

Dans certains problèmes d'ordonnancement, on trouve la notion « *d'état de la ressource* » qui doit être prise en considération. Un changement d'état de la ressource, de fonctionnel à défaillant ou en maintenance, peut survenir et devrait être pris en considération lors de la mise en place de d'un ordonnancement. On parle d'ordonnancement dynamique lorsque ces types d'évènements sont pris en considération.

1.2.3. Les contraintes

Les contraintes expriment des restrictions sur les valeurs que peuvent prendre simultanément les variables de décision. On distingue deux types de contraintes, les contraintes liées au temps et celles liées aux ressources [31]–[33].

Contraintes temporelles :

Les contraintes de temps sont généralement liées à l'intervalle de temps pour l'exécution d'une tâche ou d'une opération, c'est-à-dire que l'exécution d'une tâche ne doit pas commencer avant une date r_i et se terminer avant une date d_i.

Une contrainte temporelle peut être spécifiée ainsi (Équation 1.2) :

$$r_i \leq t_i \leq d_i \text{ et } r_i \leq C_i \leq d_i \quad (1.2)$$

Où t_i (*Start time*) est l'instant où la tâche i commence son exécution effective. Dans les figures ci-dessous, on illustre un exemple d'exécution d'une tâche i avec un temps de disponibilité $r_i = 0$ et une date limite $d_i = 5$. Dans la Figure 1.2, l'ordonnancement respecte les contraintes, par contre dans la Figure 1.3, l'ordonnancement ne respecte pas les contraintes temporelles.

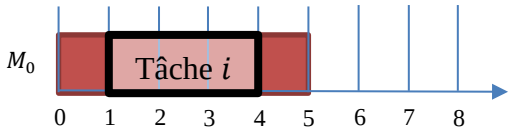

Figure 1.2. Ordonnancement réalisable

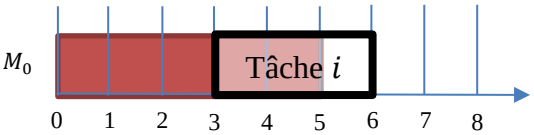

Figure 1.3. Ordonnancement non réalisable

Parmi contraintes temporelles on peut aussi définir les relations précédence, c'est-à-dire qu'une opération O_i doit s'exécuter avant l'opération O_j (gamme) la condition à respecter est donc date de fin d'exécution de l'opération i doit être avant la date de début de l'exécution de la tâche j (Équation 1.3).

$$C_i \leq t_j \quad (1.3)$$

Contraintes des ressources :

Ces contraintes spécifient la capacité ainsi que la disponibilité des ressources. Une ressource est renouvelable si elle est toujours disponible avec la même capacité initiale après l'exécution d'une opération. Elle est considérée comme consommable si après une exécution d'une tâche sa capacité est réduite et pouvant être par conséquent indisponible après un certain nombre d'opérations.

Une seconde contrainte consiste dans le nombre de tâches qu'une ressource peut opérer en même temps. La ressource est cumulative si elle peut faire passer plusieurs tâches à la fois et disjonctif si elle ne permet l'exécution que d'une seule et unique opération à la fois.

Dans la thèse de Tangour Toumi [30] une autre classification des contraintes est décrite. Elles sont classées en deux types :

La classe des contraintes endogènes :

Regroupe toutes les contraintes en relation directe avec le système de production tel que :

- La capacité des machines et des moyens de transports,

- Les dates de disponibilités des machines et des moyens de transport,
- Les séquences des opérations (gammes des produits).

La classe des contraintes exogènes :

Regroupe les contraintes imposées par l'environnement extérieur du système de production :

- Les dates d'échéance des produits imposées par le client ou par la nature du produit,
- Les priorités des produits ou commandes,
- Les retards possibles permis pour certains produits/commandes.

1.2.4. Les objectifs

Il existe généralement un nombre très important de solutions correspondant à chaque problème d'ordonnancement et chacune de ces solutions dispose de ses propres caractéristiques. Lors de la résolution de tels problèmes on favorise une solution à une autre selon leurs caractéristiques, selon l'environnement, et selon la nature des commandes ou d'autres critères. Il est rarement possible de satisfaire toutes les caractéristiques voulues dans une solution donnée. Toutefois, dans ce genre de cas il faut revoir les préférences à la baisse et essayer de trouver un compromis afin de trouver une solution comprenant plus ou moins les critères voulus [28].

Plusieurs mesures de performances peuvent être utilisées pour l'évaluation de l'ordonnancement. Ces mesures reposent sur les temps d'achèvement, les échéances, les inventaires, ou les coûts d'utilisation.

Dans la littérature on distingue plusieurs classifications des objectifs, en fonction de leurs critères ou par rapport aux contraintes. Dans la thèse de Djeridi [33] les objectifs sont regroupés en trois familles, à savoir :

- Critères liés au temps : tel que la minimisation du temps total d'exécution C_{max} (*makespan*) ou du retard maximal des tâches T_{max} (*maximum tardiness*),
- Critères liés aux ressources : minimisation de l'utilisation des ressources ou la réduction du temps d'utilisation,
- Critères liés aux coûts : minimisation des coûts de lancement, ou des coûts des retards.

Boukef Ben Othman [28] groupe les objectifs en deux familles, critères réguliers et critères irréguliers. Les critères réguliers constituent des fonctions décroissantes des dates de fin d'exécution. Par exemple, les dates de fin des opérations/tâches, la minimisation de la date maximale de fin d'exécution, leur moyenne, ou encore leur moyenne pondérée.

Quant aux critères irréguliers, ils regroupent, minimisation du coût de stockage des matières premières et/ou produits finis, l'équilibrage des charges machines, ou l'optimisation des changements d'outils.

Benbouzid Sitayeb [32] énumère plusieurs objectifs, comme présenté dans le Tableau 1.1 ci-dessous :

Tableau 1.1. Critères d'optimisation

Critères	Maximum	Somme	Somme pondérée
Fin de traitement	$C_{max} = max_{i=1,n} C_i$	$\bar{C} = \frac{1}{n}\sum_{i=1}^{n} C_i$	$\overline{C_w} = \sum_{i=1}^{n} w_i C_i$
Décalage	$L_{max} = max_{i=1,n} L_i$	$\bar{L} = \frac{1}{n}\sum_{i=1}^{n} L_i$	$\overline{L_w} = \sum_{i=1}^{n} w_i L_i$
Retard	$T_{max} = max_{i=1,n} T_i$	$\bar{T} = \frac{1}{n}\sum_{i=1}^{n} T_i$	$\overline{T_w} = \sum_{i=1}^{n} w_i T_i$
Avance	$E_{max} = max_{i=1,n} E_i$	$\bar{E} = \frac{1}{n}\sum_{i=1}^{n} E_i$	$\overline{E_w} = \sum_{i=1}^{n} w_i E_i$
Durée de flot	$F_{max} = max_{i=1,n} F_i$	$\bar{F} = \frac{1}{n}\sum_{i=1}^{n} F_i$	$\overline{F_w} = \sum_{i=1}^{n} w_i F_i$
Nombre de retards	-	$\bar{U} = \frac{1}{n}\sum_{i=1}^{n} U_i$	$\overline{U_w} = \sum_{i=1}^{n} w_i U_i$

Afin d'identifier une solution à un problème d'ordonnancement il est possible de considérer un ou plusieurs objectifs en fonction des besoins, la nature de l'atelier, l'environnement de l'entreprise, mais aussi les besoins des clients. On parle alors d'optimisation à « *objectif unique* » ou d'optimisation « *multi-objectifs* ». Dans les problèmes d'ordonnancement à objectif unique, le but est de trouver une solution qui optimise un seul critère prédéfini, par exemple minimiser le temps d'exécution total (fin de traitement ou makespan).

Bien que les recherches existantes soient axées sur les problèmes à objectifs unique, d'autres chercheurs tentent de résoudre des problèmes multi-objectifs [17], [34], [35]. Cependant, le mélange de plusieurs objectifs à la fois peut mener à la dégradation d'un ou plusieurs critères à un niveau inacceptable. Une fonction multi-objective est définie par l'Équation 1.4 :

$$\min f(x),\ x \in X \text{ où } f(x) = \left[f_1(x), f_2(x), \dots, f_p(x)\right]^T \tag{1.4}$$

Avec $f(x) \in Y$, p le nombre d'objectifs, X l'ensemble de solutions possibles, et Y l'espace objectif.

Il est donc nécessaire de trouver un compromis qui satisfait tous les objectifs simultanément. Par exemple, lors de la résolution d'un problème d'ordonnancement on peut considérer le temps total d'exécution et le nombre de tâches en retard comme objectifs. Améliorer un des objectifs peut dégrader la qualité du second, et vice-versa. Par conséquent trouver une bonne solution ou une solution optimale devient beaucoup plus difficile.

1.3. Les classes d'ordonnancement

Dans [31] et [36], les auteurs précisent que lorsqu'on tente de trouver un ordonnancement, on découvre plus d'une solution répondant aux critères sélectionnés. Par exemple, les dates de début de certaines opérations peuvent être retardées sans qu'il n'y ait d'effet sur la valeur du critère. Le but des classes d'ordonnancement (voir Figure 1.4) est de limiter la recherche à un sous-ensemble d'ordonnancements constitué de meilleures solutions, et ainsi éviter d'explorer tout l'espace de solutions et accélérer la recherche.

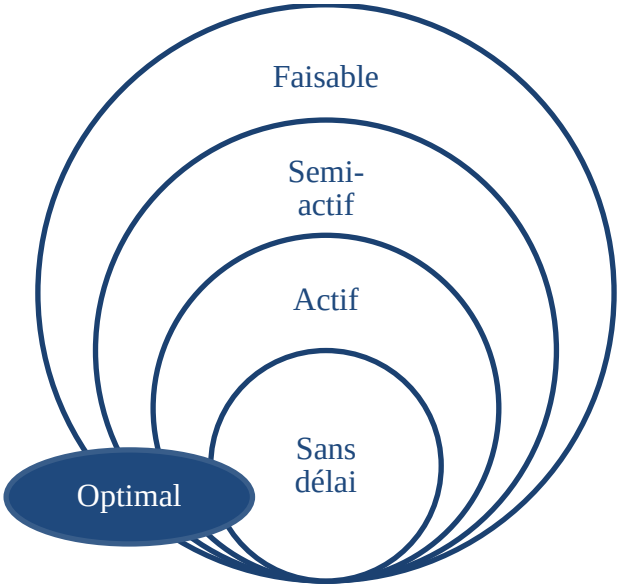

Figure 1.4. Les classes d'ordonnancement

1.3.1. Ordonnancement faisable

Un ordonnancement est dit faisable si et seulement si toutes les tâches respectent l'ensemble des contraintes du problème d'ordonnancement.

1.3.2. Ordonnancement semi-actif

Un ordonnancement est semi-actif s'il n'est pas possible de commencer une opération plus tôt sans pour autant violer une contrainte ou sans changer l'ordre des opérations sur les ressources (machines). Dans ce type d'ordonnancement la seule solution pour optimiser la fonction « objectif » est de réordonnancer toutes les tâches/opérations sur les ressources. Si un ordonnancement semi-actif (l'ordre de passage des tâches) est représenté avec un diagramme de Gantt, il est impossible de décaler une tâche à gauche.

1.3.3. Ordonnancement actif

Un ordonnancement est actif si tout décalage oblige à retarder la date de début d'exécution d'une autre opération. Il est aussi impossible d'insérer une nouvelle opération entre de deux tâches sans remettre en cause l'ordre d'exécution ou violer certaines contraintes. Un ordonnancement actif est aussi en même temps semi-actif. Sur un diagramme de Gantt, un ordonnancement est considéré comme actif s'il est impossible de décaler une tâche à gauche.

1.3.4. Ordonnancement sans délai

Un ordonnancement est « *sans délai* » si à chaque instant toute opération qui dispose des ressources nécessaires est commencée. C'est-à-dire qu'une machine ne doit pas rester inactive alors qu'une tâche/opération est présente dans la file d'attente de la machine. On ne doit pas retarder l'exécution d'une tâche si celle-ci est en attente et si la ressource est disponible.

1.3.5. Ordonnancement optimal

Un ordonnancement est optimal s'il donne les meilleures valeurs possibles étant donné un ensemble d'objectifs à optimiser. Dans un problème d'ordonnancement il est possible de trouver plus d'une solution qui donnent le même résultat (même valeur de la fonction objectif)

Afin de concevoir un algorithme d'optimisation, il faut impérativement choisir l'une des classes. Parmi ces classes, la tendance se porte sur celle contenant le plus faible nombre d'ordonnancements. Pourtant, si l'on veut que l'algorithme puisse trouver la solution optimale, il faudrait que cette classe la contienne.

1.4. Les ateliers

Il est possible de classer (voir Figure 1.5) les problèmes d'ordonnancement en fonction du nombre des machines et des gammes des produits.

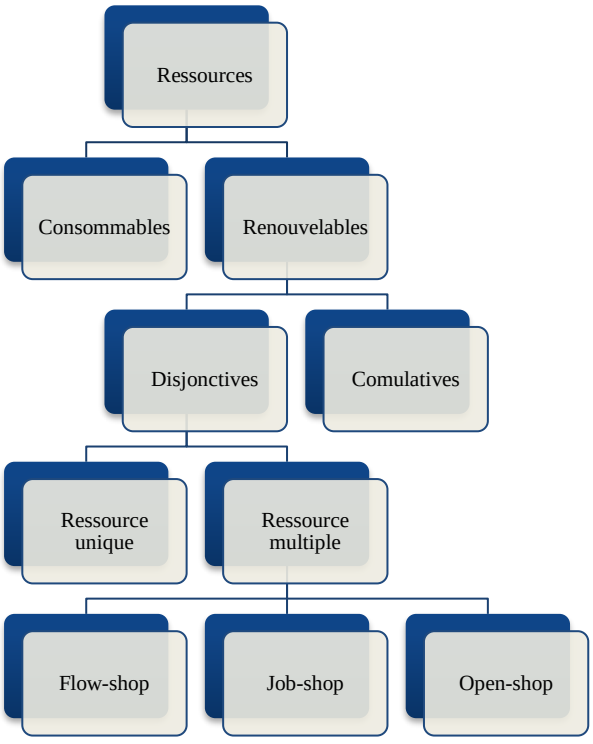

Figure 1.5. Classification des ateliers

Un atelier est caractérisé par le nombre de machines qu'il contient ainsi que par leurs types. Les problèmes d'ordonnancement peuvent être classés en deux catégories, la première regroupe les

problèmes pour lesquels chaque tâche nécessite une seule machine, la deuxième regroupe les problèmes où les tâches nécessitent plusieurs machines pour leurs exécutions [29], [30].

1.4.1. Single Machine

L'atelier à Machine Unique, ou appelé « *Single Machine* », et ses différentes variantes ont fait l'objet d'études et de recherches approfondies. Notamment les ateliers à Machine Unique avec la somme des retards pondérés comme objectif, ou la somme des retards, l'avance, ou autres objectifs. Mais aussi d'autres variantes liées aux caractéristiques des tâches à exécuter, comme celles avec temps de préparation, dates d'échéance communes, ou familles de produits. Dans la littérature il existe beaucoup d'approches proposées pour la résolution du problème à une machine. Dans le Tableau 1.2 nous présentons un certain nombre de travaux de recherche ayant porté sur l'atelier à Machine Unique avec un descriptif de la méthode utilisée, l'objectif ou les objectifs considérés. Les contraintes liées aux tâches comme les dates d'arrivée ou les temps de préparation ainsi que le nombre de tâche considéré lors des expérimentations sont aussi présentés dans le Tableau 1.2. Il est utile de noter que les dates d'arrivée des tâches sont fixées par défaut à 0, c'est-à-dire que toutes les tâches arrivent au début de l'exécution et en même temps, à part où dans la colonne des contraintes sur les tâches est spécifié « date d'arrivée » qui veut dire que chaque tâche possède sa propre date d'arrivée sur l'atelier qui diffère des autres tâches. Une description complète du Single Machine avec somme des retards pondérés est donnée dans la Section 6.2 du Chapitre 6.

Tableau 1.2. Etat de l'art du problème Single Machine

Réf.	Année	Méthode de résolution	Objectif	Contraintes tâches	Perturbations	Nombre de tâches
[37]	1990	Heuristique proposée	Total tardiness	Temps de préparation	Pannes machine	10, 20
[38]	1997	Heuristiques proposées LIST, CMHEU	Total weighted completion time Maximum completion time	-	Arrivée dynamique des tâches	Aléatoire
[39]	2000	Algorithme génétique	Total tardiness	Temps de préparation	-	6, 8, 10, 15, 25, 30, 35, 45, 50, 70, 90, 110
[40]	2001	Deux heuristiques proposées (proposition de benchmarks)	Total earliness Total tardiness	Date d'échéance commune	-	10, 20, 50, 100,200, 500, 1000
[41]	2002	Prédictif-Réactif (heuristique)	Total weighted tardiness Makespan	Date d'arrivée dynamique	Arrivée dynamique des tâches	-
[42]	2002	Heuristiques proposées	Number of tardy jobs		Arrivée dynamique des tâches	5, 10
[43]	2003	Méthode évolutionnaire Recuit simulé Recherche locale (Threshold Accepting)	Total earliness Total tardiness	Date d'échéance commune	-	10, 20, 50, 100, 200, 500, 1000
[44]	2004	On-line algorithm (heuristique)	Total weighted completion time	-	Arrivée dynamique des tâches	-
[45]	2004	Algorithme génétique	Total weighted tardiness	Dates d'arrivée Dates d'échéance	-	20, 40, 60, 800, 100
[46]	2005	Algorithme génétique	Total weighted tardiness	-	-	7, 10, 25, 50, 100, 200, 500

Réf.	Année	Méthode de résolution	Objectif	Contraintes tâches	Perturbations	Nombre de tâches
[47]	2006	Heuristique proposée (théorème)	Makespan	-	Pannes machine	-
[48]	2006	Algorithme génétique hybride	Makespan	Traitement par lots	-	
[49]	2007	Algorithme génétique	Total weighted tardiness Stability	Dates d'arrivée	Pannes machine	20, 40, 60, 80, 100
[50]	2007	Algorithme génétique	Total weighted tardiness	-	-	40, 50, 100
[51]	2008	Hybridation de Modified Due Date avec un algorithme de recherche locale Hybridation d'une heuristique backward phase avec une recherche locale	Total weighted tardiness	-	-	40, 50, 100
[52]	2008	On-line algorithm (heuristique)	Makespan + delivery time	Dates d'arrivée Temps de livraison	-	-
[53]	2009	Algorithme génétique modifié (Experienced Learning GA)	Total weighted tardiness	-	-	40, 50, 100
[54]	2009	Modified Optimized Surrogate Measure Heuristic	Total weighted earliness	Dates d'arrivée	Pannes machine	50, 100, 200
[55]	2010	Recherche locale	Total weighted tardiness	-	-	40, 50, 100
[56]	2010	Algorithme génétique et arbres de décision	Weighted maximum lateness	-	-	10

Réf.	Année	Méthode de résolution	Objectif	Contraintes tâches	Perturbations	Nombre de tâches
[57]	2010	Algorithme génétique	Total tardiness Machine utilization Sum of cycle times	Famille de tâches Dates d'arrivée	-	10, 20, 50, 100
[58]	2011	Recuit simulé	Total weighted tardiness	Dates d'arrivée	Pannes machine	30, 50, 70, 90
[59]	2011	Heuristiques proposées	Makespan Makespan variation	Détérioration des tâches	Pannes machine	-
[60]	2012	Heuristique proposée	Makespan	Famille de tâches Temps de préparation	Arrivée dynamique des tâches	50, 100, 200
[61]	2013	Mixed Dispatching Rules (heuristique)	Total weighted tardiness	-	-	40, 50, 100
[62]	2014	Heuristiques proposées	Maximum lateness	-	Pannes machine	50, 10, 15, 50, 100,150, 20, 300, 500
[35]	2015	Algorithme génétique et arbres de décision	Total weighted tardiness	-	-	40, 50, 100
[63]	2015	Colonies de fourmis	Total weighted earliness	-	-	40, 50, 100, 150, 200, 250, 300
[64]	2016	Heuristique proposée	Total tardiness	Famille de tâches Temps de préparation	-	8, 10, 12,15, 20,30, 40, 50

Le lecteur peut aussi consulter les états de l'art présentés par Koulamas [65] ayant porté sur le Single Machine avec la somme des retards pondérés ou celui d'Adamu & Adewumi [5] ayant traité du Single Machine avec le nombre pondéré des tâche en retard (*weighted number of tardy jobs*).

Dans cette première catégorie regroupant les ateliers à machine unique on peut inclure les ateliers à machines parallèles ou non dédiées [29]. Ce genre d'ateliers se caractérise par le fait que plusieurs machines sont disponibles pour l'exécution d'une tâche qui n'en nécessite qu'une seule.

On peut distinguer trois sous-classes d'ateliers selon la vitesse d'exécutions des machines qui le composent :

- Ateliers à machines identiques : chaque tâche peut s'exécuter sur n'importe quelle machine avec un même temps d'exécution (quand la machine est libre),
- Ateliers à machines uniformes : chaque machine possède sa propre vitesse et ceci indépendamment des tâches à exécuter,
- Ateliers à machines indépendantes : la vitesse des machines dépend de la tâche à effectuer.

Quant à la deuxième classe de problèmes, elle englobe les « *ateliers composés* ». Ces ateliers sont composés de m stations différentes. Chaque station comporte une ou plusieurs machines en parallèle. Les tâches à exécuter sont composées de n opérations et en fonction du mode de passage des opérations sur les machine on peut distinguer trois types d'ateliers qui sont le Job Shop, le Flow Shop et Open Shop.

1.4.2. Flow Shop

Les ateliers de type flow-shop ou aussi appelés « *ateliers à cheminement unique* », sont des ateliers où une ligne de fabrication est constituée de plusieurs machines en série (voir Figure 1.6). Toutes les tâches passent par les mêmes machines dans le même ordre. En ce qui concerne les ateliers de type « *Flow Shop Hybride* », une machine peut exister en plusieurs exemplaires similaires fonctionnant en parallèle. Ce genre d'ateliers sont est dédié à la production de masse de peu de variété de produits.

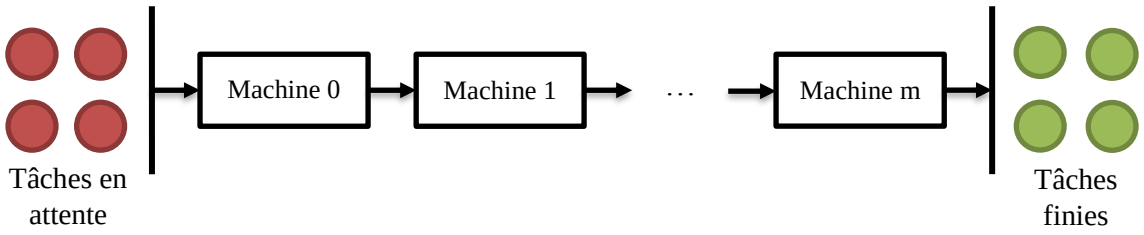

Figure 1.6. Atelier Flow Shop

Ce type d'atelier a aussi fait l'objet de nombreuses recherches parmi lesquelles on cite notamment Rajendran & Holthaus [66], qui proposent une étude comparative de règles de priorité pour la résolution d'un atelier Flow Shop composé de 10 machines où le but est de

déterminer l'impact de règles sur plusieurs objectifs liés au temps total d'exécution à savoir « *mean, maximum and variance of flowtime* ». Les expérimentations sont par la suite étendues aux ateliers de type Job Shop. Dans la même perspective, Song et al. [25] proposent une approche basée sur la simulation d'un atelier Flow Shop hybride pour comparer l'impact de différentes règles de priorité sur le temps d'exécution total (makespan). Dans cet atelier la production se fait en deux phases. Durant la première, trois machines similaires sont disponibles pour le préchauffage des matériaux et durant la seconde phase de production, une seule machine est disponible. Les temps d'exécution des tâches sont connus à l'avance. D'autres travaux de recherche portant sur le Flow Shop existe comme l'utilisation d'un algorithme génétique pour la résolution d'un problème Flow Shop hybride [67] ou l'utilisation de la fouille de données pour la génération de nouvelles règles de priorité [68], [69].

1.4.3. Job Shop

Dans ce type d'ateliers connu sous le terme « *atelier à cheminement multiple* », chaque tâche a son propre séquencement sur les machines comme présenté dans la Figure 1.7. Ce sont des ateliers où les opérations sont réalisées selon un ordre bien déterminé, variant selon la tâche à exécuter. Le « *Job Shop Flexible* » est une variante du Job Shop classique, où l'on dispose de plusieurs machines parallèles capables de faire un sous-ensemble d'opérations.

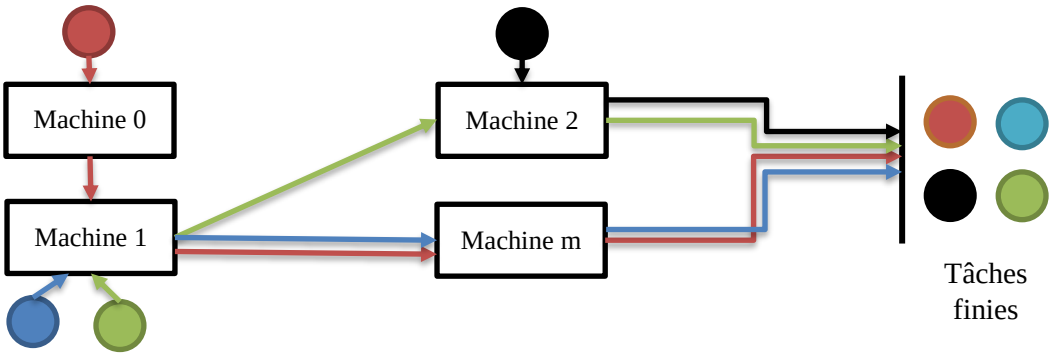

Figure 1.7. Atelier Job Shop

Le Job Shop est considéré comme un des problèmes les plus généraux car les autres problèmes à machine multiple comme le Flow Shop ou le open shop peuvent être considérés comme des cas spéciaux du problème Job Shop [31]. Ceci explique en partie l'intérêt qui lui est porté par les chercheurs et praticiens ainsi que son développement. Le Job Shop a été largement considéré dans l'état de l'art comme montré dans le Tableau 1.3 ci-dessous. Dans ce Tableau 1.3 nous spécifions la méthode de résolution proposée, les contraintes liées aux tâches comme les dates d'arrivée ou les temps de préparation, ainsi que les caractéristiques du système de production comme la prise en charge des pannes machines ou la flexibilité de l'atelier. Nous spécifions aussi les critères d'optimisation pris en considération et le nombre de tâches et machines envisagés lors des expérimentations. Une description plus complète de ce problème est présentée dans la Section 4.2 du Chapitre 4.

Tableau 1.3. Etat de l'art sur le Job Shop

Réf.	Année	Méthode de résolution	Objectif	Contraintes tâches	Autres caractéristiques du problème	Nombre tâches (T) et machines (M)
[11]	1991	Heuristique proposée + Branch & Bound	Makespan	-	-	T : 10 M : 10
[13]	1994	Algorithme génétique	Makespan	-	-	T : 6, 10, 20 M : 5, 6, 10
[70]	1995	Algorithme génétique	Makespan	-	-	T : 6, 10, 20 M : 5, 6, 10
[71]	1997	PT+WINQ, PT+WINQ+AT, PT+WINQ+SL, PT+WINQ+AT+SL, Modified COVERT (règle de priorité)	Mean, max, mean flow time Tardy jobs Mean, max, variance tardiness	Dates d'échéance	-	T : 500 M : 10
[72]	2000	Heuristiques proposées	Makespan Mean tardiness	-	Pannes machine	T : 9, 18, 12, 24 M : 6, 12, 6, 12
[73]	2000	Algorithme génétique + fouille de données	Makespan	-	-	T : 6 M : 6
[74]	2001	Recuit simulé + algorithme génétique	Makespan	-	-	T : 6, 10, 15, 20, 30 M : 5, 6, 10, 15
[34]	2003	Algorithme génétique	Makespan Workload	Dates d'arrivée	Job Shop flexible	T : 10 M : 7, 10
[75]	2006	Règle de priorité (proposée)	Weighted mean tardiness	Dates d'échéance Poids Dates d'arrivée	Dynamique	T : 2000 M : 10
[76]	2006	Règles de priorité	Tardy rate Mean, max tardiness	Dates d'échéance	-	T : 20, 30, 40, 50 M : 15, 20

Réf.	Année	Méthode de résolution	Objectif	Contraintes tâches	Autres caractéristiques du problème	Nombre tâches (T) et machines (M)
[77]	2007	Recherche Tabou	Makespan	-	-	T : 10, 15, 20 M : 5, 10
[78]	2007	Heuristique proposée	Makespan Maximum tardiness	Dates d'échéance	Pannes machines (réactif)	T : 60, 90, 120 M : 4, 8
[15]	2007	Heuristiques littérature FIFO FASFS/FIFO $WSPT^h$/FIFO $WTWKR^h$/RRP Heuristiques proposées $WTWKR^\tau$/LFT $WECT^\tau$/LFT $WLFT^\tau$/ECT $WEFT^\tau$/ECT $WJDD^\tau$/ LFT $WJDD^\tau$/ECT	Weighted Mean Flowtime Maximum Weighted Flowtime Weighted Mean Tardiness Weighted Mean Conditional Tardiness Maximum Weighted Tardiness Percentage of Tardy Jobs	Dates d'arrivée Dates d'échéance Poids	-	Niveau utilisation machine 80, 90%
[79]	2008	Touring Ant Colony Optimization + règles de priorité	Mean absolute percentage error (classification)	Dates d'arrivée	Possibilité de d'exécuter une tâche plusieurs fois sur une machine	T : 18000 M : 24
[80]	2008	Algorithme génétique	Makespan	-	Job Shop flexible	T :10, 15, 20 M : 4, 6, 8, 10, 15
[81]	2010	Mixed-integer linear programming model (MILP-1, MILP-2)	Makespan	-	Job Shop flexible	T : 5, 10, 20 M : 5, 10
[82]	2010	Recherche tabou, arbres de décision et règles de priorité	Total tardiness	-	-	T : 6 M : 6

Réf.	Année	Méthode de résolution	Objectif	Contraintes tâches	Autres caractéristiques du problème	Nombre tâches (T) et machines (M)
[83]	2010	Colonies de fourmis	Makespan	-	Job Shop flexible	T : 50-200 M :10-20
[84]	2010	Algorithme génétique	Makespan	-	-	T : 6, 10, 15, 20 M : 5, 6, 10, 15
[85]	2011	Algorithme génétique + recherche locale	Makespan	-	Avec/sans pannes machines	T : 15 M : 15
[86]	2011	Algorithme génétique	Makespan	-	-	T : 10 M : 4
[87]	2011	4 mécanismes ORR + 6 règles de priorité	Mean absolute deviation Mean shop floor throughput time	Dates d'échéance Dates d'arrivée	Order review/release	Niveau utilisation machine 80, 85, 90%
[88]	2011	Algorithme génétique + règles de priorité	Total tardiness	Dates d'échéance	-	T : 50, 100, 200 M :
[89]	2011	Algorithme génétique	Makespan	-	Job Shop flexible	T : 7 M : 7
[90]	2012	Heuristique proposée	Completion times + intermediate operation completion time	Dates d'échéance	-	T : 10 M : 10
[17]	2012	Règles de priorité (propositions et comparaison)	Makespan Total average time Total wait time Average queue Waiting time Queue length	-	Temps de transport	T : 10 M : 14

Réf.	Année	Méthode de résolution	Objectif	Contraintes tâches	Autres caractéristiques du problème	Nombre tâches (T) et machines (M)
[91]	2012	Recherche taboue + arbres de décision	Maximum lateness	-	-	T : 6 M : 6
[18]	2013	Weight Biased Modified RRrule (règle de priorité)	Mean tardiness	Poids Dates d'échéance	-	T : aléatoire M : 6
[23]	2013	Programmation génétique (génération de règles de priorité)	Makespan Total weighted tardiness	Dates d'échéance Poids	-	T : 10, 15, 20, 50, 100 M : 5, 10, 15, 20
[92]	2013	Proposition de benchmarks	Multiple	-	-	-
[26]	2014	Règles de priorité (étude comparative)	Total tardiness Tardy jobs Makespan	Arrivée dynamique des tâches Temps d'exécution aléatoires Dates d'échéance	-	T : aléatoire M : 8
[9]	2015	Optimisation par essaim particulaire	Makespan	-	-	T : 4 M : 4
[93]	2015	Règles de priorité (étude comparative)	Makespan Mean, max flow time Mean, max tardiness Nbr tardy jobs Total setups Mean setup time	Temps de préparation Dates d'échéance Dates d'arrivée	Arrivé dynamique des tâches	Niveau utilisation machine 85, 90%
[94]	2015	Algorithme génétique +	Makespan	Temps d'exécution flou	Job Shop flexible	T : 10 M : 10
[95]	2015	Règles de priorité + arbres de décision	Mean tardiness	Dates d'échéance	Pannes machine	T : aléatoire M : 4

Il existe plusieurs états de l'art récents portant sur l'ordonnancement dans les ateliers Job Shop. Par exemple Gupta [96] énumère un certain de nombre de travaux dans des ateliers à acheminement multiple pour la fabrication des semi-conducteurs. Fan et al. [21] font le tour des publications où les règles de priorité sont utilisées pour la résolution du Job Shop. Quant à Çaliş & Bulkan [7], ils s'intéressent aux travaux de recherche sur l'intégration de l'intelligence artificielle pour la résolution de problèmes Job Shop.

1.4.4. Open Shop

Comme le Job Shop l'Open Shop a un acheminement multiple (voir Figure 1.8), seulement la différence est que les tâches (jobs) ne possèdent pas de gammes fixes (pas de contrainte de précédence). Donc, l'ordre de passage des opérations est ainsi quelconque. Ces ateliers sont communément appelés « *ateliers à acheminement libre* ». Ce dernier est déterminé lors de l'ordonnancement.

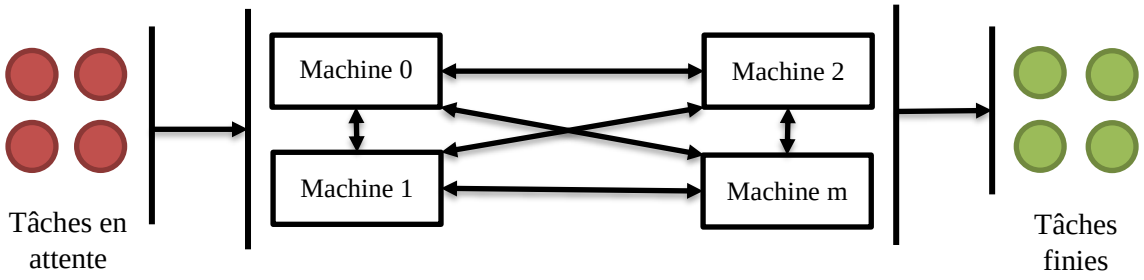

Figure 1.8. Atelier Open Shop

Il existe dans la littérature beaucoup de travaux qui se sont intéressé à ce type d'atelier, notamment Bai et al. [97] qui propose une approche pour résoudre l'open shop flexible ou Zhang et al. [98] qui ont proposé une approche pour la résolution d'un problème open shop à deux machines avec le temps d'exécution total comme objectif.

1.5. Complexité

La difficulté d'un problème d'ordonnancement est directement liée à ses caractéristiques, comme le nombre de machines, les nombre de tâches, ou les gammes. Cette difficulté est aussi définie suivant la complexité de la méthode de résolution utilisée [60], [99]. Dans cette section nous présentons des notions générales sur la complexité avant de présenter les différentes méthodes de résolution utilisées pour les problèmes d'ordonnancement.

La théorie de la complexité offre la possibilité de classer les problèmes, ordonnancement entre autres, en problèmes faciles ou difficiles. On distingue deux catégories de complexité, à savoir, complexité algorithmique et complexité problématique. Les problèmes d'ordonnancement ont été démontré comme étant difficiles « *NP-Hard* » [38], [40], [41].

1.5.1. Complexité algorithmique

Lors de la résolution de problèmes d'optimisation, ordonnancement notamment, il est impératif de déterminer le temps d'exécution requis ainsi que l'espace mémoire nécessaire pour chaque algorithme de résolution. Ceci dans le but de choisir celui qui nécessite le moins de temps CPU et le moins de mémoire et qu'en même temps puisse converger rapidement et retourner une solution dans les meilleurs délais. Déterminer la complexité des algorithmes de résolution permet aussi de classifier le problème à résoudre, et faire la distinction entre problème d'optimisation et problème de décision [30], [100].

La fonction $f(n)$ est appelée complexité en temps algorithme. Elle représente le nombre maximal d'opérations élémentaires requises pour résoudre un problème de taille n où n représente le nombre de variables décrivant le problème (comme le nombre de machines ou le nombre de tâches) [29], [30].

Un algorithme de résolution a une complexité $o(f(n))$, s'il existe une constante $c > 0$ et un entier n_0 où $\forall\, n \geq\ n_0,\ T(n) \leq cf(n)$.

Un algorithme est considéré comme polynômial si sa fonction de complexité $f(n) \in o(p(n))$ où p est un polynôme en n. S'il existe une constance $c > 0$ tel que $f(n) \in o(n^k)$. Si aucune fonction polynômiale n'existe, l'algorithme est dit « *Non Polynômial* » (NP) ou exponentiel [30].

1.5.2. Complexité problématique

Cette complexité dépend directement du problème à résoudre mais aussi de la méthode de résolution utilisée pour trouver une solution optimale correspondante à une fonction objectif définie initialement. Les problèmes d'ordonnancement sont regroupés en deux familles en fonction de leur degré de complexité [38], [101], [102].

- Les problèmes décidables, pouvant être résolus, qui sont de classe P ou NP [101],
- Les problèmes indécidables, ne pouvant être résolus, qui sont les problèmes d'ordonnancement les plus difficiles pour lesquels il n'existe aucune méthode de résolution connue [30].

Dans la complexité problématique il est aussi impératif de faire la distinction entre problème d'optimisation ou problème de décision. Dans un problème d'optimisation le but est de trouver une solution faisable optimisant une fonction « *objectif* ». Cependant, dans un problème de décision il est possible de répondre uniquement par « *oui* » ou par « *non* ». Harrath [29] illustre un problème de décision avec l'exemple suivant. Soit un problème d'ordonnancement avec n tâches et soit y un entier positif. Un problème de décision associé à ce problème d'ordonnancement est de savoir s'il existe une solution avec un $C_{max} \leq y$. La réponse à ce problème de décision pouvant être « *oui* » ou « *non* ».

Les problèmes de décision peuvent aussi être en classés en deux catégorie P s'il existe un algorithme polynômial pour le résoudre ou NP s'il ne peut pas être résolu en un temps

polynômial en utilisant un algorithme déterministe [65]. Mais il est possible de le résoudre en un temps polynômial avec une méthode approchée [77], [80].

1.6. Modélisation et représentation des problèmes d'ordonnancement

La modélisation est un élément crucial lors de la résolution d'un problème d'ordonnancement. C'est une écriture simplifiée de toutes les données d'un problème d'ordonnancement en se basant sur un formalisme bien adapté pour représenter un problème choisi [28], [103]. Dans la littérature, on distingue deux méthodes pour la modélisation des problèmes d'ordonnancement, à savoir, les méthodes mathématiques et les méthodes graphiques.

1.6.1. Les méthodes graphiques

Parmi les méthodes de représentation graphiques, il est possible d'utiliser le « *Graphe Potentiel-Tâches* » ou la méthode « *Pert* » mais aussi les réseaux de « *Petri* » qui sont considérés comme l'une des plus importantes évolutions des méthodes graphiques. Dans ce qui suit, nous décrivons quelques méthodes basées sur les graphiques.

1.6.1.1. Graphe Potentiel-Tâches

La représentation d'un problème d'ordonnancement est possible en utilisant un graphe. Dans ce type de représentations, les tâches correspondent à des nœuds, et les contraintes de précédence sont représentées par des arcs conjonctifs en incluant les durées de chaque tâche. En ce qui concerne les ressources, elles sont sous la forme d'arcs disjonctifs [103]. Pour représenter le début et la fin d'un ordonnancement, deux nœuds fictifs peuvent être rajoutés au graphe.

Un problème d'ordonnancement peut alors être représenté par un graphe potentiel-tâches $G(X;U)$ où X est l'ensemble des sommets et U l'ensemble des arcs. Si on considère T comme étant l'ensemble des n tâches à réaliser, X est alors l'union de l'ensemble T et l'ensemble $\{s,p\}$ où s représente la tâche de début fictif et p la tâche de la fin fictive. L'ensemble des arcs U est défini par $\{(s;i),(i;j)$ et $(i;p)\}$ où $i,j \in T$. Cet ensemble représente les contraintes des tâches [103].

Les valeurs des arcs en fonction de leurs types correspondent à :

- Arc de type $(s;i)$ représente la date disponibilité de la tâche i,
- Arc de type $(i;j)$ représente le temps d'exécution de la tâche i,
- Arc de type $(i;p)$ porte sur la durée opératoire de i,
- Il est aussi possible de rajouter un arc de type $(i;j)$ avec une valeur négative (arc inversé) si la tâche j doit commencer immédiatement après la tâche i (nécessaire pour représenter les séquences des opérations).

Dans la Figure 1.9, nous présentons un problème d'ordonnancement [29] composé de 8 tâches numérotées consécutivement de 1 à 8 et de trois ressources A, B, et C. Les tâches 1, 5, et 6 requièrent la ressource A, les tâches 2, 4, et 7 la ressource B et les tâches 3 et 8 la ressource C.

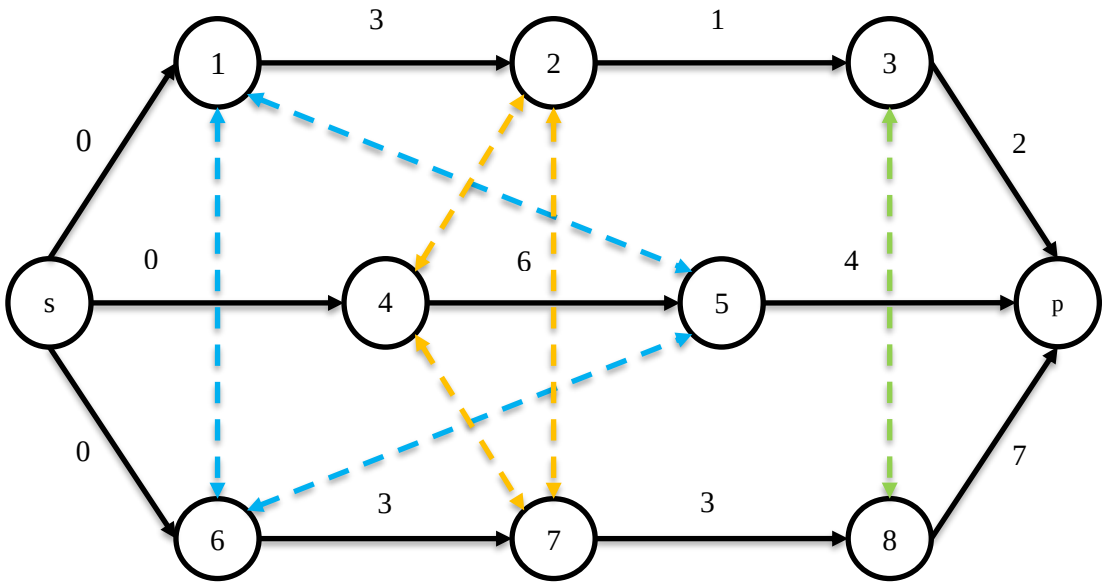

Figure 1.9. Exemple d'un graphe potentiel-tâches

1.6.1.2. Méthode Pert

Dans cette représentation dite Pert, pour « *Program Evaluation and Research Task* », il est aussi possible de représenter un problème d'ordonnancement par un graphe. Les tâches correspondent à des arcs et chaque valeur associée à un arc représente la durée d'exécution. Les sommets quant à eux, marquent la fin d'exécution d'une ou de plusieurs tâches, ils permettent ainsi de garder trace de la succession des évènements.

1.6.1.3. Réseaux de Petri

Les réseaux de Petri (RdP) ont été proposés en 1962 par Carl Adam Petri dans sa thèse de doctorat. Les RdP sont de puissants outils graphiques et mathématiques qui permettent la modélisation et la vérification des comportements dynamiques des systèmes à évènements discrets tel que les systèmes manufacturiers.

Réseau de Petri non marqué :

Un RdP non marqué est un graphe orienté biparti composé d'arcs, de places, et de transitions. Les arcs relient les places aux transitions. On appelle RdP non marqué un quadruplet $Q =< P, T, I, O >$ où :

- P est un ensemble fini non vide de places,
- T est un ensemble fini non vide de transitions,
- $P \cap T = \emptyset$,

- $I(T_i)$ est l'ensemble des places qui sont en entrée de la transition i. $I: P \times T \rightarrow IN$ (l'ensemble des entiers naturels). Si $I(p,t) > 0$ alors l'arc est orienté de p vers t si $I(p,t) = 0$ alors l'arc n'est pas orienté,
- $O(T_i)$ représente l'ensemble des places qui sont en sortie de la transition i. $O: T \times P \rightarrow IN$. Si $O(t,p) > 0$ alors l'arc est orienté de t vers p, et n'est pas orienté quand $O(t,p) = 0$.

Réseau de Petri marqué :

Un RdP marqué est un doublet $< Q, M_0 >$ où Q est un RdP non marqué et M_0 le marquage initial [103].

- Chaque place doit contenir un nombre $M(p_i) \geq 0$ entier de marques ou de jetons,
- Le marquage du réseau M est défini par l'ensemble des valeurs $M(p_i)$,
- Le marquage à un instant spécifique définit l'état du système.

Réseau de Petri borné :

Une place p_i est dite bornée par un marquage initial M_0, si et seulement si, pour tout marquage accessible de M_0 le nombre de marques de p_i reste borné. On parle aussi de place « *k-bornée* » si le nombre de marques de $p_i \leq k$.

D'autres extensions de réseaux de Petri liés au temps existent, comme les réseaux associés aux transitions dits « *t-temporisés* » ou les réseaux liés aux places « *p-temporisés* » [103].

1.6.1.4. Modélisation polyédrique

Il est aussi possible d'utiliser un autre type de modélisation graphique connu sous le nom de « *modélisation polyédrique* » [30]. Cette modélisation consiste à trouver des contraintes linéaires dans le but de définir des « *demi-espaces* ». Une fois tous les demi-espaces définis, leur intersection permet de construire un polyèdre dont les sommets constituent la solution au problème d'ordonnancement.

On parle notamment de « *programmation linéaire* » (*PL*) lorsque la fonction objectif définie est linéaire. L'utilisation de la PL permet pour certains problèmes d'atteindre la ou les solutions optimales.

La modélisation polyédrique est utilisée pour la résolution des problèmes d'ordonnancement [30], [104]. T est l'ensemble de vecteurs t où ce dernier est composé des dates de débuts calculées de chaque opération à ordonnancer. Le polyèdre P représente l'enveloppe convexe des points T.

1.6.2. La modélisation mathématique

La modélisation mathématique consiste à représenter les données, les contraintes, et la fonction objectif du problème d'ordonnancement sous forme d'équations et/ou d'inéquations mathématiques. L'utilisation de ce type de modélisation est très courante car elle permet de simplifier la présentation du problème et par conséquent sa résolution en programmation (voir exemple dans le Tableau 1.4).

Soit un problème d'ordonnancement [30], [103] composé de 7 tâches $i \in \{1, 2, 3, 4, 5, 6, 7\}$. Le but est de :

- Calculer les dates de début d'exécution t_i de chaque tâche i,
- Minimiser la fonction objectif makespan C_{max},
- Satisfaire l'ensemble des contraintes données dans le Tableau 1.4, tel que les contraintes de précédence (l'ordre de passage des tâches/opérations), des tâches (temps d'exécution de chaque tâche/opération), et des ressources (nombre de tâches/opérations à faire passer simultanément).

Tableau 1.4. Exemple de modélisation mathématique

Contraintes de données	**Contraintes de précédence**	**Contraintes de ressources**
$p_1 = 3$	$t_1 + p_1 \leq t_3$	$(t_1 + p_1 \leq t_2)$ ou $(t_2 + p_2 \leq t_1)$
$p_2 = 2$	-	-
$p_3 = 4$	$t_1 + p_1 \leq t_4$	$(t_3 + p_3 \leq t_5)$ ou $(t_5 + p_5 \leq t_3)$
$p_4 = 1$	-	$(t_3 + p_3 \leq t_7)$ ou $(t_7 + p_7 \leq t_3)$
$p_5 = 8$	$t_3 + p_3 \leq t_6$	$(t_7 + p_7 \leq t_3)$ ou $(t_5 + p_5 \leq t_7)$
$p_6 = 3$	$t_4 + p_4 \leq t_7$	-
$p_7 = 7$	$t_2 + p_2 \leq t_5$	$(t_4 + p_4 \leq t_6)$ ou $(t_6 + p_6 \leq t_4)$

En plus de sa simplicité, cette modélisation peut aussi être directement exploitée par les méthodes de résolution [103].

1.6.3. Notation

Une méthode de notation qui est toujours d'actualité a été proposée par Graham et al. [105] qui permet de représenter un problème d'ordonnancement d'une manière simple et intuitive. Cette notation est composée de trois champs α, β, et γ, et est présentée sous la forme $(\alpha \mid \beta \mid \gamma)$.

1.6.3.1. Le champ α

Le champ α spécifie le type de l'atelier ou le type de machines ainsi que leur nombre. Il est à son tour composé de trois éléments : α_1, α_2, et α_3. α_1 peut prendre l'une des valeurs suivantes $\{o, P, Q, R, O, F, J\}$.

- $\alpha_1 = o$: l'atelier ne possède qu'une seule machine et par conséquent, les tâches (jobs) ne sont composées que d'une seule opération avec un temps d'exécution noté p_i,

- $\alpha_1 = P$: l'atelier est composé de plusieurs machines parallèles et identiques. Le temps d'exécution d'une tâche est similaire sur toutes les machines,
- $\alpha_1 = Q$: l'atelier est composé de machines parallèles uniformes. Chaque machine possède sa propre vitesse b_i (peut aussi être notée s_i). Le temps d'exécution des tâches est par conséquent redéfini. Le temps d'exécution d'une tâche j sur une machine M_i est définit alors par : $P_{ij} = p_j/b_i$,
- $\alpha_1 = R$: l'atelier est composé de machines parallèles et le temps d'exécution nécessaire pour achever une tâche varie d'une machine à une autre,
- $\alpha_1 = O$: atelier de type open shop,
- $\alpha_1 = F$: atelier de type Flow Shop,
- $\alpha_1 = J$: atelier de type Job Shop.

- $\alpha_2 \in \mathbb{Z}+$: définit le nombre de machines.

- $\alpha_3 = o$: le nombre de machines m est variable.

1.6.3.2. Le champ β

Dans ce champ est défini les contraintes liées aux tâches et aux machines comme par exemple la contrainte de précédence entre tâches. Le champ est composé de 5 sous-champs, à savoir : $\beta_1, \beta_2, \beta_3, \beta_4, \beta_5$.

- $\beta_1 = pmtn$: la préemption entre tâches est tolérée, c'est-à-dire que l'exécution d'une tâche peut être interrompue et reprise plus tard,
- $\beta_1 = o$: pas de préemption,
- $\beta_1 = split$: les tâches peuvent être décomposées en sous-tâches et donc être exécutées en parallèle.

- $\beta_2 = Prec$: les opérations des tâches doivent respecter la relation de précédence,
- $\beta_2 = Tree$: la relation de précédence entre opérations est sous forme d'arbre,
- $\beta_2 = o$: la précédence n'est pas prise en considération.

- $\beta_3 = r_i$: chaque tâche i possède une date d'arrvée,
- $\beta_3 = o$: toutes les tâches arrivent à l'instant $t = 0$.

- $\beta_4 = p_i = 1$: si $\alpha_1 \epsilon \{o, P, Q\}$, toutes les tâches possèdent le même temps d'exécution égal à 1,

- $\beta_4 = p_{ij} = 1$: si $\alpha_1 \epsilon \{O, F, J\}$, toutes les opérations possèdent le même temps d'exécution égal à 1. Le temps d'exécution de la tâche est défini alors par $\sum p_{ij} = N_i$ où N_i est le nombre total d'opérations requises pour achever la tâche i,
- $\beta_4 = o$: chaque tâche/opération possède sa propre durée d'exécution supérieure ou égale à 1.

- $\beta_5 = S_{nsd}$: les machines requièrent un temps de montage indépendant des séquences,
- $\beta_5 = R_{nsd}$: les machines requièrent un temps de démontage indépendant des séquences,
- $\beta_5 = R_{sd}$: les machines requièrent un temps de démontage dépendant des séquences,
- $\beta_5 = b_{j,j+1}$: il existe une limite de stockage entre les machines M_j et M_{j+1}.

1.6.3.3. Le champ γ

Ce dernier champ indique la fonction objectif utilisée. Parmi les critères $C_{max}, L_{max}, T_{max}, E_{max}$, …. (cf. Section 1.2.4 sur les objectifs).

Une représentation légèrement différente de celle proposée par Graham et al. [105] existe aussi dans la littérature et est suggérée par Garey & Johnson [101]. Elle est aussi composée de trois champs $(a \mid b \mid c)$:

- Dans le champ a on décrit le type d'atelier, le nombre de machines, ainsi que le nombre de tâches (jobs) à réaliser,
- Les champ b et c spécifie respectivement les contraintes et la fonction objectif comme les champs β et γ introduits par Graham et al. [105],

Exemple : soit un problème d'ordonnancement Job Shop composé de 40 tâches et 4 machines avec prise en considération des contraintes de précédence, de contraintes sur les dates d'arrivées, et les dates d'échéance. La fonction objectif est de minimiser la valeur du makespan.

Les champs a, b et c sont alors définis ainsi :

- Le champ $a = J, 40, 4$. J pour le type du problème Job Shop, 40 pour le nombre de tâches et 4 le nombre de machines,
- Le champ $b = Prec, r_i, d_i$. $Prec$ est la contrainte de précédence, r_i pour les dates d'arrivée des jobs (release date) et d_i pour les dates d'échéance (due date),
- Enfin, le champ c contient la ou les critères d'optimisation. Dans cet exemple il est défini par C_{max} (makespan).

La représentation de ce problème est ainsi donnée par l'Équation 1.5 :

$$J, 40{,}4 \mid Prec, r_i, d_i \mid C_{max} \tag{1.5}$$

1.6.4. Représentation des solutions

Afin de visualiser la solution d'un problème d'ordonnancement, on utilise généralement le « *diagramme de Gantt* ». Ce diagramme permet de présenter l'ordre d'exécution des tâches avec leurs dates de début de dates de fin sur les machines de l'atelier tout au long de la production. Il est possible d'obtenir deux représentations par diagramme de Gantt. L'une par *produits* et l'autre par *machines* comme illustré dans la Figure 1.10 et la Figure 1.11 avec une solution d'un problème d'ordonnancement composé de 3 tâches et de 2 machines.

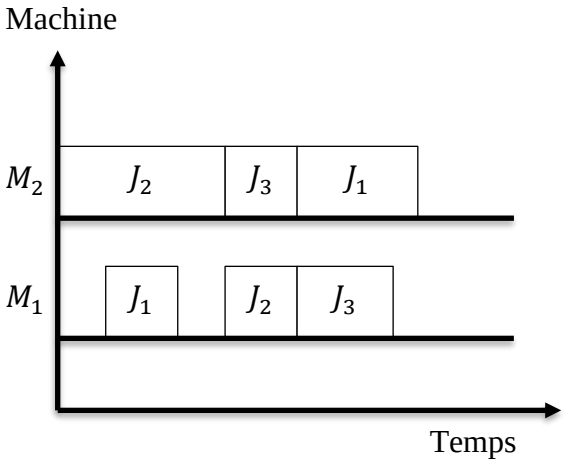

Figure 1.10. Exemple d'un diagramme de Gantt par Machines

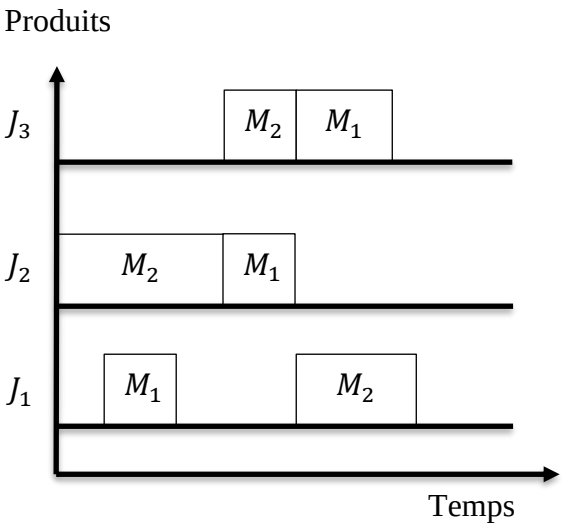

Figure 1.11. Exemple d'un diagramme de Gantt par Produits

1.7. Méthodes de résolution

Les problèmes d'ordonnancement et les problèmes d'optimisation en général sont en majorité très difficiles à résoudre dû au nombre de solutions possibles. Dans la littérature plusieurs méthodes sont proposées [62], [68], [81], [86], [106] pour la résolution de problèmes d'ordonnancement, comme le Job Shop, Flow Shop ou encore le Single Machine. Ces méthodes sont groupées en deux classes : les « *méthodes exactes* » qui offrent la possibilité de trouver des solutions optimales pour des problèmes de petite taille, et les « *méthodes approchées* » qui sont utilisées pour résoudre des problèmes de grande taille et permettent de trouver des solutions en un temps raisonnable [107].

1.7.1. Les méthodes exactes

En faisant recours aux méthodes exactes pour la résolution des problèmes d'ordonnancement de petite taille où le nombre de combinaisons est faible, il devient possible de trouver des solutions optimales [31]. Cette optimalité peut être mesurée en fonction des critères d'optimisation définis comme le C_{max}. Parmi les méthodes exactes les plus sollicitées on trouve le fameux « *Branch & Bound* » (*B&B*), en français « *méthode de séparation et évaluation* » [108]. D'autres méthodes existent aussi comme la « *programmation dynamique* » ou la « *programmation linéaire* ». Ces méthodes explorent l'espace de recherche d'une manière implicite afin de retourner une solution. C'est particulièrement pour cette raison que ces méthodes ne sont utilisées que pour des problèmes de petite taille.

1.7.1.1. La méthode Branch & Bound

La méthode B&B affecte progressivement les ressources disponibles aux tâches en attente tout en explorant un arbre de recherche composé de l'ensemble des combinaisons possibles. Chaque sommet de l'arbre représente un sous-problème et les arcs issus d'un sommet représentent les décompositions possible du problème initial ou des sous-problèmes [28], [29]. Lors de l'exploration de l'arbre, les branches qui décrivent des solutions non-réalisables et celles qui conduisent éventuelles à des pires solutions que celles déjà connues sont ignorées (*cut*). Cette exploration permet en réalité d'éviter d'explorer l'ensemble des solutions possibles. Le B&B possède quatre principales fonctionnalités qui ont :

- La *séparation* : qui consiste à décomposer un problème en un ensemble de sous-problèmes de taille réduite,
- L'*évaluation* : permet d'affecter une borne minimale en termes de la fonction objectif à chaque sous-problème (chaque sommet),
- Le *sondage* : a pour but d'identifier si un sommet est une solution inadmissible, une solution optimale, ou peut conduire à une meilleure solution et donc s'il mérite d'être séparé à son tour ou non,
- La *sélection* : consiste à choisir parmi l'ensemble des sous-problèmes à explorer un seul. On distingue deux stratégies d'exploration, la première appelée *séparation et évaluation progressive* qui favorisent le sous-problème à fort potentiel. La deuxième appelée

séparation et évaluation séquentielle qui parcourt les sommets (sous-problèmes) en profondeur.

1.7.1.2. La programmation dynamique

Cette méthode se base sur le principe de Bellman [109] qui a comme principe : « Si C est un point qui appartient au chemin optimal entre A et B, alors la portion de ce même chemin allant de A à C est le chemin optimal entre A et C » [28]. Autrement dit, « toute politique optimale est composée de sous-politiques optimales » [29]. Le but est de trouver une solution optimale pour tous les sous-problèmes d'une manière ascendante afin d'atteindre une solution optimale pour le problème initial. Cette méthode est destinée à la résolution de problèmes d'optimisation à vocation plus générale que la méthode Branch & Bound, mais elle ne permet pas de traiter de problèmes complexes ou de grande taille [28], [29], [110].

1.7.1.3. La programmation linéaire

La programmation linéaire est une des branches de l'optimisation, elle permet de résoudre de nombreux problèmes économiques et industriels et plus spécifiquement les problèmes d'ordonnancement. La méthode consiste à trouver une solution de façon à minimiser une fonction de coût Z (fonction objectif) tout en respectant l'ensemble des contraintes, comme les contraintes de précédence. Le programme linéaire est composé de n variables non négatives (Équation 1.8), m contraintes d'égalité ou d'inégalité (Équation 1.7) ainsi que de la fonction objectif (Équation 1.6) (ne pas confondre avec n et m pour le nombre de tâches et machines).

$$\max ou \min Z = \sum_{j=1}^{n} c_j x_j \tag{1.6}$$

$$\forall\, i = 1 \dots m : \ \sum_{j=1}^{n} a_{ij} x_j \leq ou \geq \ b_i \tag{1.7}$$

$$\forall\, j = 1 \dots n : \ x_j \geq 0 \tag{1.8}$$

Où c_j correspond au coût de profit de la variable x_j, a_{ij} celui de la contrainte i, et b_i est une constante.

On parle de « *Programme Linéaire en Nombre Entiers* » (*PLNE*) lorsque les variables sont entières et de programme linéaire en 0-1 lorsque les variables ne peuvent prendre que ces deux valeurs. Dans ce cas les valeurs sont appelées « *booléennes* », « *binaires* » ou de « *décision* » [29]. Une autre variante existe aussi appelée « *Programme Non Linéaire* » (*PNL*) lorsque les contraintes ou la fonction objectif ne sont pas une combinaison linéaire.

Pour conclure avec les méthodes exactes, il faut noter que bien qu'elles permettent de trouver des solutions optimales elles ne peuvent être utilisées pour résoudre des problèmes de grande taille ou de problèmes difficile comme le Job Shop [63], [70], [106].

1.7.2. Les méthodes approchées

Le temps d'exécution élevé requis par les méthodes exactes pour la résolution de problèmes d'ordonnancement a poussé les chercheurs vers les méthodes approchées, ou « *approximate methods* », ou encore, « *heuristiques/méta-heuristiques* ». Ceci malgré l'apparition de nouveaux calculateurs ultra-performants et le progrès continuel de ce type de machines. Les

méthodes approchées sont développées afin d'exploiter au mieux la structure du problème permettant ainsi d'explorer intelligemment l'espace de recherche [111]. La résolution en se basant sur ces méthodes permet de trouver un séquencement des tâches de bonne qualité en un temps CPU raisonnable mais ne garantit pas l'optimalité. Harrath [29] énumère un certain nombre d'avantages offerts par les méthodes approchées comparées aux méthodes exactes :

- Trouver une solution optimale avec une méthode exacte peut s'avérer impossible pour certains types problèmes dû à la dynamique de l'environnement (par exemple l'arrivée de nouvelles tâches, ou les pannes machine, …), le nombre élevé de variables (nombre de tâches et/ou machines, …), l'optimisation multi-objectifs, ...,
- Le temps d'exécution nécessaire est plus court comparé aux méthodes exactes et permet dans certains cas de fournir plusieurs solutions de bonne qualité,
- Les méthodes approchées sont généralement très simples et intuitives, et peuvent être facilement comprises, implémentées, et utilisables par des utilisateurs non expérimentés,
- Il est possible de combiner plusieurs de ces méthodes afin de maximiser leur efficacité et d'améliorer les solutions trouvées [74].

Les méthodes approchées peuvent être classées en quatre catégories selon Boukef Ben Othman [28], à savoir, les « *méthodes constructives* » (heuristiques), les « *méta-heuristiques basées sur la recherche locale* », les « *méta-heuristiques basées sur les algorithmes évolutionnaires* », et les méthode de « *recherche globale* ».

1.7.2.1. Les méthodes constructives

Dans ce genre de méthodes, la solution est vide initialement est on y insère graduellement des opérations (ou tâches) jusqu'à obtenir la solution finale. Ces heuristiques, appelées « *Règles de Priorité* » (« *Dispatching Rule* » ou « *Priority Dispatching Rules* » en anglais), sont des méthodes empiriques qui donnent généralement de bons résultats sans pour autant être démontrables [30]. Elles se basent sur des stratégies de décision dans le but de construire une solution proche de la solution optimale tout en maintenant le temps calcul nécessaire au strict minimum. Dans la littérature il existe beaucoup de règles de priorité, les plus connues sont [16], [111], [112] :

- FIFO (First In First Out) : la première tâche arrivée est la première à être exécutée,
- LIFO (Last In First Out) : contrairement à FIFO, la dernière tâche arrivée est exécutée en premier,
- SPT (Shortest Processing Time) : la tâche avec le plus petit temps d'exécution est traitée en premier,
- LPT (Longest Processing Time) : la priorité est donnée à la tâche avec le temps d'exécution le plus important,
- EDD (Earliest Due Date) : la tâche avec la date d'échéance la plus proche est traitée en premier,
- LLF (Last Laxity First) : donne la priorité à la tâche dont la distance entre la fin de son exécution et son délai est le plus court,

- HPF (High Priority First) : si les tâches possèdent une priorité, celle avec la plus grande est traitée en premier,
- SB (Shifting Bottleneck) : la méthode est utilisée pour le Job Shop, où le problème est découpé en plusieurs sous problèmes (problèmes d'ordonnancement à une machine). La machine ayant la plus grande valeur de C_{max} où un goulot d'étranglement existe (bottleneck machine) est sélectionnée en premier et sa file d'attente est optimisée. Le processus est répété jusqu'à ce que toutes les machines aient été traitées.

1.7.2.2. Les méthodes de recherche locale

Ces méthodes sont des algorithmes itératifs qui explorent l'espace de recherche en partant d'une solution faisable choisie aléatoirement ou à l'aide d'une heuristique. La solution initiale est modifiée progressivement dans le but de l'améliorer (voir Figure 1.12). Pour se faire, il est impératif de définir un voisinage pour chaque solution ce qui permet de passer d'une solution à une autre solution voisine qui diffère de la première et qui améliore la fonction objectif. Le processus se répète jusqu'à ce qu'un critère d'arrêt soit atteint ou un optimum local soit rencontré.

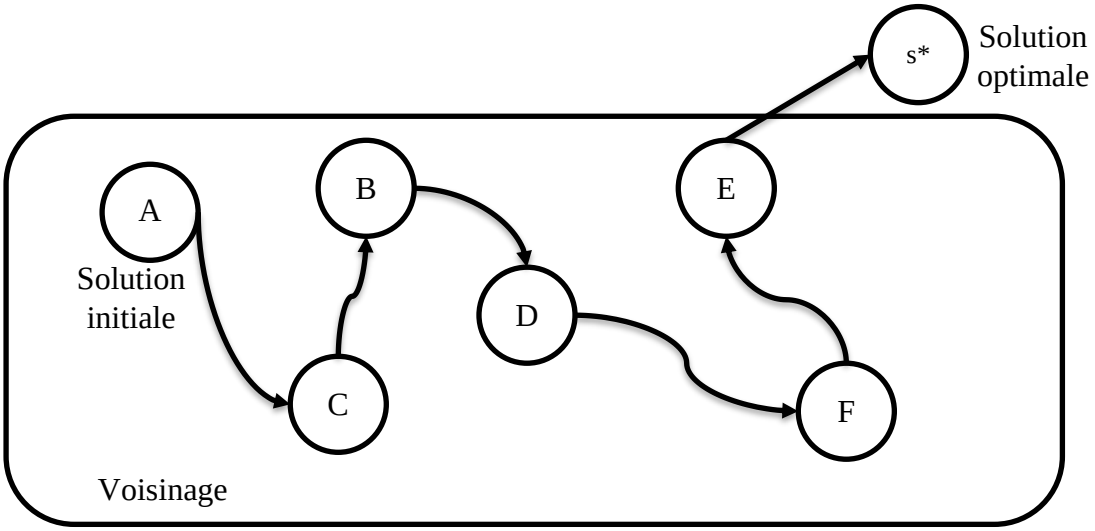

Figure 1.12. Principe de la recherche locale

L'un des inconvénients majeurs de ce type de méthodes réside dans le fait que l'optimum local trouvé par ces solutions est souvent de mauvaise qualité [28]. D'où la nécessité d'utiliser des méthodes basées sur la recherche locale à fort potentiel de transformation (mouvement d'une solution à une autre) [77], [113].

- **La méthode de descente**

Cette méthode explore l'espace de solutions en choisissant à chaque fois le meilleur voisin de la solution courante [29]. Le processus continue jusqu'à ce qu'il ne soit plus possible d'améliorer la valeur de la fonction objectif. C'est-à-dire qu'un minimum local a été atteint (voir Figure 1.13). Cette méthode comporte une grosse lacune qui réside dans le fait qu'en fonction de la solution initiale choisie, la recherche va s'arrêter au premier minimum local

atteint. D'où l'apparition de méthodes plus efficaces comme le recuit simulé et la recherche tabou.

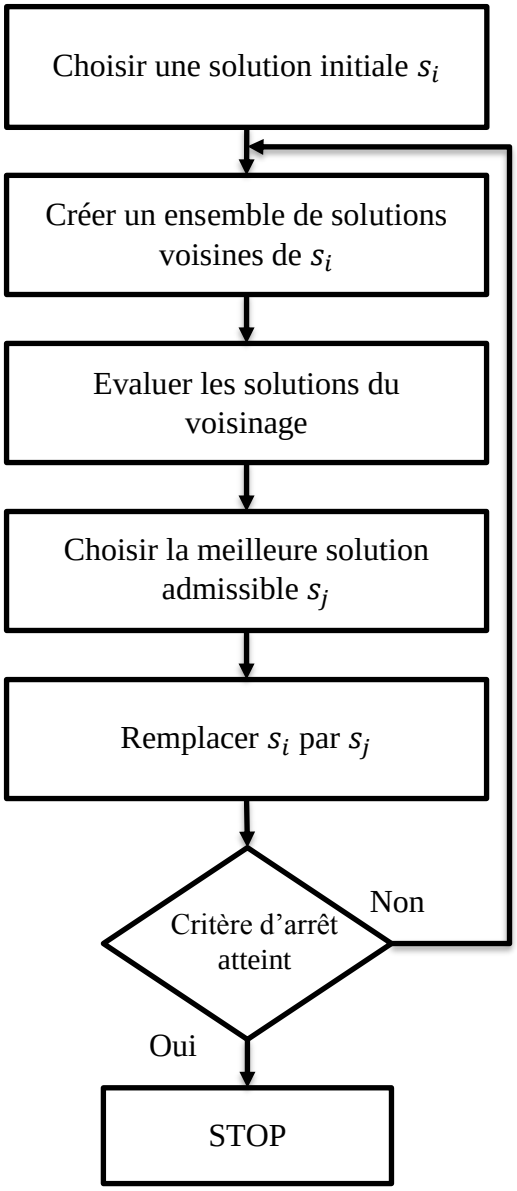

Figure 1.13. Principe de fonctionnement de la méthode descente

- **Recuit simulé**

Le recuit simulé est une méthode basée sur la recherche locale inspirée du recuit simulé physique utilisé en métallurgie [28]. Ce dernier est un processus physique de chauffage. On commence par chauffer un métal solide qui devient ensuite liquide après avoir atteint une certaine température et par conséquent les atomes qui le composent voient leurs degrés de liberté augmenter. Et inversement, en baissant la température, le degré de liberté commence alors à diminuer. En baissant suffisamment la température, le métal redevient alors solide. En fonction de la façon utilisée pour baisser la température, on peut obtenir deux types de solides :

- Une baisse rapide de la température produit une structure amorphe, tel le verre où les atomes ne respecte aucune structuration aucun ordre (minimum local d'énergie),
- Une baisse lente de la température produit une masse où les atomes sont structurés tel un cristal (minimum global d'énergie). Lorsque cette baisse est trop rapide on peut avoir des défauts au niveau du cristal, le *recuit* permet alors de redonner la liberté aux atomes afin d'atteindre un nouvel état dynamique.

Le recuit simulé introduit par Kirkpatrick et al. [114], démarre par une solution initiale aléatoire, ou obtenue par heuristique, et explore l'espace de solutions en effectuant des modifications mineures à la solution initiale. Si après modification, la nouvelle solution obtenue est meilleure que la précédente elle est retenue. Si ce n'est pas le cas, elle peut être retenue suivant la probabilité P (Équation 1.9) où ΔE représente la détérioration de la nouvelle solution et T un paramètre inversement proportionnel au nombre d'itérations [28], [29] (voir Figure 1.14).

$$P = \exp(-\Delta E/T) \tag{1.9}$$

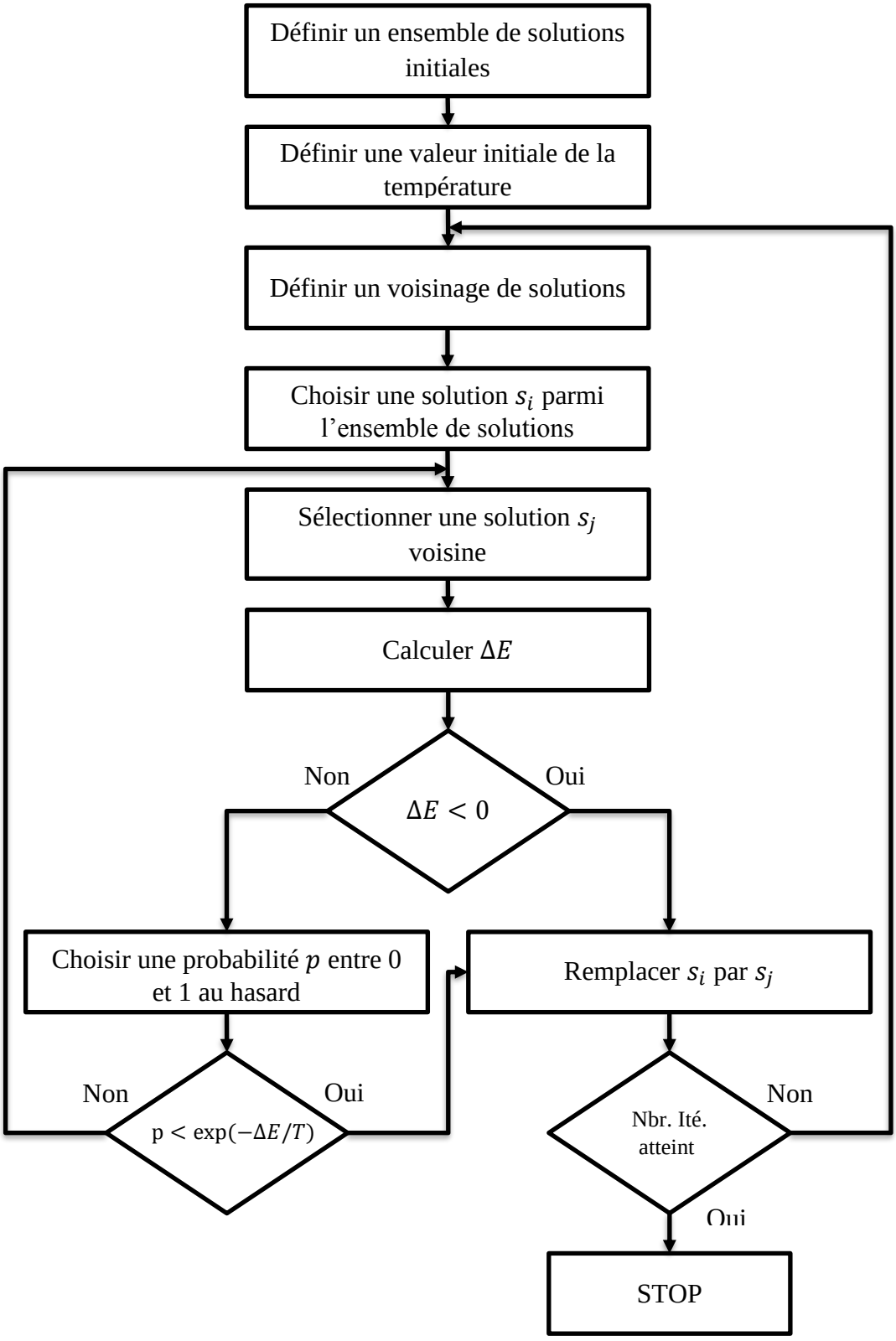

Figure 1.14. Principe de fonctionnement du recuit simulé

De nombreuses approches basées sur le recuit simulé ont vu le jour [58], [113] car elle offre de nombreux avantages comme la possibilité (probabilité) de changer la solution actuelle par une solution moins performante permettant d'atteindre plus de solutions de l'espace de recherche

et d'éviter de converger prématurément vers un optimum local. Cette méthode peut aussi être utilisable pour la résolution de problèmes d'optimisation combinatoire et obtient des résultats satisfaisant grâce à l'exploration de l'espace de solutions.

- **Recherche tabou**

La recherche tabou a fait son apparition dans les années 1970 et ne fut utilisée qu'en 1986 par Glover [115], comme les méthodes basées sur la recherche locale, elle a pour but d'améliorer la fonction objectif en parcourant l'espace de solutions. Elle est basée sur deux principes, le premier consiste à améliorer la solution en choisissant la meilleure solution voisine. Si aucune des solutions voisine n'améliore la fonction objectif, alors la moins mauvaise est sélectionnée. Le second principe a pour but de mémoriser les dernières solutions visitées et les mettre sur une « *liste taboue* » afin d'interdire de futures visites de ces mêmes solutions [28] (voir Figure 1.15).

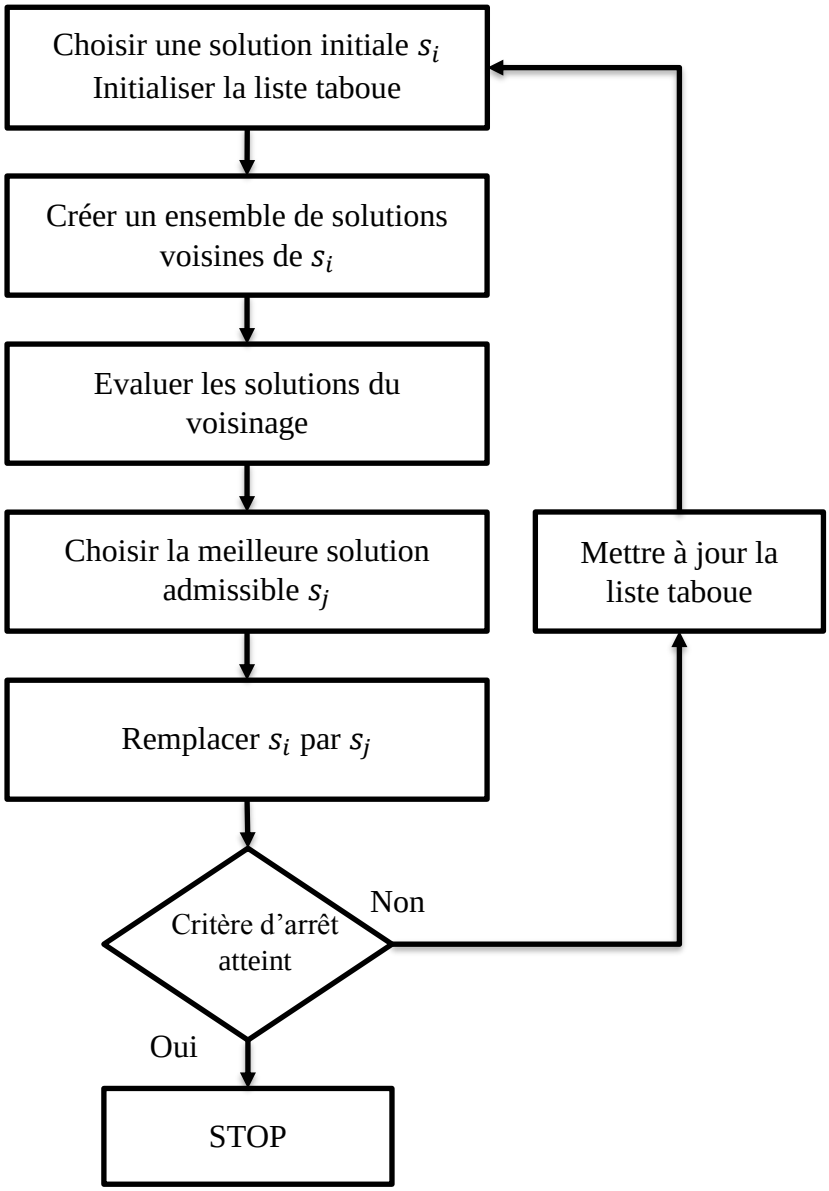

Figure 1.15. Principe de fonctionnement de la recherche tabou

La méthode présente plusieurs avantages, parmi lesquels on cite :

- En cas d'absence de meilleure solution dans le voisinage, elle sélectionne une solution moins performante et évite ainsi de rester bloquée dans un optimum local,
- Interdit l'accès aux dernières solutions visitées,
- Elle offre la possibilité de résoudre des problèmes NP-difficiles [24], [77].

1.7.2.3. Les méthodes évolutives

Ces méthodes s'inspirent des principes de l'évolution, elles ont pour but de faire évoluer un ensemble de solutions appelé « *population* », ce qui n'est pas le cas des méthodes basées sur la recherche locale. Cette évolution permet ainsi d'atteindre des nouvelles solutions, basées sur celles présentes dans la population précédente, afin d'optimiser une fonction prédéfinie.

Parmi les méthode évolutives, l'*Algorithme Génétique* (*AG*) est le plus sollicité [46], [48], [70], [80], [86], [116]. Il a été proposé pour la résolution de problèmes combinatoire en 1975 par Holland [117], son but de est d'optimiser une fonction objectif appelée « *fitness* ». La population est composée d'*individus* (ou *chromosomes*), et chaque individu est un ensemble d'éléments fondamentaux (ou *valeurs*) appelés « *gènes* ». Les individus correspondent en réalité à des solutions d'un problème d'optimisation, et les gènes sont les valeurs ou les attributs de cette solution.

L'algorithme génétique passe par cinq étapes, comme indiqué dans la Figure 1.16 :

- Le codage : a pour but de représenter l'ensemble des solutions de l'espace de recherche, et associe ainsi un individu à chaque solution. Une mauvaise représentation qui ne peut couvrir l'ensemble des solutions existantes implique que l'AG ne peut atteindre ces solutions. On distingue deux types de codage, le codage binaire (à valeurs binaires) et le codage réel,
- La génération de la population initiale : cette fonctionnalité doit pouvoir générer aléatoirement ou en utilisant une heuristique ou méta-heuristique un ensemble d'individus non homogènes,
- La sélection : son but est de choisir les individus qui vont se reproduire, à l'aide des opérateurs de croisement ou de mutation, et transmettre ainsi leurs gènes,
- Le croisement et la mutation : le croisement sélectionne et combine les gènes des individus parents afin de créer de nouveaux individus fils. La mutation change aléatoirement un gène d'un individu, son principal but est d'éviter de converger trop rapidement vers un optimum local,
- Le remplacement : une fois les nouveaux individus créés par croisement et/ou mutation, il est indispensable de décider de la façon d'intégrer ces nouveaux individus et s'ils doivent ou non remplacer des individus de la population précédente.

D'autres paramètres de l'algorithme génétique doivent être spécifiés, comme la taille de la population, le nombre total de générations, les critères d'arrêt, et les probabilités de croisement et de mutation.

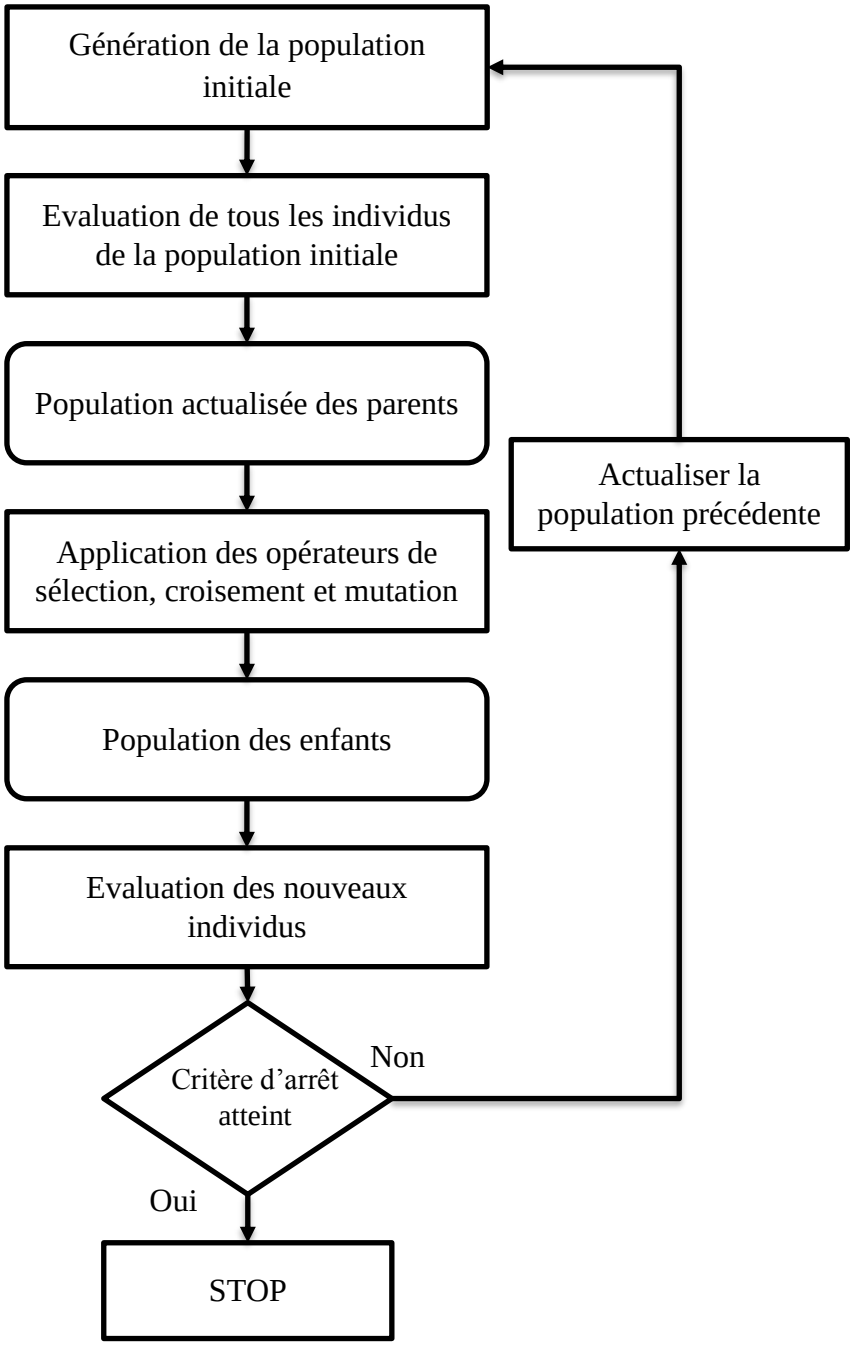

Figure 1.16. Fonctionnement de l'algorithme génétique

1.7.2.4. Les méthodes de recherche globale

- **L'algorithme de colonies de fourmis**

Cet algorithme est inspiré du comportement des fourmis, qui sont capable de trouver le plus court chemin pour aller de leur nid à une source de nourriture et de s'adapter aux changements de l'environnement grâce à la phéromone. La phéromone est substance laissée par les fourmis sur leur chemin de retour d'une source de nourriture qui permet de laisser une trace de ce chemin. Cet algorithme est proposé initialement par Colorni et al. [118] pour la résolution des problèmes NP-difficiles, et est souvent utilisé pour les problèmes d'ordonnancement [63], [83], [116]. Le

principe de l'algorithme de colonies de fourmis [28] est illustré dans la Figure 1.17, la Figure 1.18, et la Figure 1.19, et où la nourriture représente une solution dans le cas d'un problème d'optimisation. L'algorithme général des colonies de fourmis et décrit comme suit :

Tant que la condition d'arrêt n'est pas atteinte **faire**

Initialiser la colonie de fourmis

Initialiser la phéromone

Trouver le chemin de chaque fourmi

Mettre à jour la phéromone

Retenir la meilleure solution

Fin Tant que

Dans la Figure 1.17, une source de nourriture est posée à proximité de la fourmilière, les fourmis empruntent alors le chemin le plus direct pour l'atteindre.

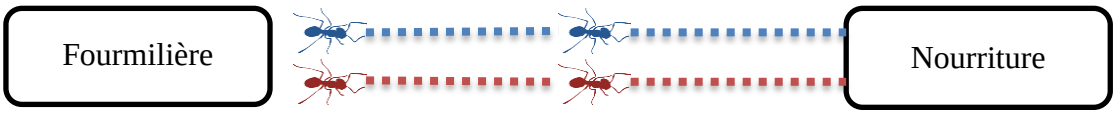

Figure 1.17. Déplacement des fourmis de la fourmilière vers une source de nourriture

Dans la Figure 1.18, un obstacle est posé entre la fourmilière et la nourriture. Les fourmis commencent alors à contourner cet obstacle afin d'atteindre la source de nourriture, à leur retour elles déposent de la phéromone afin de marquer leur passage. La quantité de phéromones, ou le nombre de fourmis, est plus importante sur le chemin le plus court i.e. la partie supérieure de l'obstacle. Cette quantité importante de phéromones va alors attirer la majorité des fourmis vers ce chemin comme montré dans la Figure 1.19.

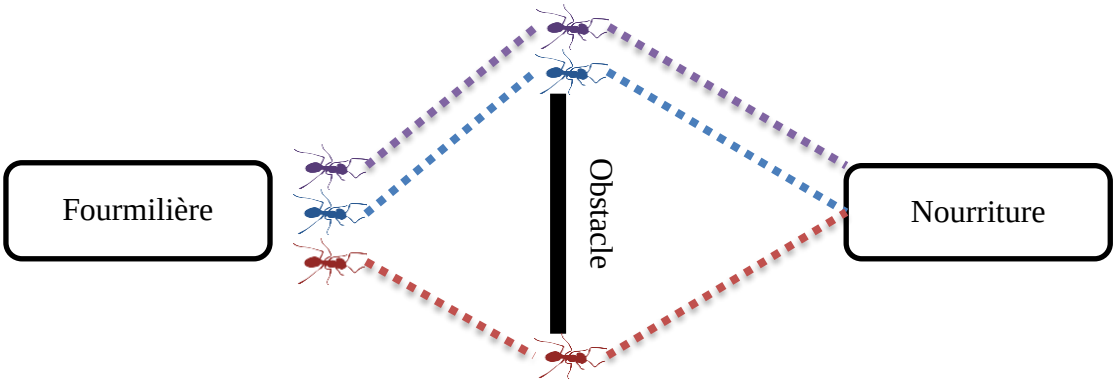

Figure 1.18. Déplacement des fourmis vers une source de nourriture avec un obstacle

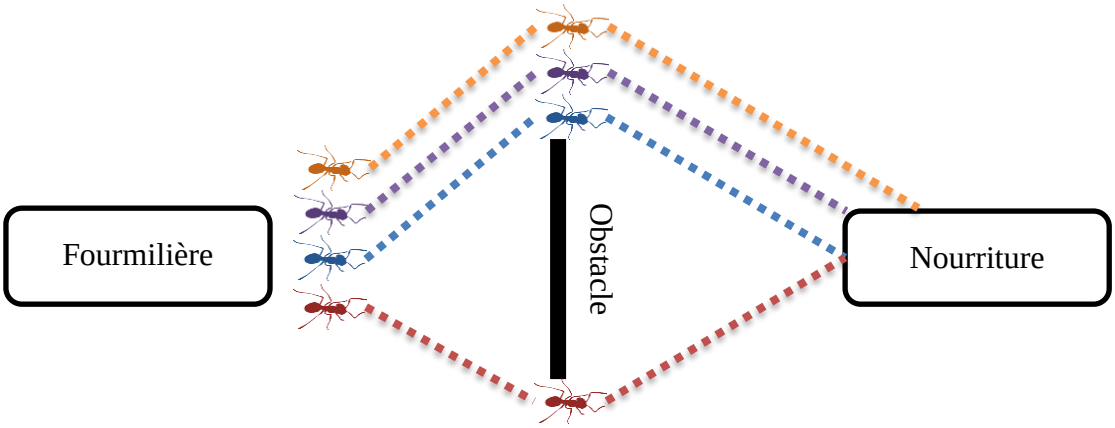

Figure 1.19. Choix du plus court chemin par la majorité des fourmis

- **L'algorithme d'optimisation par essaim particulaire**

L'optimisation par essaim particulaire, ou « *Particle Swarm Optimization* » en anglais, a été introduite par Eberhart & Kennedy [119] en 1995. Elle s'inspire fortement des relations grégaires des oiseaux migrateurs qui doivent parcourir des longues distances et qui doivent donc optimiser leurs déplacements en termes d'énergie dépensée. Cette procédure de recherche est basée sur une population d'individus appelés particules. Ces particules sont placées aléatoirement dans l'espace de recherche de la fonction objectif. A chaque itération [120] les particules se déplacent dans l'espace de recherche (voir Figure 1.20) tout en prenant en compte en considération :

- Leur meilleure position, en se basant sur leur expérience,
- La meilleure position d'un certain nombre de leurs voisins en les interrogeant.

La particule a aussi la possibilité de se déplacer au hasard ou en utilisant une méthode de recherche locale ou de recherche globale.

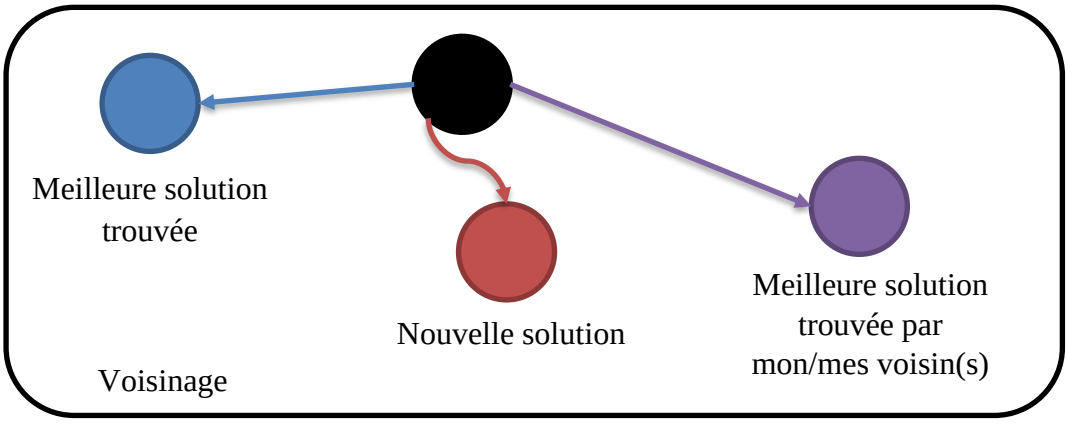

Figure 1.20. Déplacements possibles d'une particule dans l'espace de recherche

L'algorithme général de l'essaim particulaire est donné comme suit :

Initialiser les particules

Tant que la condition d'arrêt n'est pas atteinte **faire**

Mettre à jour la position et la vitesse de chaque particule

Evaluer chaque particule

Retenir la meilleure solution

Fin Tant que

Le déplacement des particules est similaire à celui de l'être humain qui se base aussi sur sa propre expérience ou celle de son entourage pour la prise de décision [119].

1.8. Conclusion

Dans ce chapitre nous avons présenté les concepts de base des problèmes d'ordonnancement, les types d'ateliers, les annotations et représentations possibles, les méthodes de résolution utilisées, ainsi qu'un état de l'art avec un intérêt particulier pour les ateliers à Machine Unique et les ateliers à cheminement multiple. Les méthodes de résolution groupées en deux familles, exactes et approchées, présentent des avantages et des inconvénients en fonction des problèmes à résoudre, de leurs types, et de leurs complexités. Cependant ces méthodes restent limitées face à des problèmes dynamiques et à très forte complexité d'où la nécessité de les combiner entre elles (et elles sont alors connues sous le nom de méthodes « *hybrides* »), ou de les assembler à des techniques issues d'autres domaines.

Dans le prochain chapitre nous nous intéresserons aux règles de priorité qui font partie des méthodes « *heuristiques* » et qui ont fait l'objet d'un grand intérêt de la part des chercheurs dans le domaine de l'ordonnancement en raison de leur simplicité, efficacité, faible temps d'exécution mais aussi en raison de leur responsivité dans les ateliers de production dynamiques.

CHAPITRE

2. Les Règles de Priorité : un Etat de l'Art

Résumé : Les règles de priorité font partie des méthodes les plus sollicitées pour la résolution de problèmes d'ordonnancement. Ceci est notamment dû à leur simplicité et leur rapidité d'exécution car elles reposent sur de simples tests d'attributs des tâches et/ou machines. Chaque machine de l'atelier de production est chargée d'opérer la tâche avec la priorité la plus élevée/faible se trouvant en tête de la file d'attente, évitant ainsi que la machine reste inactive alors qu'il existe au moins une tâche qui la requière. Les règles de priorité sont notamment très utilisées, en raison de leurs points forts, pour la résolution de problèmes d'ordonnancement NP-Difficile, ou dans un environnement dynamique où l'on ne maitrise pas l'arrivée des tâches, les changements de priorité, ou encore d'autres évènements imprévus telles les pannes machines.

Mots-clés : Règle de Priorité · Classification des Règles · Facteurs de Performances

2.1. Introduction

Pour la résolution des problèmes d'optimisation ou, plus spécifiquement, des problèmes d'ordonnancement, une multitude d'approches existent dans la littérature, allant des méthodes exactes telles que la programmation linéaire ou la méthode d'évaluation et de séparation jusqu'aux méta-heuristiques comme les algorithmes génétique ou la recherche tabou [107].

Si les méthodes exactes sont capables de trouver des solutions pour des problèmes statiques ou de petite taille, elles ne peuvent résoudre des problèmes NP-difficiles comme le Job Shop en un temps polynomial. En revanche, les méta-heuristiques permettent de trouver une solution acceptable pour la majorité des problèmes NP-difficiles statiques en un temps d'exécution polynomial [101], [102]. Cependant, Koulamas & Chen [18], [65] affirment que la résolution de problèmes de très grande taille et/ou à forte dynamique n'est pas faisable par les méta-heuristiques même en utilisant les algorithmes évolutionnaires [9], [19], la recherche tabou [77], ou le recuit simulé [113]. Ce qui a conduit les chercheurs à s'orienter vers des heuristiques connues sous le nom de « *règles de priorité* » (en anglais « *Dispatching Rules* », ou « *Priority Dispatching Rules* », ou encore « *Scheduling Rules* »). Ces techniques sont dédiées à la résolution de problèmes dynamiques de grande taille et à très forte complexité [21].

Une règle de priorité définit une séquence d'exécution pour les tâches sur la file d'attente de chaque machine en affectant une priorité à chacune de ces tâches. Lorsqu'une machine devient inactive, la tâche avec la priorité la plus élevée/basse est lancée, et ainsi de suite jusqu'à ce que la file soit vide. L'affectation de priorité aux tâches à l'aide des règles se fait très rapidement, ce qui permet de renforcer la réactivité du système en cas de perturbation, comme l'arrivée d'une nouvelle tâche ou de pannes machine. En outre, étant donné que ces règles sont des algorithmes simples, elles sont facilement compréhensibles et applicables dans les industries complexes et dans des environnements dynamiques. En plus, elles peuvent être intégrées aisément dans un modèle de simulation afin d'évaluer la qualité de leurs résultats.

L'ordonnancement par règles de priorité est un processus à deux étapes, la première détermine une valeur de priorité k_i, en fonction d'un certain critère défini par chaque règle, pour chaque tâche J_i. La deuxième étape consiste à trier les tâches en fonction de leurs priorités et la tâche la plus prioritaire est ensuite traitée si la machine est libre [121].

Les règles de priorité sont très sollicitées [24], [71], [87], [122]–[124] malgré leur faible performance, ceci en raison de leur simplicité, leur très faible temps de calcul, et la possibilité de les utiliser pour l'ordonnancement dynamique et dans les industries complexes. Ce type de méthodes de résolution devient encore plus efficace pour l'ordonnancement en ligne (en temps-réel) où les dates d'arrivée des tâches ne sont connues qu'au cours de l'exécution et où il est difficile de recourir aux méta-heuristiques.

Pour l'ordonnancement en temps-réel et la gestion des tâches sur les files d'attentes des machines, les règles de priorité sont considérées comme les outils les plus adaptés, car elles permettent de répondre à ces deux aspects d'une manière plus efficace que le reste des méthodes de résolution. Cet état de fait est prouvé par l'engouement des chercheurs pour ces méthodes constaté par le grand nombre de contributions sur le sujet [24], [71], [87], [122]–[124].

Dans ce chapitre, nous présentons les différentes classes des règles de priorité et les facteurs qui affectent leurs performances. Nous présentons aussi un état de l'art aussi exhaustif que possible sur l'utilisation des règles de priorité pour l'ordonnancement, et nous exposons un certain nombre de règles que nous avons jugé intéressantes.

2.2. Classification des règles de priorité

Afin de pouvoir étudier les règles de priorité, leurs comportements, et leurs performances, il est primordial de les classer pour pouvoir les comparer. De nombreuses publications existent portant sur la classification des règles de priorité. Les classification les plus connues sont celle de Panwalker & Iskander [125] basée sur la structure des règles, ou celle de Blackstone et al. [126] basée sur les informations utilisées par les règles lors de l'affectation des priorités.

2.2.1. Classification basée sur la structure

Dans cette classification [125] les règles de priorité sont groupées en trois familles (voir Figure 2.1), « *règles basiques* », « *règles d'ordonnancement heuristiques* », et « *autres règles* » (*basic rules, heuristic scheduling rules* et *other rules*). La famille des règles basiques est à son tour composée de trois sous-familles :

- *Règles simples* : ces règles utilisent les informations sur les tâches, comme le temps d'exécution, ou les dates d'échéance,
- *Règles composées* : ce sont des règles combinées qui sont appliquées à certains groupes de tâche ou en fonction de l'état de l'atelier,
- *Règles pondérées* : ce sont des règles formées de plusieurs règles simples pondérées.

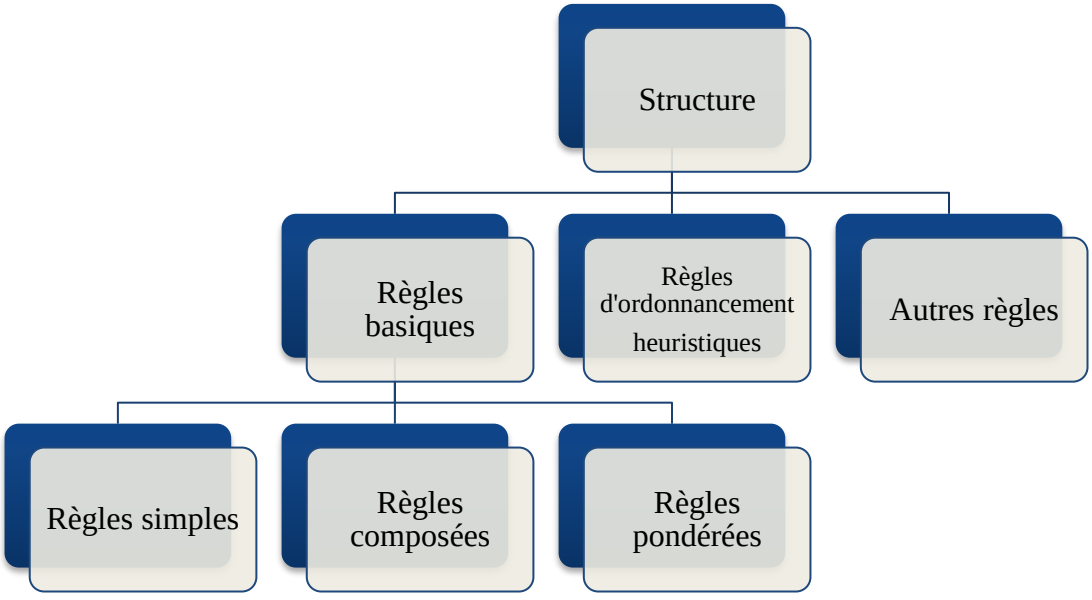

Figure 2.1. Classification basée sur la structure

Les règles de priorité issues de la famille des « *règles d'ordonnancement heuristiques* » considèrent des attributs plus complexes des tâches ou de l'atelier comme par exemple le temps

d'attente prévu sur la prochaine machine mais aussi la possibilité d'insérer des tâches, par inspection visuelle, dans les intervalles de temps où une machine est libre [16], [31].

La famille des « *autres règles* » regroupe les règles ne pouvant pas faire partie des deux premières catégories, comme les de règles de priorité spécifiques à un atelier.

2.2.2. Classification basée sur l'information

Dans cette classification proposée par Blackstone et al. [126] et inspirée de la classification basée sur la structure [125], les règles de priorité sont regroupées en trois familles, « *contenu de l'information* », « *portée* » ou « *dépendance temporelle* » comme illustré dans la Figure 2.2 [82], [127].

2.2.2.1. Classification par contenu de l'information

Regroupe les règles basées sur les informations des tâches comme le temps d'exécution ou les dates d'échéance, ou les informations sur les machines, comme la charge. Dans cette classification on peut trouver par exemple les règles « *Shortest Processing Time* », « *First In First Out* », ou « *Earliest Due Date* »

2.2.2.2. Classification par portée

Les règles de priorité sont classées en fonction de la portée des variables utilisées comme critère pour l'affectation de priorités aux tâches. On parle de « *règle locale* » si la variable utilisée comporte uniquement la ou les informations sur les tâches en attente sur la file d'attente actuelle ou sur les informations de la machine actuelle. Une règle est dite « *globale* » si les informations portent sur les tâches en attente sur d'autres machines ou la charge des autres machines de l'atelier.

Les règles globales sont à leur tour divisées en deux sous-groupes, à savoir, « *règles globales directes* » et « *règles globales indirectes* ». Les règles globales directes sont basées sur l'agrégation de plusieurs attributs du système comme la distribution des temps exécution. Ces règles ne font aucune référence explicite aux attributs des tâches sur les files d'attente, c'est-à-dire que l'information utilisée pour l'affectation de priorité aux tâches est basée sur plusieurs attributs à la fois. Quant aux règles globales directes, elles sont basées sur des informations explicites ou des attributs clairs et précis relatifs aux tâches, comme le temps d'exécution des tâches ou leurs dates d'échéances [31].

2.2.2.3. Classification par dépendance temporelle

Dans cette classification les règles de priorité peuvent être classées dans deux sous-catégories, « *règles statiques* » et « *règles dynamiques* ». Les règles statiques sont basées des informations (attributs des tâches ou des machines) qui donnent la même priorité aux tâches en attente indépendamment du temps, comme les règles FIFO et SPT. La priorité d'une tâche restera la même à l'instant t comme à l'instant $t + c$. Par contre, les règles dynamiques changent les priorités des tâches à chaque instant, comme celles basées sur le temps d'attente des tâches (*wait time*) [17].

Une autre classification existe dans la littérature où les règles de priorité sont groupées en deux familles, à savoir, « *passage unique* » (*single-pass*) et « *passage multiple* » (*multi-pass*) [22]. La majorité des règles existantes sont issues de la famille à passage unique où les priorités sont affectées aux tâches en un temps, c'est le cas des règles EDD et SPT par exemple. Les règles à passage multiple affectent aussi des priorités aux tâches dans un premier temps et réitère le processus plusieurs fois afin d'améliorer la solution.

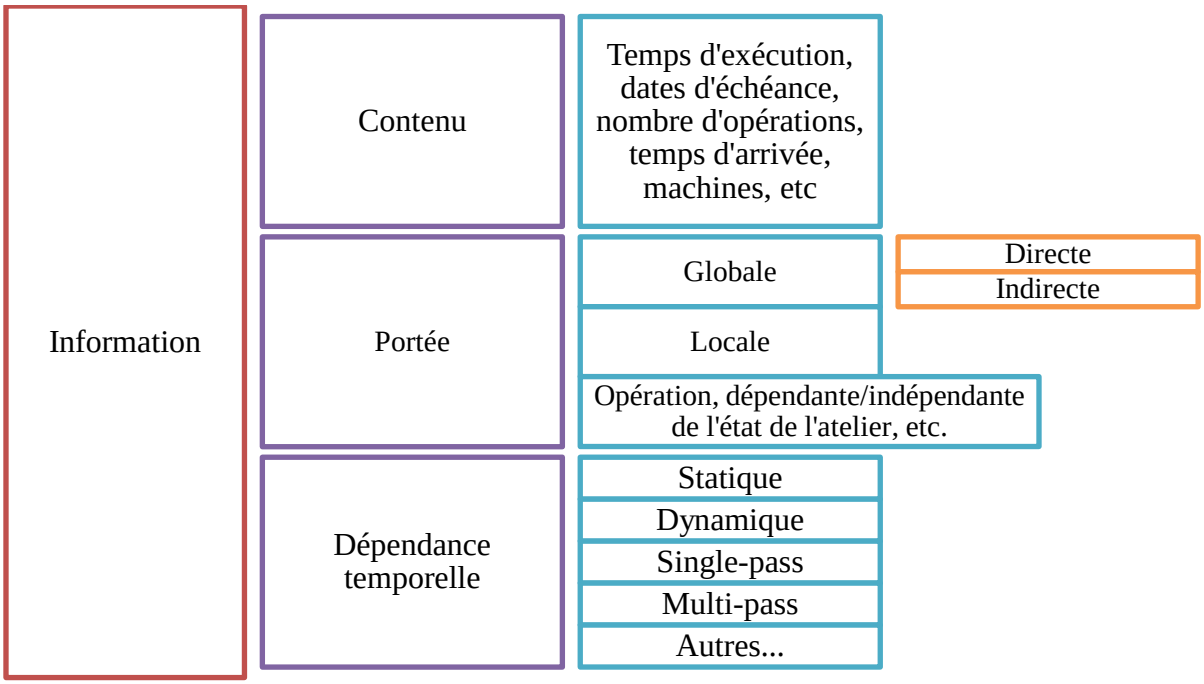

Figure 2.2. Classification basée sur l'information

Kemppainen [127] propose une classification des règles génériques sous la forme d'une matrice, comme montré dans la Figure 2.3. Les catégories sont comme suit :

- **Index de priorité fixé à l'entrée, basé sur les informations des tâches** : dans ce groupe on trouve les règles de priorité basées sur des données statiques des tâches. Par exemple Earliest Due Date (EDD), Number Of Operations (NOP) ou encore First Come First Served (FCFS),
- **Index de priorité fixé à l'entrée, basé sur les informations des opérations** : notamment les règles Shortest Processing Time (SPT), Longest Processing Time (LPT) ou Operational Due Date (ODD),
- **Index de priorité fixé à l'entrée, basé sur les informations des ressources et de la charge** : on trouve les règles de priorité basées sur les informations des machines, par exemple la règle Shortest Setup Time (SST),
- **Index de priorité mis à jour par étapes, basé sur les informations des tâches** : ces règles ont une dépendance temporelle et par conséquent les priorités des tâches peuvent changer en fonction du temps. Par exemple Modified Due Date (MDD) ou SLAK remaining (SLK/SLACK),

- **Index de priorité mis à jour par étapes, basé sur les informations des opérations** : parmi ces règles se trouve Processing Time + Wait Time (PT+PW), Slack per Number Of Operations (S/OPN) ou Slack per Remaining Processing Time (S/RPT),
- **Index de priorité mis à jour par étapes, basé sur les informations des ressources et de la charge** : dans cette catégorie figure les règles de priorité Work In Next Queue (WINQ), Number of jobs in Next Queue (NINQ) ou PT+WINQ,
- **Index de priorité adapté par sondage et les informations des tâches** : High Response Ratio (HRN) ou Shortest Expected Processing Time (SEPT),
- **Index de priorité adapté par sondage et les informations des opérations** : dans cette catégorie on trouve une variante de la règle SPT qui est CEXPT qui tente d'éviter que des tâches ne finissent très en retard à cause de leur temps d'exécution, comme c'est le de la tâche avec le temps d'exécution le plus élevé,
- **Index de priorité adapté par sondage, charge et les informations sur les ressources** : comprends les règles de priorité COVERT et ATC.

	Attributs des tâches	*Détail des opérations*	*Charge et ressources*
Fixé à l'entrée	EDD NOP FCFS	SPT LPT ODD	SST
Mis à jour par étapes	MDD SLACK	CR, MOD S/OPN, CR/SI S/RPT+SPT	WINQ PT+WINQ
Adapté par sondage	CEXSPT		BD MF, RR

Figure 2.3. Classification des règles de priorité proposée par Kemppainen

2.3. Les facteurs affectant les performances des règles de priorité

Shahzad [31] présente un certain nombre de facteurs pouvant affecter les performances des règles de priorité. L'auteur les classe en trois familles « *variation de la charge* », « *variation des dates d'échéance* », et « *méthodes d'affectation des dates d'échéance* ».

2.3.1. La variation de la charge

La charge d'un système est principalement déterminée par le nombre de tâches dans l'atelier, leurs temps d'exécution, les dates qui séparent l'arrivée de plusieurs tâches, ou encore le routage des tâches. La charge à son tour peut affecter le nombre de tâche en standby sur les files d'attente et par conséquent la performance d'une règle de priorité. Dans la littérature la charge est souvent prise en considération [68], [72], [116], [127], et en se fiant aux résultats de ces approches, on remarque qu'il existe une corrélation entre la charge et la performance d'un ordonnancement, obtenu par règle de priorité ou en utilisant d'autres méthodes de résolution.

2.3.2. La variation des dates d'échéance

La variation des dates d'échéance, mais aussi l'écart entre le temps d'exécution, la date d'arrivée, et la date d'échéance, peuvent fluctuer les résultats des règles de priorité quand l'objectif considéré est basé sur le retard « *tardiness* » ou le « *flow time* » [128]. Dans une étude comparative des règles de priorité proposée par Lejmi & Sabuncuoglu [128], les auteurs démontrent que la variation des dates d'échéance ont deux effets sur les règles de priorité « *Type-I* » et « *Type-II* ». Si les dates d'échéance des tâches changent, les dates de fin d'exécution des tâches C_i peuvent aussi changer. On parle alors de Type-I. Quand un changement des dates d'échéance des tâches survient, l'objectif lié au retard/précocité (*tardiness/earliness*) change aussi, on parle alors de Type-II.

2.3.3. Méthodes d'affectation des dates d'échéance

Lors de la résolution d'un problème d'ordonnancement les dates d'échéance sont considérés comme étant un paramètre d'entrée et sont alors pris en considération lors de la résolution [31]. Dans un atelier de production réel où on dispose d'informations, la date d'échéance peut prendre comme valeur la date de préemption de la matière première, ou la date fixée par un client. Mais lorsqu'il s'agit de problèmes à données générées [39], [40], [45], il est impératif de définir une ou plusieurs fonctions ou règles afin d'associer une date d'échéance à chaque tâche. Ces dates d'échéance doivent être générées d'une façon à créer des scénarios proches de la réalité et où les solutions satisfont l'intégralité ou la majorité des dates d'échéance. Shahzad [31] définit deux classes de méthodes d'affectation des dates d'échéance :

2.3.3.1. Méthodes exogènes

Avec ce type de méthodes, les dates d'échéance sont fixées au préalable avant l'arrivée des tâches sur l'atelier. Elles ne prennent pas en considération les attributs de l'atelier ou ceux des autres tâches, comme le nombre de tâches sur l'atelier, ou la charge des machines. Elles peuvent être de deux types :

- Constant : on affecte la même date d'échéance à toutes les tâches,
- Aléatoire : les dates d'échéance sont choisies aléatoirement.

2.3.3.2. Méthodes endogènes

Les dates d'échéances des tâches sont définies en fonction des attributs de celles-ci, ou celles de l'atelier, ou la charge des machines. Parmi les règles utilisées on cite notamment :

- Total WorK content (TWK) : les dates d'échéance sont basées sur la somme totale des temps d'exécution,
- Number of OPerations (NOP) : les dates d'échéances sont déterminées à base du nombre d'opérations,
- SLACK : les dates d'échéances sont calculées à base du temps d'attente ou du slack (temps supplémentaire ajouté à une opération sans dégrader les performances),

- RAndom Total WorK content (RATWK) : les dates d'échéance sont basées sur les règles TWK, NOP et RAN à la fois.

Il existe aussi un certain de nombre de benchmarks dans la littérature [40], [92], [129], [130] contenant des jeux données pour des problèmes Single Machine, Job Shop, Flow Shop, open shop (avec ou sans flexibilité) où les dates d'échéances sont déjà définies suivant certains critères expliqués dans ces benchmarks. Il peut s'avérer donc très intéressant de tester une approche sur ces données et, grâce aux meilleures solutions déjà connues et fournies dans ce benchmarks, il est possible d'évaluer et de comparer l'approche proposée.

2.4. Etat de l'art

Le but des règles de priorité est d'optimiser le flux des tâches sur les machines tout en maximisant ou en minimisant un certain critère, comme la date maximale de fin d'exécution (makespan) [48], la maximisation du taux d'utilisation des machines (resource utilization) [131], ou encore le retard pondéré (weighted tardiness) [132] pour ne citer que les plus utilisés. Certaines règles de priorité sont construites afin d'optimiser certains critères. C'est notamment le cas de la règle « *Earliest Due Date* » qui est souvent couplée aux problèmes d'ordonnancement avec la minimisation du retard comme critère d'optimisation ou la règle « Shortest Processing Time » pour le makespan. Par ailleurs, lors de l'optimisation multi-objectifs, en utilisant une règle de priorité afin d'améliorer un objectif, celle-ci peut en détériorer un autre, d'où la nécessité de bien identifier la ou les règles d'ordonnancement afin de trouver un compromis entre les différents objectifs. Ceci pousse alors les responsables d'ateliers à tester diverses règles de priorité en utilisant la simulation afin de déterminer celle qui optimise au mieux la fonction objectif.

Comparons par exemple le comportement des règles FIFO (*First In First Out*) et SPT (*Shortest Processing Time*) : La règle FIFO donne la priorité à la tâche arrivée en premier. Elle garantit donc qu'aucune tâche ne soit bloquée dans la file d'attente car le tour de passage des tâches est déterminé par leur temps d'arrivée dans la file d'attente. Ce qui n'est pas le cas avec SPT si on utilise cette règle pour gérer la fille d'attente. Si une tâche est placée dans une file d'attente avec un temps d'exécution beaucoup plus élevé que les temps d'exécution des autres tâches cette tâche va alors attendre indéfiniment jusqu'à ce que toutes les autres tâches, prioritaires en raison de leur faible temps d'exécution, soient traitées par la ressource. D'un autre point de vue, SPT est très utile lorsque l'objectif en question est de réduire le makespan.

L'avantage des règles de priorité réside dans le fait qu'elles sont de simples heuristiques, facilement compréhensibles et peuvent être aisément implémentées par les responsables des ateliers de production. Elles peuvent être aussi couplées avec des simulateurs dans le but d'analyser et d'évaluer leurs performances. Mais aussi de superviser le comportement des tâches et des ressources afin de mieux appréhender le processus et par conséquent permettre au responsable de l'atelier de pouvoir intervenir lors de la construction de l'ordonnancement afin de l'améliorer. Les règles de priorité sont notamment sollicitées pour les problèmes d'ordonnancement à forte complexité ou dans le cas de problèmes dynamiques. C'est le cas par exemple pour la production des semi-conducteurs [122], [131], [133], [134] où l'utilisation de

techniques classiques d'optimisation comme le Branch & Bound ou les méthodes bio-inspirées s'avèrent être improductives. Les règles de priorité sont aussi favorisées pour les problèmes d'ordonnancement dynamiques où les informations sur les tâches et sur l'atelier ne sont pas connues à priori mais au fur et à mesure (dates d'arrivées ou temps d'exécution des tâches, maintenance, ou pannes des ressources [4], [26], [42], [124].

Dans la littérature il existe un nombre important de contributions sur les règles de priorité. Ces contributions couvrent plusieurs aspects, à savoir : la création de nouvelles règles de priorité, la combinaison de règles, la sélection de règles par simulation ou par intelligence artificielle, ou le réordonnancement. Les premiers travaux sur les règles de priorité remontent aux années 50/60 [135]–[139]. La base des règles de priorité selon Shahzad [31] remonte à l'approche présentée par Giffler & Thmpson [140] qui proposent un algorithme pouvant générer des ordonnancement de type actif (cf. Section 1.3.3). Cet algorithme consiste à choisir une tâche/opération avec la date de fin d'exécution la plus proche et d'identifier la machine sur laquelle elle doit être opérée. Ensuite, l'ensemble des autres tâches/opérations nécessitants la même ressource et qui commencent avant la date de fin d'exécution de la tâche initiale sont tous regroupés dans une liste, avec la tâche initiale. Enfin, à chaque fois que la ressource est disponible, une tâche de la liste est sélectionnée aléatoirement pour être traitée jusqu'à ce que toutes les tâches soient exécutées.

Nous présentons l'état de l'art sur les règles de priorité dans trois sections distinctes. La première couvre plusieurs règles qui ont été introduites pour la résolution de problèmes d'ordonnancement. La seconde expose les travaux de recherche qui se sont intéressés aux études comparatives des règles de priorité dans plusieurs types d'ateliers tel le Job Shop, le Single Machine, ou en fonction des critères d'optimisation. Enfin, la troisième traite de la sélection automatisée des règles de priorité, c'est-à-dire, l'utilisation d'heuristiques, aussi appelées « *hyper-heuristiques* », pour l'identification, la sélection, et l'allocation de règles de priorité aux ressources en fonction de leurs états et de l'état de l'atelier.

2.4.1. Proposition de règles de priorité

Parmi les travaux de recherche portant sur le développement de règles de priorité, on cite Holthaus & Rajendran [71] qui proposent 5 règles de priorité (voir les Équations 2.1 à 2.5) afin d'optimiser les objectifs liés au temps d'écoulement (flow time), à savoir minimiser ou maximiser le flow time et sa variance. Mais aussi minimiser le nombre de tâches en retard, le retard moyen et sa variance. Ces règles de priorité sont :

$$\text{PT+WINQ } Z_i = p_{ij} + W_i \tag{2.1}$$

$$\text{PT+WINQ+AT } Z_i = p_{ij} + W_i - (\tau - r_i) \tag{2.2}$$

$$\text{PT+WINQ+SLK } Z_i = p_{ij} + W_i - \min(SLK_i, 0) \tag{2.3}$$

$$\text{PT+WINQ+AT+SLK } Z_i = p_{ij} + W_i + r_i + \min(SLK_i, 0) \tag{2.4}$$

$$Z_i = {}^{c_i}/_{p_{ij}} \text{ où } c_i = \begin{cases} {}^{(W_i - SLK_i)}/_{W_i} \; si \; 0 \le SLK_i < W_i \\ 0 \; si \; SLK_i \ge W_i \\ 1 \; si \; SLK_i < 0 \end{cases} \tag{2.5}$$

Où Z_i dénote la priorité de la tâche i, $W_i = \sum_{j=0}^{N_i} p_{ij}$ (peut être dénoté p_i) représente le temps total d'exécution de la tâche i et τ est le temps auquel la décision est prise (choix d'une tâche). Ces règles de priorité sont comparées aux règles SPT, WINQ, FIFO, RR, et CONVERT dans un atelier Job Shop.

Jayamohan & Rajendran [14] proposent aussi plusieurs règles de priorité et effectue une étude comparative à base de simulations avec les règles SPT, PT+WINQ, ATC, MOD, EDD, ODD, RR, PT+WINQ+SL, et AT-RPT. Les règles proposées sont :

Flow Due Date (FDD) décrite par l'Équation 2.6 :

$$Z_i = FDD_{ij} = r_i + \sum_{q-1}^{j} p_{iq} \tag{2.6}$$

Processing Time + Wait Time (PT+PW) (Équation 2.7):

$$Z_i = p_{ij} - C_{i,j-1} \tag{2.7}$$

PT+PW+FDD (Équation 2.8) et PT+PW+ODD (Équation 2.9) :

$$Z_i = p_{ij} - C_{i,j-1} + FDD_{ij} \tag{2.8}$$
$$Z_i = p_{ij} - C_{i,j-1} - ODD_{ij} \tag{2.9}$$

Operational Flow Slack+Processing Time + Flow Due Date (OPFSLK/PT;FDD) donnée par l'Équation 2.10 et OPFSLK/PT;ODD décrite par l'Équation 2.11 :

$$Z_i = \max\{\tau + p_{ij} - FDD_{ij}; 0\}/p_{ij} \tag{2.10}$$
$$Z_i = \max\{\tau + p_{ij} - ODD_{ij}; 0\}/p_{ij} \tag{2.11}$$

Une autre règle de priorité proposée appelée AVPRO pour « AVerage PROcessing time » est décrite comme suit (Équation 2.12) :

$$Z_i = \sum_{j=1}^{m_i} p_{ij}/m_i \tag{2.12}$$

D'autres règles de priorité existent aussi dans la littérature comme par exemple « *Mixed Dispatching Rule* » (*MDR*) proposée par Yin & Wang [61] pour la résolution du problème d'ordonnancement à Machine Unique avec le retard moyen pondéré comme objectif (Single Machine total weighted tardiness). La règle de priorité proposée, selon les auteurs, prend en considération non seulement les attributs des tâches tels que le temps d'exécution ou les dates d'échéance, mais aussi les valeurs de la fonction objectif lors de la sélection de la tâche à exécuter. La règle de priorité MDR est décrite comme suit :

Soit $J = \{J_0, J_2, \dots, J_{n-1}\}$ l'ensemble des taches à ordonnancer et t le temps actuel. $t = 0$ désigne le début de l'ordonnancement.

Etape 1 :	Définir JUD $= \{J_k \mid d_k \geq t + p_k,\ J_k \epsilon J\}$, JD $= \{J_k \mid d_k < t + p_k,\ J_k \epsilon J\}$.
Etape 2 :	Si JUD $\neq \emptyset$, JD $= \emptyset$ choisir $J_i \epsilon J$ avec la date d'échéance la plus proche. Aller à l'étape 5.
Etape 3 :	Si JUD $= \emptyset$, JD $\neq \emptyset$ choisir $J_i \epsilon J$ avec le rapport p_i/w_i le plus petit. Aller à l'étape 5.
Etape 4 :	Si JUD $\neq \emptyset$, JD $\neq \emptyset$ choisir $J_i \epsilon$JUD avec la date d'échéance la plus proche ; choisir $J_j \epsilon JD$ avec le rapport p_i/w_i le plus petit. Calculer $w_j \times (t + p_i + p_j - d_j)$ et $w_j \times (t + p_j - d_j) + w_i \times \max\{t + p_i + p_j - d_j, 0\}$, • Si $w_j \times (t + p_i + p_j - d_j) \geq w_j \times (t + p_j - d_j) + w_i \times \max\{t + p_i + p_j - d_j, 0\}$, choisir la tâche $J_j \epsilon$JD. Aller à l'étape 5. • Si $w_j \times (t + p_i + p_j - d_j) < w_j \times (t + p_j - d_j) + w_i \times \max\{t + p_i + p_j - d_j, 0\}$, choisir la tâche $J_i \epsilon$JUD. Aller à l'étape 5.
Etape 5 :	Supposons que la tâche sélectionnée soit J_1, mettre à jour la valeur de t, $t = t + p_1$.
Etape 6 :	Définir $J = J \backslash \{J_1\}$, si $J = \emptyset$, arrêt. Sinon revenir à l'étape 1.

La règle de priorité proposée MDR est testée pour des problèmes « Single Machine » avec retard moyen pondéré fournis dans la librairie OR-Library [141] (benchmarks).

Dans la littérature on dénombre aussi de nombreux travaux où l'on développe des algorithmes pour l'ordonnancement en ligne. Par exemple Albers & Schmidt [142] proposent un algorithme reprenant les spécificités des règles de priorité pour l'ordonnancement en ligne afin de réagir aux perturbations machines. Cet algorithme prend parmi les entrées les temps des pannes qui sont connues à l'avance. Anderson & Potts [44] en s'inspirant de la règle de priorité « *Shortest Weighted Processing Time* » proposent une nouvelle règle appelée « *Delayed SWPT* » qui est plus effective que le SWPT pour les problème Single Machine en ligne avec le total des retards pondérés comme critère d'optimisation. Les auteurs démontrent aussi que la règle suggérée a un ratio (rapport) de compétitivité (*competitive ratio*) de 2, où ce ratio est donné par la valeur du critère d'optimisation d'un ordonnancement en ligne divisé par celle d'un ordonnancement hors ligne. La valeur de l'ordonnancement en ligne étant toujours supérieures ou égale et celle du hors ligne du aux nombreuses perturbations pouvant avoir lieu dans le système de production.

Dans la même perspective, Cheng et al. [134] proposent un système où à chaque fois qu'une perturbation est détectée, comme une panne machine, une notification est émise. Cette notification est envoyée d'une manière récursive suivant certains chemins (arcs) définis préalablement et à la fin toutes les tâches/machines sont informées de la panne survenue que ça soit d'une manière directe ou indirecte. Les tâches/opérations modifient alors leurs dates de début d'exécution d'une manière à ce que l'ordonnancement initial puisse inclure la perturbation. Hildebrandt et al. [122] utilisent la programmation génétique (*Genetic Programming*) pour la génération de nouvelles règles de priorité où le but est de réduire les temps des cycles des lots (*cycle time of lots*). Une étude comparative de l'approche proposée est ensuite est réalisée qui, selon les auteurs, surpasse plusieurs règles de priorité de la littérature.

Une liste plus complète des différentes règles de priorité proposées dans la littérature est donnée dans le Tableau 2.1 de la Section 2.5. Il est utile de noter que cette liste ne couvre pas la totalité des règles existantes. La fouille de données a aussi été très largement utilisée pour la génération

de règles de priorité à base de méthodes de résolution exactes ou approchées, ce processus est généralement appelé « *extraction de règles de priorité* ». Cette approche est présentée dans la Section 3.4.2 du Chapitre 3.

2.4.2. Evaluation des règles de priorité

Dans la littérature figure aussi des travaux de recherche dont le but est d'évaluer l'impact des règles de priorité sur un ou plusieurs objectifs dans différents types d'atelier. Le but de ces travaux est de rapporter les règles de priorité qui s'adaptent au mieux en fonction du contexte et peuvent servir comme point de départ lors de la résolution de problèmes d'ordonnancement à base de règles de priorité. Ceci évite d'explorer l'ensemble des règles, en utilisant la simulation, et de concentrer les efforts sur un sous-groupe de règles plus efficaces.

Rajendran & Holthaus [66] explorent l'impact de différentes règles de priorité sur plusieurs critères d'optimisation dans un atelier Job Shop. Les règles de priorité utilisées sont : FIFO, AT (*Arrival Time*), EDD, S/OPN (*Slack per remaining OPeratioN*), COVERT (*Cost OVER Time*), RR (*Raghu & Rajendran rule*), SPT, PT+WINQ (*Process Time plus Work In Next Queue*), et PT+WINQ+AT et PT+WINQ+SL. Les objectifs considérés sont la moyenne, le maximum, la variance du temps d'écoulement (*mean, maximum et variance flow time*), pourcentage des tâches en retard (*percentage of tardy jobs*), ainsi que la moyenne, le maximum et la variance du retard des tâches (*mean, maximum et variance tardiness*).

Hicks & Pongcharoen [112] ont étudié l'influence des mises à jours des données, des temps de préparation, et des temps de transferts entre les machines/ateliers. Dans cette étude, les auteurs comparent les performances de huit règles de priorité à savoir : EDF (*Earliest Due date First*), FEF (*First Event First*), LOF (*Longest Operation time First*), LRF (*Least Remaining operations First*), LSF (*Least Slack First*), MRF (*Most Remaining operations First*), RND (*random*), et SOF (*Shortest Operation time First*). Les auteurs proposent aussi trois nouvelles heuristiques (PT+WINQ)/TIS, PT/TIS, et AT-RPT (*TIS : Time In Shop, RPT : Remaining Processing Time*) L'objectif défini comme critère d'évaluation est de minimiser la moyenne des retards (*mean tardiness*). Les expérimentations sont effectuées sur des problèmes de type Job Shop réels, fournis par plusieurs compagnies, composés de 56 produits avec 5539 opérations et un ensemble de 36 machines.

Chiang [76] explore l'impact de seize règles de priorité sur trois critères d'optimisation à la fois (optimisation multi-objectifs), le pourcentage des tâches en retard (*tardy rate*), le retard maximal (*maximum tardiness*), et le retard moyen (*mean tardiness*). Le problème considéré est composé de 20, 30, 40, et 50 tâches avec 15 ou 20 machines. Les auteurs introduisent aussi une nouvelle heuristique basée sur les règles SPT, EDD, et LPT où le but est de trouver des solutions avec des performances équilibrées.

Song et al. [25] expérimentent plusieurs règles de priorité dans un atelier Flow Shop à deux niveaux. Le premier étant constitué de trois machines parallèles identiques alors que le second est composé d'une seule machine. Le but des règles de priorité testées est de minimiser le temps d'exécution total (makespan). En plus des règles de priorité classiques FIFO, SPT, et LPT les auteurs proposent deux heuristiques modifiées qui sont SPT2 et LPT2. Ces heuristiques

suggérées additionnent les temps d'exécution des tâches requises dans le premier et le second niveau. Deux autres heuristiques sont proposées qui consistent à calculer la différence du temps d'exécution requis lors de la première phase avec celui de la seconde, et la priorité est donnée à la tâche qui minimise la valeur avec la règle STD (*Shortest Time Difference*) ou celle qui la maximise (*Longest Time Difference*).

Sharma & Jain [93] effectuent aussi une étude comparative de règles de priorité dans un atelier Job Shop où les tâches possèdent des temps de préparation par séquence (sequence-dependant setup time) et qui arrivent d'une façon dynamique sur l'atelier. Les règles FCFS, SPT, SIMSET, EDD, SSPT, JSPT, JEDD, JMED, et JSSPT sont considérés pour l'optimisation du temps d'exécution total (makespan), la moyenne et le maximum des retards (mean et maximum tardiness), la moyenne et le maximum du temps d'écoulement (mean et maximum flow time), nombre de tâches en retard (number of tardy jobs), la somme et la moyenne des temps de préparation (total et mean setup times). L'atelier est composé de dix machines avec deux taux d'utilisation considérés, 85 et 90%.

2.4.3. Sélection des règles de priorité

Dans les problèmes d'ordonnancement multi-objectifs l'utilisation de simples règles de priorité devient improductive car il n'existe pas de règles pouvant optimiser plusieurs critères. Comme par exemple la règle SPT qui retourne de bons résultats [143] si l'objectif considéré est le makespan, alors que si l'objectif est de réduire les retards, cette règle aura des très mauvaises performances. Les règles de priorité ayant été développées afin de maximiser/minimiser un certain objectif. Afin de remédier à ce type de problèmes, différentes règles de priorité sont utilisées à la fois sur plusieurs machines pour optimiser l'ensemble des objectifs. Il est aussi possible de changer continuellement les règles de priorité en fonction de l'état des tâches et/ou ressources afin de maintenir l'ensemble des critères d'optimisation à des niveaux satisfaisants. Par exemple, Pierreval & Mebarki [144] proposent de changer d'une façon dynamique les règles de priorité utilisées afin d'optimiser la fonction objectif. Ce changement de règles de priorité est effectué lorsqu'une ressource est libre ou quand une nouvelle tâche arrive en plus d'autres évènements définis par les auteurs. Pour décider des règles de priorité à utiliser les auteurs utilisent « *Standard Rules Heuristic Strategy* » avec la moyenne du temps d'écoulement (*mean flow time*) comme objectif principal et le retard moyen (*mean tardiness*) comme objectif secondaire.

Dans la même perspective, Jeong & Kim [145] proposent un système de sélection automatisée des règles de priorité pour un système de production flexible (*flexible manufacturing system*). Ce système est composé d'un simulateur, d'un ordonnanceur, de plusieurs bases de données, et d'un contrôleur. Trois bases de données sont considérées dans cette architecture, une qui contient les informations sur l'état des tâches (en cours de traitement, en attente, ou en transit), sur l'état des machines (occupée, libre, ou en maintenance). Quant à la seconde base, elle regroupe les informations sur les différentes règles de priorité utilisées ; les auteurs considèrent 16 règles. Enfin, la troisième base de données contient les temps d'exécution des tâches, les dates d'échéances, et les ordres de passage. Le module de contrôle supervise l'exécution des tâches sur le système de production et met à jour les données de la base contenant les

informations du système (première base de données). Si une perturbation survient, celui-ci notifie le module d'ordonnancement. Lorsqu'une perturbation est signalée par le contrôleur ou lorsque les performances de l'ordonnancement se détériorent, le module d'ordonnancement réagit en ordonnant au simulateur de déterminer la meilleure règle de priorité pour le scénario en cours. Le simulateur, à son tour, teste les 16 règles et détermine celle qui optimise au mieux la fonction objectif et sera ainsi renvoyée au contrôleur, en passant par l'ordonnanceur, qui la mets en place dans l'atelier.

Chan et al. [124] proposent un algorithme pour affecter d'une manière dynamique une règle de priorité à chaque machine dans un système de production flexible. Cette affectation se fait suivant certains évènements, où le système juge nécessaire d'utiliser d'autres règles de priorité afin d'optimiser l'ordonnancement. L'atelier considéré supporte un certain nombre de tâches qui peuvent attendre dans une zone tampon (ou une file d'attente partagée par les machines). Si le buffer est saturé et qu'une nouvelle tâche arrive dans l'atelier, la simulation est arrêtée et la ou les règles de priorité utilisées sont considérés comme inefficaces et une autre combinaison de règles et testée. L'atelier est composé de cinq machines pour le traitement et d'une autre pour le chargement et le déchargement des tâches. Le transport d'une machine à une autre est assuré par deux véhicules (*Automated Guided Vehicules*). Les objectifs utilisés sont la moyenne du temps d'écoulement (*mean flow time*), la moyenne des retards (*mean tardiness*), et la moyenne de l'avance des tâches (*mean earliness*).

Korytkowski et al. [19] proposent l'utilisation d'un algorithme génétique pour l'affectation de règles de priorité aux machines. Chaque chromosome représente un ensemble de règles et sa taille est égale au nombre de machines m. Les auteurs considèrent sept règles de priorité FIFO, SPT, EMODD (*Earliest Modified Operational Due Date*), CR+SPT (*Critical Ratio*), S/OPN (*minimum Slack time per remaining OPeratioN*), S/RPT+SPT (*S/RPT: Slack per Remaining Process Time*) et PT +WINQ + SL (*WINQ: Work In the Next Queue, SL: Slack*). Les objectifs considérés sont la moyenne et le maximum du temps d'écoulement (*mean & maximum flow time*), et la moyenne et le maximum du retard (*mean & maximum tardiness*). L'algorithme génétique proposé est couplé avec un module de simulation développé avec Arena[1] afin de tester les différentes combinaisons de règles de priorité. Enfin, les expérimentations sont effectuées sur un atelier Job Shop constitué de neuf machines et trois types de tâches.

La sélection des règles de priorité, aussi adressée comme « *combinaisons de règles* » [143], a pour but d'affecter une règle de priorité, qui peut être différente, à chacune des ressources du système de production. La règle en question doit pouvoir maximiser/minimiser une ou plusieurs critères d'optimisation en se basant sur les caractéristiques de la ressource en question ou des tâches qui requière celle-ci. Cependant, le comportement des ressources évolue continuellement générant un besoin de changement de l'approche considérée, c'est-à-dire un changement de règle de priorité en prenant en compte l'état réel de l'atelier [124], [143], [144].

[1] Arena est un logiciel de simulation et d'automatisation développé par « *System Modeling* » est acheté par « *Rockwell Automaton* ». Il peut être utilisé pour reproduire un système de production et pour tester les règles de priorité. Pour plus d'informations visitez www.***arenasimulation***.com.

Dans la littérature on recense de nombreux autres travaux de recherche ayant porté sur la sélection des règles de priorité mais en utilisant l'intelligence artificielle et la fouille de données. Ces travaux seront présentés et discutés en détails dans la Section 3.4.1 du Chapitre 3 portant sur l'utilisation de la fouille de données dans l'ordonnancement.

2.5. Les règles de priorité

Dans cette section, nous présentons plusieurs règles de priorité proposées dans la littérature [31], [127] (voir Tableau 2.1). Chaque règle de priorité est facilement compréhensible et peut être rapidement implémentée. Ces règles sont utilisées afin d'affecter une priorité à chaque tâche lorsqu'une machine est libre et qu'il existe des tâches en attente dans sa file.

Tableau 2.1. Les règles de priorité

Règle	Définition	max min	Indice de priorité	Source
ATC	Apparent Tardiness Cost	***max***	$\frac{1}{p_{ij}}\exp\left(-\frac{d_j - t - p_{ij} - h_2\sum_{i=l+1}^{o_j} p_{ij}}{h_3\sum_{i=l}^{o_j} p_{ij}}\right)$	[146]
AT-RPT	Arrival Time – Remaining Processing Time	***min***	$r_i - \sum_{i=l}^{o_j} p_{ij}$	[66]
AVPRO	Average Processing time	***min***	$\sum_{j=1}^{m_i} p_{ij}/m_i$	[147]
BD	Bottleneck Dynamics	***max***	$w_i U_{ij}(t)/\sum_{q=j}^{w_i} R_{k(q)}(t)p_{iq}$	[148]
CEXSPT	Conditionally Expecting SPT		Voir note [i]	[149]
COST	Composite Cost rule	***min***	$C_i V_i(d_{ijk} - t) + C_{2q}K_{1k}p_{ijk}{}^{-1}$	[150]
COVERT	Cost Over Time	***max***	$\frac{w_i}{p_{ij}}\max\left(0,1 - \frac{\max(0, d_i - t - \sum_{i=l}^{o_j} p_{ij})}{k\sum_{i=l}^{o_j} w_{ij}}\right)$	[139]
CR	Critical Ratio	***min***	$d_i - t/\sum_{i=l}^{o_j} p_{ij}$	[151]
CR+SI	Combinaison des règles CR et SI	***min***	$p_{jk} \times \max\left((d_j - t)/\sum_{i=l}^{o_j} p_{ij}, 1\right)$	
CR+SPT	Combinaison des règles CR et SPT	***min***	$p_{ij} \times \max\left(d_i - t/\sum_{i=l}^{o_j} p_{ij}, 1\right)$	[152]
EDD	Earliest Due Date	***min***	d_i	[135]
EFCD	Earliest Fraction Completed Date	***min***	$f d_j + (1-f)r_j$ où $f = w_j^{com}/w_j^{com} + w_j^{rem}$	[153]
EFD	Earliest Finish Date	***min***	C_i	[154]
ERD	Earliest Release Date	***min***	r_i	[155]
FCFS	First Come First Served	***min***	r_i	[138]
FIFO	First In First Out	***min***	C_i	
LOPT	Longest Operation Processing Time	***max***	p_j^c où $j \in q_m$	[125]
LPT	Longest Processing Time	***max***	p_i	[138]
LS	Least Slack	***min***	$(d_j - t_c) - w_j^{rem}$	[125]
LS*OPT	Least Slack times Operation Processing Time	***min***	$\left((d_j - t_c) - w_j^{rem}\right) \times p_j^c$	[153]
LS/OPT	Least Slack per Operation Processing Time	***min***	$(d_j - t_c) - w_j^{rem}/p_j^c$	[153]

Règle	Définition	max min	Indice de priorité	Source
LS/RO	Least Slack per Remaining Operations	***min***	$(d_j - t_c) - w_j^{rem} \Big/ \lvert O_j^{rem} \rvert$	[125]
LS+LR	Least Slack plus Late Rule	***min***	$\max(0, (d_j - t_c) - w_j^{rem})$	[153]
LWKR	Least total Work Remaining	***min***	$\sum_{i=l}^{o_j} p_{ij}$	[135]
MAXPEN	Maximum Penalty	***max***	w_i	[156]
MDD	Modified Due Date	***min***	$\left(d_i, t + \sum_{i=l}^{o_j} p_{ij}\right)$	[155]
MF	Multi-Factor	***max***	$\frac{w_i}{p_{ij}} \max\left(W_{ij} - \left(d_j - \max t - \sum_{i=l}^{o_j} p_{ij}\right)\right)$	[157]
MOD	Modified Operation Due date	***min***	$\max(d_{ij}, t + p_{ij})$	[155]
MT/OPT	Most Tardy over Operation Processing Time	***max***	$t_c - d_j \Big/ p_j^c$	[125]
MT/RPT	Most Tardy over Remaining Processing Time	***max***	$t_c - d_j \Big/ w_j^{rem}$	[153]
MT+ER	Most Tardy plus Early Rule	***max***	$\max(0, t_c - d_j)$	[153]
MW/D	Maximum Work per Day	***max***	$w_j^{rem} \Big/ \max(1, \lceil d_j - t_c \rceil)$	[153]
MXPROF	Most Profitable job in the queue	***max***	$(p\ m)_i$	[158]
NOP	Number Of Operation	***min***	m_i	[135]
ODD	Operational Due Date	***min***	d_{ij}	[135]
P/TWK	Relative length of next operation	***min***	$p_{ij} \Big/ \sum_{i=l}^{o_j} p_{ij}$	[71]
P/TWK	Relative Processing Time	***min***	$p_{ij} \Big/ \sum_{i=l}^{o_j} p_{ij}$	[135]
PT/TIS	Processing Time per Time In Shop	***min***	$p_{ij} \Big/ (t - r_i)$	[66]
PT+ WINQ	Combinaison de PT et WINQ	***min***	$p_{ij} - W_{i,j-1}$	[71]
PT+PW	Combinaison de PT et PW (wait time)	***min***	$p_{ij} - C_{i,j-1}$	[14]
PT+WINQ/ TIS	Processing Time + Work In Next Queue per Time In Shop	***min***	$\left(p_{ij} + \sum_{q-1}^{j} p_{iq}\right) \Big/ (t - r_i)$	[66]

Règle	Définition	max min	Indice de priorité	Source
RRrule	Raghu & Rajendran rule	***min***	$\left(\frac{\left(d_j - t - \sum_{i=l}^{o_j} p_{ij}\right)\exp(-\eta) \times p_{ij}}{\sum_{i=l}^{o_j} p_{ij}} + \exp(\rho) \times p_{ij} + W_{nxt}\right)$	[159]
S/OPN	Slack per Remaining Operation	***min***	$\frac{d_i - t - \sum_{i=l}^{o_j} p_{ij}}{o_j - i - 1}$	[136]
S/RAT	Slack per Remaining Allowable Time	***min***	$\left(d_i - t - \sum_{i=l}^{o_j} p_{ij}\right) / (d_i - t)$	[160]
S/RPT	Slack per Remaining Processing Time	***min***	$\frac{d_i - t - \sum_{i=l}^{o_j} p_{ij}}{\sum_{i=l}^{o_j} p_{ij}}$	[136]
S/RPT+ SPT	Combinaison des règles S/RPT et SPT	***min***	$p_{ij} \times \max\left\{\frac{d_i - \sum_{i=l}^{o_j} p_{ij} - t}{\sum_{i=l}^{o_j} p_{ij}}, 1\right\}$	[152]
SI	Shortest Imminent	***min***	p_i	[135]
SI+SLACK+ SQNO	Combinaison des règles SI, SLACK et SQNO	***max***	$\frac{\left(h_1 \varsigma_j^* + h_2 RQNO_{ji}\right)}{p_{ij}}$	[75]
SIO	Shortest Imminent Operation	***min***	p_{ij}	[135]
SIRO	Service In Random Order		Voir note [ii]	
SI^x	Modified Shortest Imminent		Voir note [iii]	[137]
SLACK/MS T	Minimum Slack Time	***min***	$d_i - t - \sum_{i=l}^{o_j} p_{ij}$	[135]
SOPT	Shortest Operation Processing Time	***min***	p_j^c où $j \in q_m$	[125]
SPT	Shortest Processing Time	***min***	p_i	[138]
SPT-T	Shortest Processing Time with Truncation	***min***	$\min(SI + \gamma, S/OPN)$	[161]
SQNO	Shortest Queue at the Next Operation first	***min***	$\left\|q\left(m(j,+1)\right)\right\|$	
SST	Shortest Setup-Time	***min***	s_{ijk}	[150]
VALADD	Value-Added	***max***	V_{ij}	[158]
WECT^h/LFT	Weighted Earliest Completion Time/Latest Finish Time	***min***	$(r_i + l_i)/h_i$ [iv]	[15]
WEFT$^\tau$/ECT	Weighted Earliest Finish Time/Earliest Completion Time	***min***	$\left(r_i + EFT_{ij}\right)/\tau_i$	[15]
WINQ	Work In Next Queue	***min***	$W_{i,j+1}$	[135]

Règle	Définition	max min	Indice de priorité	Source
WJDD/LFT	Weighted Job Due Date/Latest Finish Time	***min***	d_i/τ_i	[15]
WLFT[τ]/ECT	Weighted Latest Finish Time/Earliest Completion Time	***min***	$(r_i + LFT_{ij})/\tau_i$[v]	[15]
WLS	Weighted Loss of Slack		Voir note [vi]	[162]
WMDD	Weighted Modified Due Date	***min***	$\max\{d_i - 1, p_i\}/w_i$	[163]
WSPT	Weighted Shortest Processing Time	***max***	v_i/p_{ij}	[146]
WTWKR[h]/ LFT	Weighted Total Work Content Remaining/Latest Finish Time	***min***	$TWKR_i/h_i$ [vii]	[15]

[i] CEXSPT utilise trois files d'attentes et les tâches en attente sont classées dans l'une de ces files. La première file d'attente regroupe les tâches en retard « *late queue* ». La seconde file contient les tâches avec un retard opérationnel « *behind schedule* » et la dernière les tâches en avance « *ahead of schedule* ». La tâche avec le temps d'exécution le plus petit (SPT) de la file 1 est exécutée en premier. Si cette opération engendre une autre tâche en retard alors la 2ème file d'attente est triée en utilisant SPT est la tâche la plus prioritaire est sélectionnée. Si cette dernière crée une tâche pouvant faire partie de la 2ème file alors la tâche avec le temps d'exécution le plus petit de la 3ème file est alors exécutée.

[ii] La tâche à exécuter est choisie aléatoirement parmi les tâches en attente dans la file. Ce processus est répété jusqu'à ce que toutes les tâches soient traitées.

[iii] Deux files d'attente 1 et 2 sont utilisées, chacune est triée en utilisant la règle de priorité SI. Les tâche avec un slack (temps supplémentaire ajouté aux opérations sans dégrader les performances afin de les retarder) inférieur à un paramètre Q est placé dans la première file d'attente.

[iv] l_i est la somme des temps d'exécution des produits le long du chemin critique de la tâche i. LFT est utilisée pour le tie-break.

[v] τ_i est le poids pour le retard de la tâche i. Et ECT pour le tie-break.

[vi] Cette règle compare la situation des autres opérations, dans la même file d'attente, si une tâche i est exécutée. Ceci causera un changement de slack des opérations restantes. Les opérations restantes sont divisées en deux files d'attente. La 1ère, « high-priority » contient les opérations avec un slack inférieur ou pouvant avoir un slack si l'opération i est exécutée. Cette première file est triée en utilisant une formule de WLS donnée dans [162]. La 2ème file d'attente contient les tâches non prioritaires « low-priority » où le slack est positif. Cette dernière est triée en utilisant la règle de priorité ODD.

[vii] $TWKR_i$ est la somme des temps d'exécution des operations restantes de la tâche i et h_i est le poids pour le temps d'écoulement (flow time). LFT est utilisé pour définir la tâche la plus prioritaire, au cas où plusieurs tâches obtiennent la même priorité (tie-break).

2.6. Conclusion

Dans ce chapitre nous avons résumé plusieurs travaux de recherche ayant porté sur l'utilisation des règles de priorité avec un intérêt particulier aux ateliers Job Shop et Single Machine. Cet état de l'art indique que de nombreux chercheurs se sont intéressés de près ou de loin aux règles de priorité. On dénombre certaines recherches où l'intérêt est d'évaluer, en se basant sur la simulation, l'impact des règles de priorité sur plusieurs critères d'optimisation et dans différentes architectures d'ateliers. D'autres recherches se sont focalisées sur l'introduction de nouvelles règles ou la combinaison de règles déjà existantes pour l'optimisation multi-objectives. Mais aussi les hyper-heuristiques où le but est d'affecter une règle de priorité à chaque ressource en fonction de sa spécificité, des tâches en attentes, ou en cas de perturbations.

Cet intérêt porté aux règles de priorité est justifié tout d'abord par leur rapidité. En effet, une règle de priorité affecte une priorité à chaque tâche ou opération en attente pour une ressource donnée en se basant sur ses caractéristiques, les caractéristiques de la machine, ou en combinant les deux à la fois. Ce processus est effectué quasi instantanément évitant ainsi aux ressources de rester inactives, de maximiser leur taux d'utilisation, et minimiser le temps d'exécution total (ou optimiser tout autre objectif). L'autre particularité des règles de priorité est qu'elles sont très utiles dans les environnements dynamiques (arrivée dynamique des tâches, pannes machines, maintenances non programmées, …) où l'utilisation de méthodes exactes ou approchées peut s'avérer improductif dû à leur lenteur et leur manque de responsivité.

Cependant, en se fiant aux différentes recherches menées il en ressort qu'aucune règle de priorité ne peut surpasser les autres règles quel que soit le problème traité (Single Machine, Job Shop, Flow Shop), son type (statique ou dynamique), ou le(s) critère(s) d'optimisation considéré(s) (makespan, tardiness, earliness, ou autres). Ceci a poussé les chercheurs à développer de nouvelles méthodes, originales, pour l'identification de la ou les meilleures règles de priorité et ne plus se reposer entièrement sur la simulation. Parmi les méthodes considérées se trouve l'intelligence artificielle et la fouille de données, qui en se basant sur des expériences passées, sur la simulation, ou sur l'expérience de l'opérateur humain, génèrent un modèle de décision/sélection afin d'identifier la ou les meilleures règles de priorité à utiliser.

CHAPITRE

3. La Fouille de Données dans l'Ordonnancement : un Etat de l'Art

Résumé : Durant de la production, d'énormes quantités de données sont générées, issues de l'ordonnancement, des ressources, les capteurs, ou autre. Ces données volumineuses et, de par leur nature, complexes sont difficilement analysables et compréhensibles alors que leur potentiel pour les entreprises ou pour le secteur académique est d'une importance capitale. Ceci a donc poussé les acteurs de ces domaines à explorer de nouvelles techniques pouvant analyser et extraire de la connaissance à partir de ces gros volumes de données. La fouille de données offre ainsi une panoplie de techniques qui s'adapte quasiment à tous types d'informations et peuvent, comme annoncé dans l'état de l'art, être utilisées dans les problèmes d'ordonnancement pour assister les opérateurs humains trouver une solution. La fouille de données réussie ainsi là où il est impossible de formuler mathématiquement la connaissance des experts humains ou les données issues des systèmes de production.

Mots-clés : Extraction de Connaissances à partir de Données · Fouille de Données · Types d'Utilisation · Techniques de Fouille de Données · Arbres de Décision · Fouille de Données dans l'Ordonnancement

3.1. Introduction

Lors du processus d'ordonnancement la personne en charge peut se fier à son expérience acquise au fil des années pour la résolution de certains problèmes à très faible complexité ou faire face à quelques scénarios de perturbation. Ceci est notamment possible grâce à la capacité de l'être humain à « *apprendre* » et à constituer sa propre « *connaissance* ». Cependant, l'apprentissage de l'être humain est un processus lent est compliqué [164]. Afin d'y remédier l'Intelligence Artificielle (*IA*) a été développée pour simuler l'apprentissage de l'être humain et ouvrir ainsi de nouvelles perspectives. L'analyse, la fouille, l'apprentissage, l'extraction, la classification, et la prise de décision deviennent ainsi plus faciles et surtout plus rapides ouvrant la voie à la résolution de problèmes de très forte complexité.

Les problèmes d'ordonnancement sont considérés comme étant fortement complexes et font partie de la famille des problèmes NP-Difficile. Pour les résoudre, et comme présenté dans les chapitres précédents, il est possible de recourir aux méthodes issues de la recherche opérationnelle comme le Branch & Bound mais aussi l'IA [164]. Dans la littérature on distingue trois types d'utilisation de l'intelligence artificielle et/ou de la Fouille de Données (*FD*) dans les problèmes d'ordonnancement. Le premier type d'utilisation consiste à accélérer et automatiser le processus de sélection de règles de priorité puisque, à priori, aucune règle ne peut être privilégiée par rapport aux autres. Le but étant de ne plus utiliser de simulations pour déterminer la meilleure règle, et ainsi pouvoir en changer à la volée en cours d'exécution.

Le deuxième type d'utilisation, proche du premier, s'appuie sur l'IA et la FD afin réagir aux perturbations et aux pannes en déterminant en temps-réel la meilleure règle à utiliser pour en minimiser les impacts. Si l'on doit réagir à une perturbation, une panne machine par exemple, en utilisant une méthode classique pour la résolution (méthode exacte ou approchée), alors il est nécessaire de suspendre l'exécution de toutes les tâches dans l'atelier et de lancer la méthode de résolution afin de trouver un nouvel ordonnancement. Ensuite, lorsque la panne est rétablie, le même processus doit être refait impactant négativement le(s) critère(s) d'optimisation. Pour un problème fortement complexe comme le Job Shop, le temps de résolution est important et le système prendra beaucoup de temps pour réagir contrairement aux modèles de décision obtenus par fouille de données.

Quant au troisième type d'utilisation de l'IA et/ou de la Fouille de Données, il a pour but d'analyser et de reproduire le comportement d'une méthode de résolution exacte ou approchée. En effet, en utilisant les arbres de décision, il est possible de transformer une séquence, après un processus prétraitement, et de générer de nouvelles de priorité sous la forme « *si-sinon* ». Ces nouvelles règles imitent le comportement de la méthode de résolution utilisée pour obtenir la séquence initiale, et qui peuvent être considérés pour la résolution de nouveaux problèmes ou dans un environnement dynamique.

Selon Solarte [165] l'utilisation de l'intelligence artificielle et de la fouille de données, au-delà de l'ordonnancement s'étend aussi aux autres sous-domaines de l'ingénierie industrielle comme le contrôle de qualité, la réduction des coûts, la sécurité, la maintenance des équipements. Elle peut ainsi être un facteur déterminant au sein de l'entreprise en lui permettant d'être plus

compétitive et pouvoir faire face à la concurrence et aux nouveaux défis. La fouille de données a été appliquée avec succès là où il est difficile, voire impossible, pour un opérateur humain de capturer tous les aspects d'un système et de les transformer en modèle [166]. La fouille de donnée offre ainsi cette possibilité d'extraire de la connaissance à partir de données complexes, comme c'est le cas dans l'ordonnancement, ou quand les données sont manquantes ou lorsqu'on s'intéresse à deux gros volumes de données. Ces fonctionnalités offertes par la fouille de données ont motivé en grande partie ce travail de thèse, où le but est d'utiliser ses techniques pour l'extraction des règles de priorité, leur analyse, l'automatisation de leur sélection et pour l'optimisation de l'ordonnancement.

La fouille de données est l'étape centrale du processus d'extraction de connaissances à partir des bases de données, nous commençons alors ce chapitre par la description de ce processus et de ses différentes composantes. Nous nous intéressons ensuite à la fouille de données, son historique, son utilisation, et ses techniques. Puis nous faisons un tour de la littérature sur l'utilisation de la fouille de données pour les problèmes d'ordonnancement et les principales contributions réalisées. Nous terminons ce chapitre avec la procédure complète de création des arbres de décision selon les algorithmes ID3 et C4.5.

3.2. L'extraction des connaissances à partir de données

Le processus d'Extraction de Connaissances à partir de Données (*ECD*), connu en anglais sous le nom « *process of Knowledge Discovery in Databases (KDD)* », regroupe l'ensemble des opérations qui permettent d'exploiter avec facilité et rapidité les données stockées massivement ou les données à forte complexité [167]. L'ECD est aussi défini comme étant « *un processus non trivial d'identification de structures inconnues, valides, et potentiellement utiles dans les bases de données* » [168]. Selon Pinto [169], le terme « *donnée* » est un ensemble de faits ou d'éléments atomiques d'information, comme le cas d'un enregistrement dans une table. Tandis que « *connaissances* » désigne un concept de niveau supérieur qui se rapporte aux propriétés de la collecte de données dans son ensemble. L'extraction de connaissances à partir de données a été utilisé dans une large panoplie de domaines comme le marketing et le management [169], le domaine médical [170], ou encore dans l'industrie [171].

3.2.1. Le but de l'ECD

Dans la littérature il existe un nombre important de travaux de recherche sur l'analyse de données, l'aide à la décision, et l'extraction de connaissances basés sur l'ECD y compris dans les domaines de l'ingénierie industriel comme l'ordonnancement [172], [173]. L'extraction de connaissance à partir de données a deux principaux objectifs [167], [169] en fonction de l'utilisation et des données traitées et qui sont :

- La Prédiction/Classification : elle consiste à prédire des valeurs/attributs en se basant sur un ensemble de données utilisées pour l'apprentissage, dans le cas d'un apprentissage supervisé. Le principal but de ce processus est de générer un modèle de décision capable de classer de nouvelles instances de problèmes en leur affectant une classe (label ou famille) ou une valeur estimée. En se basant sur ce procédé, il est

possible de créer un système d'aide à la décision et ses résultats seront directement utilisés lors de la prise de décision. Seulement, il nécessaire que les décisions prises par le système soient correctes, d'où l'obligation d'améliorer son taux de bonne classification, aussi connu en tant que « *précision* »,

- La Description/Découverte : son but est d'aider l'être humain à mieux comprendre les données, à extraire certaines relations entre celles-ci, ou simplifier et vulgariser la présentation et la compréhension de ces données. Il peut ainsi aider à analyser les habitudes d'achats des clients ou à comprendre le comportement des tâches/opérations dans un système de production. Contrairement à la prédiction, l'importance n'est pas tellement celle accordée au taux de bonne classification, mais surtout à la relation découverte liant certaines données, ainsi que leur valeur.

Dans l'ordonnancement ces deux aspects ont été pris en considération. Le premier, la classification, pour le choix des règles de priorité à utiliser [22] et le second pour l'analyse et l'extraction de règles de priorité [24]. Ces deux aspects seront détaillés par la suite dans la Section 3.4 portant sur l'état de l'art des travaux de fouille de données pour l'ordonnancement.

3.2.2. Le processus d'ECD

Le processus d'extraction de connaissances à partir de données, comme illustré dans la Figure 3.1, est composé d'un ensemble d'opérations qui offrent la possibilité d'analyser, d'exploiter, ou d'extraire de la connaissance à partir de ces données [167]. Dans le travail de recherche de Fayyad et al. [168], ce processus est désigné comme étant « *non trivial ayant pour but d'identifier dans les données de nouveaux schémas, valides, compréhensibles, et potentiellement utilisables* ».

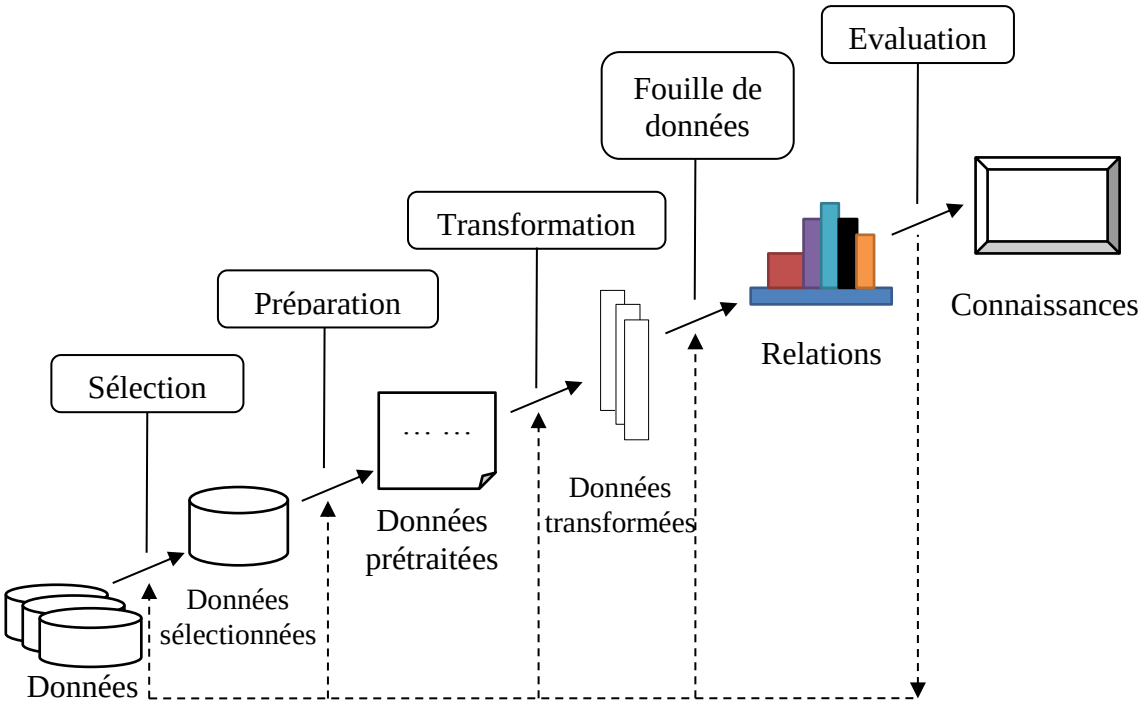

Figure 3.1. Processus d'ECD

Durant l'ECD et en se basant sur une source de données (base de données, fichiers, ordre de passage des tâches, …), un sous-ensemble ou la totalité de ces données est sélectionné afin d'être prétraité avant de pouvoir passer à la fouille de données. Un algorithme (tel que les arbres de décision ou les réseaux de neurones) est alors utilisé dans le but d'extraire certaines relations entre les données ou créer un modèle de décision capable faire de la prédiction. Ces derniers sont ensuite interprétés, évalués, et validés pour de futures utilisations.

Les principales étapes d'une extraction de connaissances à partir de données peuvent être résumées ainsi [169] :

3.2.2.1. Sélection des données

Dans le processus ECD, la première étape cruciale est d'identifier, d'obtenir, et de récolter les données utiles. Il est important que les données considérées puissent être utilisables pour la résolution d'un problème donné. Si par exemple on s'intéresse aux habitudes d'achats des clients dans supermarché, il est impératif que l'ensemble des achats effectués par les clients, avec les produits concernés, les quantités, ainsi que les dates soient mentionnés.

Il est peut-être utile de posséder d'autres informations complémentaires relatives aux clients comme leurs salaires, leurs statuts familiaux, ou le nombre des membres de la famille, mais aussi son environnement. Tous ces aspects peuvent jouer un rôle important dans cette analyse, d'où l'obligation de bien effectuer l'opération d'identification et de sélection des données. Lors de la sélection il est aussi possible de rejeter certaines informations qui n'auront aucun impact lors de l'analyser et de l'extraction, comme par exemple le nom du client ou ses informations bancaires (numéro de carte ou autre).

3.2.2.2. Prétraitement des données

Durant cette étape on s'intéresse plus aux contenus et aux valeurs des données. L'objectif étant de valider, d'explorer, et de nettoyer les données. Plusieurs points doivent être considérés liés au manque de certaines données et de la démarche à entreprendre. Suppression (nettoyage des données) pure et simple des enregistrements non complets ou une prédiction de ces valeurs afin d'en tirer un meilleur profit. Les types d'attributs, texte ou numérique et si c'est nécessaire de discrétiser les valeurs numériques. Une analyse minutieuse doit être réalisée afin d'identifier les incohérences, données manquantes, la corrélation de certains attributs, ou autre et de proposer et de mettre en place les solutions adéquates.

3.2.2.3. Transformation des données

Selon Pinto [169], la transformation des données afin d'obtenir la bonne granularité des données, où la granularité désigne le taux ou le nombre de données pris en considération. Par exemple, l'ensemble des informations sur un achat doivent être représentées par un seul et unique enregistrement. Durant cette phase il est aussi possible de procéder à une dérivation d'attributs. C'est-à-dire qu'un nouvel attribut en combinant plusieurs autres, si c'est faisable.

3.2.2.4. La fouille de données

Une fois le prétraitement terminé, les données manquantes ou erronées supprimées ou corrigées, et/ou de nouveaux attributs insérés. La fouille de données, ou « *data mining* » en anglais, peut être utilisée. La FD est l'étape centrale du processus d'extraction de connaissances à partir de données. Le but est d'extraire de l'information cachée, de la connaissance, ou d'obtenir des modèles capables de faire de la prédiction/classification en utilisant des méthodes intelligentes. Durant cette phase il est aussi important de déterminer le critère ou le moyen pour évaluer le modèle obtenu, comme la fiabilité, la rapidité de construction, la compréhensibilité, l'utilisation, et l'évolutivité [167]. En fonction du problème traité, de l'environnement, des objectifs attendus, ou du type d'utilisation, un critère, ou plus, est favorisé et choisi pour l'évaluation.

Durant ce processus, les données sont découpées en trois ensembles distincts appelés « *Apprentissage* », « *Test* », et « *Evaluation* ». Le premier ensemble est utilisé pour générer un modèle de décision, le second a pour but d'améliorer ce modèle alors que le dernier est utilisé pour son évaluation, son appréciation, et son amélioration. Il est aussi possible de découper les données en deux ensembles seulement, « *Apprentissage* » et « *Test* », la création d'un modèle de décision se fait par validation croisée en se basant sur le premier ensemble alors que le deuxième ensemble sera utilisé pour tester le modèle obtenu.

Il existe plusieurs techniques de fouille de données, comme les arbres de décision, les réseaux de neurones, ou les k-plus proches voisins. Chacune a ses avantages et ses inconvénients. Le problème durant cette phase réside dans l'identification de la meilleure technique. Quand c'est possible il est intéressant de tester différentes techniques, de les évaluer, et de sélectionner la meilleure. Il est aussi utile de noter que chacune des méthodes de fouille de données possède ses propres paramètres qui permettent de calibrer l'algorithme et d'améliorer le critère d'évaluation étudié.

3.2.2.5. Interprétation et évaluation

La dernière étape du processus d'ECD a pour objectif de valider les différents modèles obtenus en utilisant les techniques de fouille de données en utilisant les données réservées aux tests/évaluation. Ces données, ne faisant pas partie de l'ensemble d'apprentissage, permettent d'opérer une évaluation non biaisée. Une matrice de confusion est alors obtenue qui permet d'avoir le nombre de décisions correctes/incorrectes prises par le(s) modèle(s) et de déterminer les faux-positif et les faux-négatifs (évaluation statistique). Il est aussi possible de faire participer un expert humain lors de cette phase (expertise), comme c'est le cas dans le domaine médical par exemple [167], [170].

Lors de l'évaluation statistique, l'objectif est de juger le résultat obtenu et d'estimer sa qualité. La validation peut être obtenue par [167] :

- Calcul des moyennes et variances des attributs,
- Calcul de corrélation entre les champs,
- Détermination de la classe majoritaire pour la classification.

Quant à la validation par expertise, elle est effectuée en se fiant à un expert du domaine qui évaluera et jugera les résultats obtenus par fouille de données. L'expert peut aussi être sollicité afin de vérifier la compréhensibilité du modèle obtenu par fouille de données, afin qu'il puisse être utilisé comme système d'aide à la décision. C'est le cas par exemple dans le domaine médical comme le diagnostic.

L'extraction de connaissance, fouille de données comprise, peut être utilisée dans plusieurs domaines afin d'obtenir de la connaissance cachée potentiellement utile. L'ECD peut ainsi aller au-delà des techniques classiques de statistiques et faire partie intégrante du processus de décision et remplacer dans certains cas l'expert humain en raison de la complexité du problème comme l'ordonnancement.

3.3. La fouille de données

3.3.1. Définition

La fouille de données (data mining) est l'étape centrale du processus d'extraction de connaissances des bases de données (ou à partir des données). Bien que souvent l'ECD et la fouille de données soient confondus, ils sont considérés comme synonymes. La FD est l'analyse de larges quantités de données dans le but découvrir des relations, de résumer de simplifier ces données pour qu'elles soient facilement compréhensibles par l'humain ou la personne possédant ces données [168], [174]. La fouille de données est une nouvelle discipline qui englobe plusieurs topics, comme les statistiques, l'apprentissage machine, l'intelligence artificielle, la reconnaissance des formes, les bases de données et leur gestion. Elle permet de découvrir et d'identifier des relations entre les données, des formes, fournit une description complémentaire de ces données et crée des modèles de décision pour la classification et la prédiction.

3.3.2. Historique

La fouille de données est apparue au milieu des années 1990 aux États-Unis comme une nouvelle discipline regroupant, statistiques, technologies de l'information, intelligence artificielle, et apprentissage machine [175]. Selon Brahimi [167] son origine remonte aux années1960-1970 où les chercheurs faisaient de la fouille de données sans le savoir. Avec la montée en puissance des ordinateurs et les capacités de calcul qu'ils offrent, les chercheurs ont commencé à les utiliser dans divers domaines pour différents types de calculs qui étaient jusque-là impossible à réaliser manuellement. D'une part l'apparition du data mining est liée à cette évolution rapide et soutenue du matériel informatique et de sa capacité de calcul ainsi qu'aux logiciels développés dans le but d'analyser les données. D'autre part, la taille des données dans les entreprises, administrations, hôpitaux liées à leurs activités poussaient les analystes à s'y intéresser de plus en plus et à opter pour des techniques de traitement de données capables d'explorer ces tas de données [175].

Les premières utilisations de la fouille de données se sont focalisées sur le domaine de la gestion, sur l'analyse du comportement des clients afin de mieux appréhender leurs besoins et leurs attentes et s'y adapter [167]. Elle a aussi été appliquée dans le domaine médical pour l'aide au

diagnostic notamment [170]. Dans l'ordonnancement et l'ingénierie industrielle les premiers travaux incluant l'intelligence artificielle et le data mining remontent aux années 1980 et se sont accentués rapidement dans les années 1990 [172] pour l'aide à l'ordonnancement, la sélection des règles de priorité, ou l'extraction de nouvelles règles. Ces aspects seront couverts dans l'état de l'art sur l'utilisation de la fouille de données dans l'ordonnancement dans la Section 3.4.

3.3.3. Utilisation

Le data mining peut être utilisé pour effectuer plusieurs types de tâches en fonction de la méthode utilisée, du problème, et de l'objectif de l'ECD [167], [169], [174]. Il est intéressant de noter que les données utilisées lors de l'ECD sont des enregistrements (descriptions ou faits) et qui sont composés à leur tour de plusieurs champs (attributs ou caractéristiques). A chaque champ est associé une valeur de type texte ou numérique (peut être aussi numérique discrétisée). Parmi les tâches de la fouille de données, on cite :

- **Classification** : elle associe à chaque enregistrement, en se basant sur ses attributs, une classe particulière parmi plusieurs prédéfinies préalablement. Cette classe peut prendre une valeur discrète parmi plusieurs. Par exemple répondre par « *oui* » ou « *non* » à une requête d'une demande de prêt formulée par un client à une banque ; attribuer un document à la classe « *éducation* », « *histoire* », ou « *loisirs* »,
- **Estimation** : elle a pour but d'estimer la valeur d'un champ à valeurs continues à partir de ses autres attributs. Elle peut ainsi être utilisée comme étape de prétraitement afin de compléter les données manquantes avant d'utiliser une autre méthode de fouille de données pour la classification ou la segmentation. Comme exemples d'utilisation, on peut citer l'estimation des revenus des clients, l'estimation et l'évaluation de risques dans les assurances, ou l'estimation des besoins en matières premières dans l'industrie,
- **Prédiction** : en se basant sur des enregistrements historisés, la technique de fouille de données utilisée a pour but de prédire la valeur future d'un champ. Cette tâche est très similaire à l'estimation et à la classification et peut même être regroupée avec l'estimation [169]. On peut l'utiliser comme par exemple pour la prédiction météorologique, pour prédire les valeurs de futures actions (bourse), ou prédire les départs de clients [167],
- **Règles d'association** en anglais « *association rules* » : elles permettent de trouver des relations entre différentes variables. Un exemple type est celui du panier de la ménagère où un client achetant du café et du lait simultanément est susceptible d'acheter aussi du sucre. Ces règles sont ainsi utilisées afin de d'identifier les opportunités de ventes et d'organiser le magasin en groupant les produits qui seront, sûrement, achetés simultanément,
- **Segmentation** en anglais « *clustering* » : elle consiste à créer des classes regroupant des données (enregistrements) jugées similaires entre-elles et différentes de celles des autres groupes, c'est-à-dire que l'intersection des différents « *clusters* » est un ensemble vide. Une analyse est ensuite opérée par un expert du domaine afin d'identifier l'intérêt et la signification des classes. On peut l'utiliser notamment pour la segmentation dans les

images satellitaires afin d'identifier différents groupes d'objets (forêts, champs, bâtiments, ...).

La classification donnée ci-dessus peut varier car différentes tâches se ressemblent et peuvent être groupées ensemble. Comme c'est le cas de l'estimation et la prédiction. Ainsi, Innani [174] identifie six groupes de fonctions de la fouille de données légèrement différents de ceux définis par Brahimi [167] :

- Sélection de caractéristiques : identifie le ou les attributs qui ont le plus d'impacts sur la discrimination ou la prédiction,
- Prédiction/Estimation,
- Classification,
- Découverte de règles : le but est de générer un ensemble de règles simples et facilement compréhensibles pour décrire les différentes classes,
- Segmentation,
- Règles d'association.

3.3.4. Techniques

Dans la littérature il existe un certain nombre de techniques de fouille de données. Ces techniques ne sont pas utilisables pour tous les types de données et/ou tous les types de problèmes/objectifs. Certaines méthodes seront favorisées en raison de leur fonctionnement, de leurs résultats (par exemple compréhensibles par l'humain, comme les arbres de décision, ou non, tel que les réseaux de neurones), ou de la spécificité des données en entrée (par exemple attributs sous forme texte ou numérique). Lors de l'ECD, une seule méthode de data mining ne peut être utilisée pour faire une extraction de connaissances, d'où la nécessité de considérer plusieurs techniques et d'effectuer une étude comparative afin d'identifier la méthode avec le plus grand potentiel. Dans ce qui suit nous citons très succinctement quelques-unes d'entre-elles [165], [175].

3.3.4.1. Agents intelligents

Les agents intelligents, ou « *knowbot* » en anglais, sont des entités abstraites ou physiques, intelligentes, capables d'agir ou de réagir d'une façon autonome. Ils possèdent une représentation partielle de leur environnement et sont capables d'interagir avec d'autres agents. Chaque agent possède son propre objectif et jouit d'un comportement issu de ses observations, de ses compétences, ainsi que l'interaction avec les autres agents de son environnement. Benamar [175] donne ainsi l'exemple de visites et d'achat en ligne, où les agents intelligents suivent et mémorisent les mouvements et comportements de clients et peuvent par la suite être utilisés pour des simulations de ventes de produits ou pour lancer des appels d'offre.

3.3.4.2. Algorithmes génétiques

En plus de l'optimisation (cf. Section 1.7.2.3), les algorithmes génétiques sont aussi utilisés pour la découverte de connaissances. Ils peuvent être utilisés notamment pour la découverte de règles de classification qui seront considérées pour de futures classifications/prédictions. Une

règle peut être sous la forme « *Si **X** alors **Y*** », où *X* est la condition, ou une condition composée (ET/OU) et *Y* et le résultat ou la conséquence de cette règle. Par exemple « ***Si** Attribut1 = Valeur1 **ET** Attribut2 = Valeur2 **alors** Classe = C2* ». Ces règles doivent être d'abord codées, en utilisant un codage binaire par exemple, afin de pouvoir utiliser le croisement et la mutation. L'objectif de l'algorithme génétique et d'améliorer le taux de classification.

3.3.4.3. Analyse des liens

L'analyse des liens (voir Figure 3.2) est inspirée de la théorie des graphes et est dédiée à la prédiction et à la classification. Elle a pour but d'analyser les relations (arcs) entre les nœuds où chaque nœud représente une entité comme par exemple clients et produits. A chaque relation est associé un poids défini par la méthode, celui-ci défini la force et véracité de cette relation. Cette technique est notamment utilisées pour la recherche web, la sécurité, ou pour le domaine médical [167]. La possibilité offerte par l'analyse des liens de représenter graphiquement les données permet une meilleure analyse ainsi qu'une meilleure compréhension par l'humain.

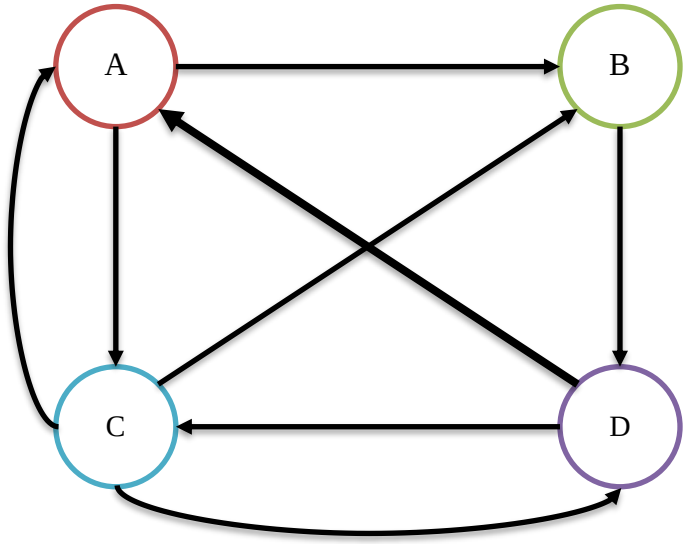

Figure 3.2. L'analyse des liens

3.3.4.4. Arbres de décision

Les arbres de décision sont des organigrammes avec une structure arborescente, où les nœuds représentent des tests ou des attributs, les branches sont les résultats des tests, et les nœuds finaux sont des classes. Les arbres de décision sont très utilisés pour la classification, la description, ou l'estimation. L'algorithme commence d'abord par identifier la variable (attribut) le plus déterminent et découpe la population (valeurs) en sous populations (par exemple attribut supérieur à, inférieur à, ou égal à). Chaque sous-population créée est ensuite analysée à son tour et le même processus est réitéré d'une façon récursive jusqu'à ce que toutes les variables soient traitées. Les règles trouvées sont sous forme d'un arbre, qui en plus de son aspect visuel claire, est facilement compréhensible ce qui explique d'une certaine manière l'engouement pour cette technique. Dans la Section 3.5, les arbres de décision sont présentés d'une manière complète et détaillée.

3.3.4.5. Raisonnement basé sur la mémoire

Le raisonnement basé sur la mémoire, ou « *Raisonnement à base de Cas* » (*RpC*) en anglais « *Case-Based Reasoning* » (*CBR*), est une méthode de classification et de prédiction basée sur la comparaison d'exemples de cas stockés auparavant. Pour chaque nouveau problème une comparaison est effectuée avec l'ensemble des cas précédemment stockés, s'il existe une similarité entre le nouveau problème et un des cas disponibles, alors le système affecte la même classe ou la même prédiction au nouveau cas. Le RpC peut être facilement mis en place et utilisé et supporte tous types de données [167]. En plus et grâce au retour « *feedback* », c'est-à-dire le stockage de nouveau cas, le système peut s'améliorer continuellement en assimilant de nouvelles connaissances et de nouveaux cas.

3.3.4.6. Règles d'association

Cette méthode et souvent référée comme « *panier de la ménagère* », bien que ce dernier soit seulement un exemple souvent illustré afin d'expliquer son principe. Les règles d'association, en anglais « *association rule learning* », ont pour but de trouver des associations en créant un lien entre les variables dans les bases de données. Une règle peut être donnée sous la forme « *si Condition(s) alors Résultat(s)* ». Il est ainsi possible d'associer plusieurs conditions avec un *ET* ou un *OU* par exemple ce qui donnera « *Si X ET Y alors Z* ». Ces règles générées sont ainsi simples et faciles à comprendre et peuvent être intégrées dans les supermarchés par exemple afin de changer de politique de marketing ou de réorganiser le magasin afin de grouper les produits qui seront probablement achetés ensemble. Cette règle « *Si Café ET Lait alors Sucre* » montre que les trois produits café, lait, et sucre seront probablement acheté ensemble. Elles peuvent aussi être utilisées dans les domaines de détection d'intrusion, la production, ou la bio-informatique.

3.3.4.7. Réseaux bayésiens

Les réseaux bayésiens sont un modèle graphique probabiliste regroupant à la fois la théorie des graphes et la théorie des probabilités pouvant représenter les distributions probabilistes sur un ensemble de variables aléatoires [33]. Les réseaux bayésiens permettent de représenter la connaissance sur un domaine d'application d'une manière intuitive et de faciliter la mise en place de modèles performants et clairs. Leur fonctionnement réside dans la création d'un réseau interconnecté de probabilité qui pourrait être utilisé pour l'aide à la décision ou pour le calcul de probabilités conditionnelles.

3.3.4.8. Réseaux de neurones

Les arbres de décision et les réseaux de neurones sont les techniques de fouille de données les plus utilisées [175]. Ils s'inspirent des neurones du cerveau humain, et sont composés de plusieurs neurones (unités de traitement) interconnectés. Ces neurones peuvent appartenir à trois groupes, à savoir, entrée, sortie, ou caché. Leur fonctionnement est lié à la connexion des différents neurones entre eux. Ils peuvent être utilisés pour la classification et la prédiction. Contrairement aux arbres de décisions, les réseaux de neurones sont considérés comme une

méthode difficile à comprendre, ou souvent appelés « *boite noire* » (*blackbox*), car il est difficile de comprendre leur fonctionnement et les résultats trouvés.

3.3.4.9. Statistiques

Solarte [165] présente plusieurs techniques « *classiques* » de statistiques qui peuvent aussi être utilisées pour la partie fouille de données dans l'ECD.

- **Segmentation par secteurs** : ou détection automatique de clusters (classes), consiste à regrouper les enregistrements en groupes (clusters) en fonction de leurs similitudes. Cette méthode permet de mieux comprendre les données lors de l'ECD et d'utiliser d'autres méthodes d'analyse en fonction de ses résultats,
- **Analyse discriminante** : consiste à analyser les attributs d'un enregistrement est de prédire et d'affecter ce dernier dans une des classes déjà prédéfinies. Elle est l'une des plus vieilles méthodes de classification,
- **Régression logistique** : est une technique utilisée pour la prédiction. Elle construit un modèle capable d'expliquer et de prédire les valeurs d'une variable quelconque. Elle est souvent appliquée dans le domaine de la santé afin d'identifier les facteurs liés à une maladie ou déterminer les causes d'un décès, …
- **Série temporelle** : est une suite de valeurs numériques (données périodiques) triées de manière chronologique. Elle a comme but de prédire des classes inconnues en analysant le comportement passé de ces données et en prédisant leur évolution.

3.3.4.10. Traitement analytique en ligne

Selon Benamar [175], le traitement analytique en ligne est un outil utilisé fréquemment préalablement à la fouille de données, car il permet d'analyser et de représenter les données relationnelles dans le but de simplifier la compréhension de ces données. Techniquement elle n'est pas une méthode de fouille de données. Le traitement analytique en ligne « *OnLine Analytical Processing OLAP* » le ROLAP « *Relational OLAP* », MOLAP « *Multidimensional OLAP* » et HOLAP « *Hybrid OLAP* » peuvent ainsi être utilisés comme une étape de prétraitement qui peut s'avérer utile pour l'analyse des données.

3.3.4.11. Visualisation des données

Il est plus facile pour l'être humain de comprendre, d'analyser et de tirer des conclusions à partir de données présentées d'une manière visuelle claire. La visualisation des données permet ainsi de présenter graphiquement les données, en utilisant un changeant les couleurs, les dimensions, ou autres afin que l'opérateur humain puisse extraire certaines relations, formes ou corrélations qui ne sont pas facilement identifiables sous forme de texte. Elle rend ainsi les données complexes plus accessibles, compréhensible, et utilisables. On cite notamment, les histogrammes, les réseaux, la dispersion graphique, et le diagramme de Gantt.

3.4. La fouille de données dans l'ordonnancement

Durant les dernières décennies, les entreprises affichent de plus en plus d'intérêt à la fouille de données car elles jugent qu'elle peut avoir un grand impact sur leur production, leur compétitivité, et par extension leur survie dans un marché en constante évolution. Pour mettre en place un système d'aide à la décision basé sur la fouille de données il est impératif de bien identifier les données susceptibles d'améliorer les processus d'ordonnancement et de production. Ceci en raison des différents types de données existants, les différentes sources, la protection de données, les données erronées, manquantes, ou incomplètes [176].

Dans la littérature on dénombre beaucoup de travaux de recherche ayant inclus le data mining dans la production. Selon Aytug et al. [172], les premiers travaux remontent aux années 1980, et la tendance s'est accentuée durant les années 1990 [176] avec son intégration dans l'ordonnancement et l'ordonnancement dynamique, la maintenance, les prévisions, l'acquisition des matières premières, ou l'aide à la décision entre-autre. La fouille de données permet d'analyser et d'exploiter les données archivées ou simulées afin d'extraire et d'identifier des relations et associations cachées qui peuvent être utilisées pour la résolution ou l'aide à la résolution de nouveaux problèmes.

Dans cet état de l'art, nous nous intéressons tout d'abord à l'utilisation de la fouille de données pour la sélection des règles de priorité pour l'ordonnancement mais aussi pour le réordonnancement et l'ordonnancement dynamique. Ensuite, nous explorons les travaux de recherche ayant intégré le data mining pour la génération de nouvelles règles de priorité.

3.4.1. La sélection des règles de priorité

L'utilisation des règles de priorité (cf. Chapitre 1 et Chapitre 2) est une bonne alternative aux méthodes exactes ou approchées car pouvant trouver une solution quasi-instantanément et peut ainsi être utilisée pour les problèmes complexes comme le Job Shop ou pour l'ordonnancement dynamique. Mais en contrepartie, il existe des dizaines de règles de priorité (cf. Section 2.5), chacune adaptée pour certains types d'ateliers ou développée pour maximiser/minimiser un ou plusieurs critères d'optimisation. Par conséquent, il est nécessaire de tester différentes règles pour chaque problème d'ordonnancement en utilisant la simulation afin d'identifier la ou les meilleures règles de priorité et d'optimiser la fonction objectif. Seulement ce processus peut s'avérer lent même avec des unités de calcul intensif en raison de la taille des problèmes et du nombre de règles existantes. Ceci a donc poussé les chercheurs à réfléchir à une autre alternative qui s'inspirerait d'expériences passées et les mettrait à contribution en vue d'accélérer et d'automatiser le processus de sélection des règles de priorité. La fouille de données et l'intelligence artificielle, en se basant sur les expériences passées, peuvent mettre en place un système d'aide à la décision pour la sélection de règles de priorité. Ce système peut être alors inclus lors de la planification et l'ordonnancement, c'est-à-dire pour la suggestion d'une règle de priorité au début de l'ordonnancement, pour le réordonnancement suite à une perturbation afin de proposer une solution de rechange, ou encore lorsqu'une ressource redevient disponible.

Dans cette perspective, nous citons le travail de recherche de Baykasoğlu et al. [79] qui repose sur un outil appelé *TACO-miner* « *Touring Ant Colony Optimisation* » ainsi qu'un perceptron multicouche pour la création de nouvelles règles composées. Les règles composées sont des règles de priorité déjà existantes et qui sont composées dans le but d'obtenir meilleures solutions aux problèmes d'ordonnancement. Ces nouvelles règles sont basées sur plusieurs paramètres des tâches et/ou de l'atelier comme l'intervalle de temps entre l'arrivée de deux tâches successives, la taille de la file d'attente d'entrée (nombres de tâches pouvant circuler dans l'atelier plus ceux qui sont en attente dans la file), la taille des files d'attentes des machines, l'écart des dates d'échéances, etc. Dans cette approche, le perceptron multicouche est d'abord construit en utilisant des données issues d'expérimentations antérieures et ses poids sont extraits. Ensuite, TACO est appliqué afin d'obtenir les nouvelles règles (*classificiation rules)*. Ces nouvelles règles sont dans une forme « *si* condition *alors* règle » qui peuvent être facilement utilisées par les responsables de la production contrairement au perceptron multicouche qui est considéré comme une boite noire avec des résultats difficilement exploitables. Un exemple d'une règle construite est donné par les auteurs dans l'Équation 3.1 :

$$\textbf{if } InterArrival = 3 \textbf{ then } use\ EDD \tag{3.1}$$

Qui veut dire que si l'intervalle de temps entre l'arrivée de deux tâches successives est égal à 3, alors utiliser la règle « *Earliest Due Date* ». La règle de priorité considérée peut aussi changer en fonction de l'état de l'atelier. Les auteurs effectuent une étude comparative avec d'autres règles de priorité comme FIFO, EDD, et SPT dans un atelier Job Shop et confirment ainsi la supériorité des nouvelles règles en termes de « *Mean Absolute Percentage Error* ».

Dans la même perspective, Aissani et al. [177] ont proposé un outil appelé *CASI*, pour « *Cellular Automata for Symbolic Induction* » qui se base sur Weka développé par Hall et al. [178]. Le but cette approche est de développer un système d'aide à la décision pour la sélection de règles de priorité et ainsi résoudre de nouveaux problèmes d'ordonnancement. L'approche se base sur des simulations et des logs de problèmes Job Shop. Le système développé est composé de trois principaux modules qui sont : la simulation, l'extraction de connaissance, et la prise de décision à base d'agents. La simulation permet de reproduire des problèmes d'ordonnancement et d'alimenter un fichier log avec les informations des tâches et des machines qui sont transformées en utilisant une codification booléenne. Ensuite, de nouvelles règles sont extraites en se basant sur un arbre de décision. Enfin, le système multi-agents peut résoudre de nouveaux problèmes d'ordonnancement en utilisant les nouvelles. Un exemple de règles obtenues est donné comme suit (Équation 3.2) :

$$\textbf{if } (Operating = D2 \textbf{ and } Waiting = A3) \textbf{ then } use\ FIFO \tag{3.2}$$

Les auteurs testent les nouvelles règles de priorité face aux règles FIFO et LIFO dans un atelier Job Shop constitué de 4 tâches et de 2 machines.

Metan et al. [22] ont développé un système capable de déterminer et de sélectionner la meilleure règle de priorité à utiliser à chaque période en fonction de l'état de l'atelier et des différents critères d'optimisation utilisés, ce genre de systèmes est connus sous le nom « *Multi-pass Scheduling* ». D'après les auteurs, en révisant et en changeant les règles de priorité à chaque

période ou évènement, comme les pannes machines, l'arrivées de nouvelles tâches, ou la disponibilité de certaines ressources, permet d'améliorer l'ordonnancement et ainsi maximiser/minimiser la fonction objectif. Le système proposé repose sur la simulation, les arbres de décision, et les graphiques de contrôle de processus statistiques pour l'identification et l'allocation de règle de priorité. Le système se base aussi sur module de surveillance en ligne afin de changer les règles à chaque fois qu'il juge nécessaire (détérioration du critère d'optimisation). Ce type d'approches permet ainsi une certaine adaptabilité du système de production et une tolérance aux évènements imprévus (ordonnancement dynamique). L'approche est testée dans un atelier Job Shop constitué de 4 tâches et de 4 machines avec le temps d'exécution total comme critère d'optimisation. Les résultats indiquent qu'en changeant continuellement les règles de priorité, la valeur de l'objectif est minimisée contrairement à l'utilisation d'une seule règle tout au long de l'exécution (*Single-pass Scheduling*).

Dans une autre contribution à base de fouille de données pour la sélection de règles de priorité, Said et al. [95] concentrent leurs efforts sur les pannes machines. Un arbre de décision utilise des expérimentations antérieures afin d'obtenir des règles sous la forme « *si* condition *alors* règle ». Le but de la nouvelle règle utilisée et de minimiser l'impact des pannes machines. Les attributs considérés pour la construction de l'arbre sont : le temps moyen nécessaire pour la réparation de la machine, le temps moyen entre deux pannes consécutives, la cadence d'arrivée de nouvelles tâches, et le facteur (ratio) utilisé pour calculer les dates d'échéance, en plus de la meilleure règle de priorité obtenue par simulations dans les expérimentations antérieures. Une étude par simulation est réalisée pour démontrer l'utilité d'un tel système en le comparant avec les règles FIFO, SPT, EDD, et MST.

L'utilisation de la fouille de données pour la classification des règles de priorité présente deux principaux avantages. Le premier est lié aux choix des règles de priorité. En effet, en raison du nombre élevé de règles de priorité existantes dans la littérature, il est nécessaire de tester et de simuler toutes les règles afin d'identifier celle qui maximise ou minimise le(s) critère(s) d'optimisation considéré(s). Cette tâche en utilisant uniquement la simulation peut s'avérer très couteuse en temps CPU. La fouille de données en se basant sur des données d'ordonnancement historiques offre la possibilité de sélectionner une règle à utiliser en fonction des attributs des machines ou des tâches des nouveaux problèmes. Ce processus, d'autant plus, est effectué en ligne contrairement aux approches basées sur la simulation.

Le second avantage de la fouille données réside dans la possibilité de résoudre les problèmes d'ordonnancement dynamiques où l'information sur les tâches n'est pas disponible (arrivée de nouvelles tâches, temps d'exécution inconnus, ou changement des temps d'exécution ou priorités), mais aussi pour faire l'ordonnancement réactif et la prise en charge des pannes machines. Contrairement aux méthodes exactes et approchées, où, à chaque fois qu'un évènement inattendu survient, il est nécessaire de suspendre l'exécution des tâches pendant une certaine durée et lancer la méthode de résolution afin d'obtenir une nouvelle solution alors qu'en même temps la dégradation de l'objectif va s'aggraver et qu'il est nécessaire de répéter la même opération lorsque la situation redevient normale après la levée de l'évènement ayant entrainé la perturbation. Par contre en utilisant les règles de priorité le système n'a pas besoin d'attendre qu'un nouveau séquencement des tâches soit trouvé, et les règles peuvent définir des

priorités en temps réel, et où le but de la fouille de donnée est de s'assurer que les règles utilisées durant la perturbation, sont bonnes.

3.4.2. L'extraction de règles de priorité

Toujours dans le cadre de l'intégration de la fouille de données dans l'ordonnancement, les chercheurs se sont intéressés à l'extraction de règle de priorité. Ce processus consiste à explorer, analyser, et extraire de la connaissance à partir de solutions obtenues par d'autres méthodes de résolutions. La connaissance est sous forme de règles « *si-alors* » qui peuvent être utilisées pour la résolution de nouveaux problèmes d'ordonnancement héritant à la fois des avantages de règles de priorité, tel que la rapidité ou la simplicité, ainsi que l'efficacité et les bons résultats des méthodes exactes ou méta-heuristiques. Le but des nouvelles règles de priorité est de tenter reproduire le plus efficacement possible le fonctionnement et le comportement des autres méthodes de résolution permettant ainsi de résoudre des problèmes d'ordonnancement complexes tel que le Job Shop ou le Single Machine en temps-réel avec des résultats proches des méta-heuristiques et pouvant même dépasser d'autres heuristiques comme le FIFO, LIFO, ou EDD.

Il existe un grand nombre de travaux de recherche qui se sont penché sur cette alternative. Parmi les premiers travaux on cite notamment Koonce & Tsai [73] qui ont exploré et reproduit le fonctionnement et la logique d'un algorithme génétique et extrait de nouvelles règles de priorité. Un algorithme génétique est développé d'abord pour résoudre des problèmes de type Job Shop et les solutions obtenues sont par la suite reformatées et préparées en vue de leur utilisation pour l'extraction de connaissances. Les auteurs utilisent « *Attribute Oriented Induction* » comme technique de fouille de données qui tente de représenter la relation entre la séquence des opérations et leurs attributs et de générer un certain nombre de règles. Les auteurs effectuent leurs expérimentations sur un benchmark Job Shop constitué de 6 machines et de 6 tâches. Il est utile de noter que les auteurs ont opté pour la discrétisation des attributs des tâches à savoir, le temps d'exécution et le temps d'exécution restant en trois valeurs, « *court* », « *moyen* », et « *long* ».

Chebel Morello et al. [179] se basent sur le processus d'ECD pour extraire de nouvelles règles de priorité. L'approche proposée est décomposée en trois phases. La première consiste à stocker les données qui dans ce cas sont des problèmes d'ordonnancement d'ateliers flexibles (FMS). Chaque problème est généré aléatoirement consistant d'un certain nombre de tâches, d'opérations, de machines, et de temps d'exécutions aléatoires. Ces problèmes sont résolus en tenant compte de plusieurs critères d'optimisation comme le temps d'exécution total ou le temps d'attente moyen des tâches. Les méthodes considérées pour la résolution sont SPT, SIO, LAWR, FIFO, et CR, et la meilleure méthode est identifiée et stockée avec sa séquence obtenue. Quant à la seconde phase, et comme dans tout processus d'ECD, elle a pour but d'analyser et de prétraiter les données. Les auteurs utilisent un filtre glouton et une sélection d'attributs les plus représentatifs. Un algorithme C4.5 est utilisé par la suite pour générer un arbre de décision et par conséquent de nouvelles règles de priorité sous la forme « *si-alors* ». Les expérimentations sont effectuées sur un atelier Flow Shop de 20 tâches et 4 machines avec la moyenne de temps d'exécution comme objective à optimiser.

Li & Olafsson [180] ont développé une approche basée sur la fouille de données pour résoudre le problème d'ordonnancement à Machine Unique. Dans cette approche le problème Single Machine est d'abord résolu en utilisant la règle de priorité « *Longest Processing Time* », ensuite la solution obtenue est transformée en un tableau de données en comparant toutes les tâches une à une et en définissant si une tâche i doit être exécutée avant une tâche j. Les quatre attributs des tâches utilisés sont la date d'arrivée, le temps de début d'exécution effectif, le temps d'exécution, et le temps de fin d'exécution. Le nouveau tableau de données est ainsi constitué de neuf colonnes, les quatre attributs de chaque tâche (deux tâches comparées à la fois) en plus d'une classe pour déterminer, en se basant sur la séquence obtenue par LPT, si la tâche i est opérée avant la tâche j. Les auteurs utilisent ensuite un algorithme d'arbre de décision C4.5 pour obtenir les nouvelles règles de priorité suivant l'Équation 3.3 :

$$\textbf{if } ProcessingTime1 \leq 7 \textbf{ then } dispatch\ job\ 2\ first \tag{3.3}$$

Durant les expérimentations, les auteurs considèrent des problèmes Single Machine à 120, 160, et 200 tâches prouvant le faible écart, en termes de makespan, entre les nouvelles règles extraites et LPT utilisé initialement.

Une autre extension de ce travail proposée par les mêmes auteurs [56] consiste à résoudre des problèmes Single Machine en utilisant un algorithme génétique. Lors de la comparaison des tâches un-à-un, quatre nouveaux attributs sont rajoutés, « *job 1 release earlier* », « *job 1 due earlier* », « *job 1 PT lower* », et « *job 1 weight higher* ». Le but de ces nouveaux attributs construits et d'améliorer l'apprentissage et d'obtenir de nouvelles règles de priorité plus efficaces et qui imitent au mieux l'algorithme génétique.

Suivant le même procédé, Geiger et al. [123] et Geiger & Uzsoy [181] proposent un système dénommé SCRUPLES « *SCheduling RUle discovery and Parallel LEarning System* » combinant des algorithmes génétiques et intelligence artificielle pour l'extraction de nouvelles règles de priorité. Selon les auteurs, ces règles donnent de bons résultats dans des ateliers où les machines peuvent opérer plusieurs opérations en même temps (batch processor scheduling). Durant les expérimentations, les problèmes sont aléatoirement générés avec un nombre de tâches égal à 10 ou 20 et avec C_{max} comme objectif. Le système proposé est comparé avec d'autres règles de priorité à savoir, EDD, MST, MDD, COVERT, et ATC.

Ingimundardottir & Runarsson [182] quant à eux considèrent une « *régression logistique* » pour la génération de nouvelles règles de priorité en se basant des solutions optimales de problème Job Shop avec minimisation de makespan. Les auteurs conduisent des expérimentations avec des problèmes job générés aléatoirement et à partir des solutions de ce problème génèrent les nouvelles règles qui sont à leur tour comparées avec d'autres heuristiques SPT, MWRM, et LWRM (Most/Least Work ReMaining).

Balasundaram et al. [69] concentrent leurs efforts sur l'atelier Flow Shop. Le problème composé de 5 tâches et de 2 machines est résolu d'abord en utilisant une heuristique, proposée par Raghu & Rajendran [159], qui programme la tâche i avant la tâche j si la valeur du temps d'exécution total est inférieure à celle obtenu en faisant passer la tâche j avant i. Le résultat de l'ordonnancement est transformé en un tableau comme présenté par Li & Olafsson [180] est un

arbre de décision est construit pour extraire de nouvelles règles de priorité. Les attributs considérés lors de l'extraction sont : les temps d'exécutions des tâches i et j sur les machines 1 et 2 ($J_iP_1, J_iP_2, J_jP_1,$ et J_jP_2), t_{j1} et t_{j2} qui représentent les temps d'exécution (somme) des tâches sur les machines 1 et 2. Deux autre attributs sont aussi considérés à savoir, différence P_1 ($J_iP_1 - J_jP_1$) et différence P_2 ($J_iP_2 - J_jP_2$) entre les deux tâches i et j.

Un autre travail intéressant est présenté par Shahzad & Mebarki [91] où ils utilisent un algorithme de recherche tabou pour la résolution d'un problème Job Shop à 6 tâches et 6 machines. La séquence est ensuite prétraitée et un algorithme C4.5 est utilisé pour extraire de nouvelles règles de priorité. D'autres versions modifiées mais suivant toujours le même principe ont aussi été présentées [24], [82]. Le but des nouvelles règles de priorité est de reproduire le plus efficacement possible la recherche tabou et ainsi gagner en temps CPU lors de la résolution de nouveaux problèmes d'ordonnancement. Dans la même perspective, Wang et al. [183] utilisent aussi la fouille de données est plus spécifiquement l'algorithme C4.5 pour la génération de règles de priorité basées sur des solutions existantes de problèmes d'ordonnancement

L'utilisation de la fouille de données pour l'extraction de nouvelles règles de priorité permet ainsi de reproduire, jusqu'à un certain point, le comportement d'autres méthodes de résolutions plus efficaces, comme les algorithmes génétiques [56], la recherche tabou [91]. Cette approche peut aussi être étendue aux méthodes exactes, car lors de l'extraction cette procédure a besoin d'un certain nombre de solutions de problèmes d'ordonnancement pour apprendre et imiter. Les nouvelles règles obtenues tirent profit des méthodes de résolutions exactes ou méta-heuristiques permettant ainsi de résoudre et trouver de bonnes solutions à de nouveaux problèmes d'ordonnancement, mais aussi, le faire en temps-réel comme tout autre heuristiques (règle de priorité). Il est donc intéressant de considérer ce procédé pour l'ordonnancement dynamique ainsi que pour la gestion de perturbations.

Il est utile de noter aussi que plusieurs états de l'art sur l'utilisation de la fouille de données dans l'ordonnancement ont été proposés, parmi lesquels on cite notamment celui d'Aytug et al. [172], ou l'apport de l'apprentissage machines dans les ateliers de production flexibles par Priore et al. [184]. Fonseca & Navaresse [185] quant à eux se sont intéressés aux réseaux de neurones pour la simulation des ateliers Job Shop. D'autres états de l'art plus récent sont présentés par Harding et al. [171], Choudhary et al. [186], Ismail et al. [176], Anderson et al. [173], ou Köksal et al. [187]. Enfin, Chaari et al. [6] présentent quelques travaux basés sur la fouille de données pour l'ordonnancement réactif et la gestion des perturbations.

3.5. L'arbre de décision

L'état de l'art (cf. Section 3.4) sur l'utilisation de la fouille de données dans l'ordonnancement permet de noter la forte présence des arbres de décision pour l'extraction de connaissances. Cet engouement est très souvent motivé par la simplicité et la rapidité des algorithmes de construction d'arbres de décision, mais surtout en raison des résultats obtenus qui sont sous la forme « *si-alors* » qui sont compréhensibles par les opérateurs humains et peuvent ainsi être implémentés et intégrés lors de l'ordonnancement, contrairement à d'autres méthodes comme

les réseaux de neurones, qui sont très complexes et considérés comme une « *boite noire* ». L'algorithme C4.5 est la méthode la plus considérées dans les différentes expérimentations [24], [180], [183]. Nous avons donc jugé utile de développer cet algorithme, mais il est d'abord nécessaire de discuter ID3 avant car C4.5 est une amélioration de ce dernier.

L'arbre de décision (*ADD*) est une technique de classification supervisée proposée par Quinlan [188] qui est considéré comme l'un des pionniers de ce type de méthodes. L'ADD est un outil automatisé, qui analyse un ensemble de données (enregistrements) préalablement classifiés et génère une structure arborescente, appelé « *ensemble d'apprentissage* ». Cette structure peut ensuite être utilisée pour la classification et/ou la prédiction des nouveaux enregistrements. Le processus consiste à isoler et partitionner le plus efficacement possible les enregistrements d'une même classe. Ces algorithmes sont très prisés en raison de leur simplicité, leur performance, et leur lisibilité due à la nature arborescente des résultats. Lorsqu'un nouvel enregistrement doit être classé, l'arbre de décision est parcouru à partir de la racine et en utilisant les valeurs de l'enregistrement, jusqu'à arriver à une feuille qui représente la classe. Chaque nœud de l'arbre est une décision atomique, et chaque réponse permet de se diriger vers tel ou tel nœud fils ; une feuille représente la réponse donnée par l'arbre à la classification de l'enregistrement.

3.5.1. Types de données

Durant la phase construction de l'arbre de décision, deux types de variables (attributs) peuvent être rencontrées [167], [189], quantitatives et qualitatives :

- Quantitative : un attribut est dit quantitatif (numérique) s'il peut prendre une valeur issue d'un ensemble fini, infini, ou défini dans un intervalle. Un attribut quantitatif peut être aussi continu ou discret :
    - Continu : si la valeur que peut prendre un attribut est réelle, composée d'un ensemble infini d'éléments, comme par exemple la taille ou le poids, ou le temps d'exécution des tâches,
    - Discret : dans le cas où l'attribut prend une valeur d'un ensemble finis d'éléments, comme par exemple les notes des élèves, ou dans le cas de l'ordonnancement, les numéros des machines. Un attribut quantitatif discret peut aussi être considéré comme étant qualitatif.
- Qualitative : on parle d'attributs qualitatifs s'ils prennent des valeurs non numériques, comme le pays, la ville, ou la priorité d'une tâche (élevée, normale, faible).

Selon Hawarah [189], on peut aussi parler de données numériques dites « *ordinales* » si les valeurs peuvent être ordonnées, comme *{court, moyen, long}* dit ordinal « *symbolique* », ou peuvent être ordinales « *numériques* » dans le cas où les éléments sont numériques et peuvent être triés selon un ordre croissante/décroissant {*0, 1, 2, 3*}. D'autres attributs sont susceptibles d'être rencontrés dits « *binaires* », par exemple si une tâche doit être exécutée sur machine quelconque {oui, non} ou {0, 1}. On parle aussi d'attribut « *nominal* » si l'ordre n'a pas d'importance comme le type de tâches à exécuter (famille), ou la situation familiale d'un individu {marié, célibataire, …}.

3.5.2. Construction

La construction d'un arbre de décision est tributaire des attributs des enregistrements utilisés (ensemble d'apprentissage). L'ensemble d'enregistrements est divisé d'une manière récursive en utilisant différents tests jusqu'à aboutir à des feuilles contenant uniquement, idéalement, des enregistrements appartenant à une seule et unique classe. Pour diviser l'ensemble d'apprentissage il est impératif de définir les attributs qui minimisent l'impureté dans les sous arbres, c'est-à-dire, qui maximisent l'information apportée par les réponses. Une fois un attribut possédant le plus grand gain d'information identifié, il est choisi comme le prochain nœud de l'arbre et le processus est répété sur chaque nœud restant. Le signal d'arrêt est donné si l'arbre contient uniquement des exemples issus d'une même classe, ou s'il n'est plus possible d'améliorer l'arbre.

La construction de l'ADD repose sur le principe de « *diviser pour régner* » (*divide and conquer*) qui est constitué de trois phases [167], [189] :

- Transformer un nœud en une feuille si toutes les instances appartiennent à la même classe, ou si le taux d'erreur est inférieur à un certain seuil toléré,
- Sélectionner un test à associer à un nœud,
- Affecter la classe majoritaire à une feuille.

Pour définir l'attribut test, il est nécessaire de calculer l'apport des attributs en utilisant l'Équation 3.4 de l'entropie :

$$H(C_i) = Entropie(P_i) = -\sum_{i=0}^{i=k} P_i log_2 P_i \tag{3.4}$$

P_i est la probabilité de retrouver la classe C_i dans l'ensemble d'apprentissage. L'entropie est égale à un s'il n'existe qu'une seule classe dans l'ensemble d'apprentissage, et est égale à $log_2 k$ si les cas k classes sont équiprobables. Lors de l'opération de partitionnement des données, on doit calculer l'information gagnante par attribut en se basant sur la formule (voir Équation 3.5) d'entropie conditionnelle [188] :

$$H(C_i|A) = \sum_{j=1}^{j=n} P(A = a_j) \times H\left(C_i|a_j\right) \tag{3.5}$$

Où n est le nombre de valeurs possibles a_j pour l'attribut A. Le gain d'information est enfin donné par l'Équation 3.6 :

$$Gain(A, C_i) = IM(A, C_i) = H(C_i) - H(C_i|A) \tag{3.6}$$

L'attribut qui maximise le gain (minimise en même temps l'entropie), est choisi pour créer le prochain nœud. On crée ensuite un sous-ensemble de données selon les valeurs possibles a_i pour l'attribut en question. Le processus est répété récursivement jusqu'à atteindre un critère d'arrêt. Le pseudocode de l'algorithme peut être donné comme suit [189] :

Algorithme 3.1 Construction-ADD(S, R)

Entrée : Ensemble d'apprentissage S, Liste d'attributs R
Sortie : Arbre de décision

1 : **Si** critère d'arrêt **alors**
2 : Créer un arbre avec la classe la plus probable
3 : **Sinon**
4 : Identifier le meilleur attribut A dans R
5 : Fractionner S en n sous-ensembles suivant les valeurs de A
6 : Créer un nœud A
7 : **Pour** chaque valeur de A
8 : Construction-ADD(S_i, $L - \{A\}$)
9 : **Fin pour**
10 : **Fin Si**
11 : **Renvoyer** l'arbre de décision

Pour construire l'arbre de décision il existe dans la littérature une panoplie de méthodes plus ou moins différentes présentant certains avantages et/ou inconvénients par rapport à d'autres. Nous suggérons la consultation du livre de Quinlan [188] et la thèse de Hawarah [189] pour plus de détails. Dans ce qui suit nous détaillons l'algorithme ID3 en soulignant ses limites et l'apport de C4.5 pour y remédier.

3.5.3. L'algorithme ID3

L'algorithme ID3 permet de construire un arbre de décision en utilisant un ensemble S d'enregistrements dédiés à l'apprentissage et où chacun est composé d'un ensemble d'attributs R (attribut classe compris).

Algorithme 3.2 ID3(S, R)

Entrée : Ensemble d'apprentissage S, Liste d'attributs R
Sortie : Arbre de décision

1 : **Si** S est vide **alors**
2 : **Renvoyer** un simple nœud de valeur Echec
3 : **Fin Si**
4 : **Si** S est composé uniquement d'enregistrements avec une même classe **alors**
5 : **Renvoyer** un nœud de cette classe
6 : **Fin Si**
7 : **Si** R est vide **alors**
8 : **Renvoyer** un nœud avec la valeur la plus fréquente de l'attribut cible trouvé dans S
9 : **Fin Si**
10 : $D \leftarrow$ l'attribut possédant le plus grand gain (D, S) parmi les attributs de R
11 : $\{d_j \; avec \; j = 1, 2, .., m\} \leftarrow$ les valeurs des attributs de D
12 : $\{S_j \; avec \; j = 1, 2, .., m\} \leftarrow$ les sous-ensembles de S composés uniquement d'enregistrements avec $D = d_j$
13 : **Renvoyer** un arbre de décision avec D comme racine et des arcs étiquetés par $d_1, d_2, \ldots, d_m$ allants vers les sous arbres ID3(S_1, $R - \{D\}$), ID3(S_2, $R - \{D\}$), …, ID3(S_m, $R - \{D\}$)

L'algorithme ID3 est efficace, néanmoins, pose un certain nombre de problèmes [189]. D'abord il nécessite un temps de calcul assez important et qui augmente rapidement en fonction de la taille des données (nombre d'attributs et/ou enregistrements). L'algorithme se fie aux probabilités des attributs qui ont le plus de chance d'aiguiller les résultats, donc n'applique pas une recherche exhaustive. L'utilisation du « *OU* » ou « *XOR* » présente un certain nombre de difficultés aussi liées notamment à la redondance des tests durant la création des sous arbres. A ça s'ajoute l'impossibilité de traiter des enregistrements avec valeurs manquantes. En plus, certaines parties de l'arbre ne sont jamais visitées et contribuent par conséquent à la complexité de l'arbre inutilement. Aussi, les attributs doivent être discrétisés ce qui peut être difficile à mettre en place. Par exemple on doit discrétiser les temps d'exécution, or, la question qui se pose : Quelles valeurs pour le découpage ? Pourquoi ces valeurs ? Comment le justifier ?

3.5.4. L'algorithme C4.5

Le C4.5 développé aussi par Quinlan [188] apporte certains éléments de réponse et permet ainsi et remédier à certains problèmes d'ID3. Tout d'abord C4.5 permet de gérer les données manquantes en évaluant le ratio de gain de l'attribut (voir l'Équation 3.7) où la donnée est manquante et en considérant uniquement les enregistrements où les valeurs de l'attribut en question sont disponibles.

- **Variables continues**

Quant aux valeurs à l'intervalle continu, l'algorithme procède comme suit. Soit R_i un attribut à valeur sur intervalle continu. Les valeurs (par regroupement) dans l'ensemble d'apprentissage S sont données par : $V_1, V_2, \dots, V_n$. Pour chacune de ces valeurs les enregistrements sont partitionnés en deux groupes, ceux avec une valeur supérieure ($> V_i$) et ceux avec une valeur inférieure ou égale à V_i ($\leq V_i$). Pour chacune des partitions on calcule le ratio de gain (voir l'Équation 3.7) et on choisit celle qui maximise ce dernier.

- **Ratio de gain**

Le gain donné par l'Équation 3.6 souffre aussi de certains problèmes soulevés dans la littérature [189] liés au fait qu'il favorise les attributs qui possèdent un nombre important de valeurs. Par exemple pour un attribut R_i qui a une valeur différente pour chaque instance aura un gain maximal. Comme solution, Quinlan [188] a introduit la notion de « *Gain Ratio* » (voir l'Équation 3.7) afin d'y remédier :

$$Gain(R_i, T) = Gain(R_i, T) / SplitInfo(R_i, T) \tag{3.7}$$

Où $SplitInfo(D, T) = I(\frac{|T_1|}{T}, \frac{|T_2|}{T}, \dots, \frac{|T_m|}{T})$ avec $\{T_1, T_2, \dots, T_m\}$ la partition T induite par la valeur de R_i.

- **Elagage**

Parmi les problèmes rencontrés lors de la construction d'un arbre de décision figure sa taille. En effet, le taille de l'arbre grandit d'une manière linéaire en fonction du nombre d'enregistrements dans la base d'apprentissage [189]. Deux familles de méthodes d'élagage

peuvent être considérés [188]. La première connue sous le nom de « *pré-élagage* » a pour but de suspendre la construction de l'arbre de décision à l'avance même s'il existe des feuilles encore non-pures (le nœud contient des enregistrements de plusieurs classes). Ce processus est effectué aussi lorsque le taux d'erreur attendu est jugé supérieur à celui d'une feuille, ou supérieur à un certain seuil. La deuxième famille « *post-élagage* » regroupe les méthodes qui permettent de simplifier et de réduire la taille de l'arbre après sa construction. Si le remplacement d'une partie de l'arbre permet de réduire le taux d'erreur alors cette dernière est remplacée par une feuille.

L'élagage permet aussi de résoudre les problèmes liés au sur-ajustement, ou « *over-fitting* » en anglais, augmentant considérablement le taux d'erreur de l'arbre de décision. Ce problème est souvent rencontré lorsque les données utilisées pour l'apprentissage sont bruitées, en raison du manque de certains exemples importants ou lorsque les exemples sont trop spécifiques.

3.6. Conclusion

Dans ce chapitre nous avons présenté brièvement le processus d'extraction de connaissances à partir de données ainsi que les techniques de fouille de données utilisées et quelques domaines d'application. Nous avons mis en évidence l'apport de la fouille de données dans à l'ordonnancement via la classification des règles, l'extraction de règles, et pour l'ordonnancement dynamique. Le data mining permet en effet, d'automatiser et de simplifier la sélection de règles de priorité dans les ateliers de production et permet de faciliter la résolution des problèmes d'ordonnancement. Pour se faire, il se base sur des expérimentations passées obtenues par simulation pour créer un modèle de décision.

La fouille de données, comme présenté dans l'état de l'art, permet aussi d'analyser et d'imiter le comportement de méthodes de résolutions méta-heuristiques tirant ainsi avantage de ces méthodes pour la minimisation/maximisation des objectifs d'ordonnancement, tout en permettant de le faire en temps réel, une fonctionnalité héritée des règles de priorité. Durant la génération de modèles de décision, les chercheurs concentrent leurs efforts sur les arbres de décision ID3 et C4.5 en raison de leur simplicité, facilité de compréhension, d'utilisation, et d'implémentation des résultats qui sont sous la forme « *si-alors* ». En analysant les différents travaux de recherche et suivant les recommandations des auteurs, il est clair que des améliorations peuvent être encore apportées, que ça soit pour la classification des règles ou pour l'extraction. Cet état de fait nous a motivé à proposer plusieurs contributions présentées dans les chapitres suivants.

CHAPITRE

4. Sélection des Règles de Priorité en Temps-Réel par Fouille de Données et Simulation

Résumé : L'identification de la meilleure règle de priorité pour la minimisation du temps d'exécution total dans un problème d'ordonnancement Job Shop est une tâche difficile. Ceci parce qu'aucune règle ne surpasse les autres dans différents scénarios, rendant la sélection de la meilleure règle de priorité lente et requérant beaucoup de temps. Dans le cadre de cette thèse, une nouvelle approche est proposée combinant règles de priorité, fouille de données, et simulation. Son objectif est d'affecter en temps-réel un ensemble de règles de priorité à chaque machine du système de production tout en minimisant le temps d'exécution total. Les expérimentations montrent que cette approche est effective et obtient de bons résultats en termes de temps d'exécution et de makespan.

Mots-clés : Intelligence Artificielle · Fouille de Données · Classification Multi-Label · Temps-Réel · Simulation · Règles de Priorité · Job Shop · Makespan

4.1. Introduction

L'ordonnancement est le principal point de blocage dans la planification des systèmes de production. Son objectif est d'allouer les machines disponibles aux tâches nécessitant ces ressources pour être achevées d'une manière efficace tout en trouvant un compromis entre dates de fin d'exécution et des contraintes des tâches et/ou machines. Pour résoudre de tels problèmes, et comme annoncé dans la Section 1.7, de nombreuses techniques ont été proposées, allants des méthodes exactes comme le Branch & Bound [12] aux méta-heuristiques tel que les algorithmes génétiques [190] ou la recherche tabou [77].

En parallèle, les chercheurs se sont aussi intéressé aux heuristiques pour la résolution de problèmes d'ordonnancement, communément appelées « *règles de priorité* » [16], [26], [76] (cf. Chapitre 2). Ces heuristiques affectent une priorité à toutes les tâches en attente sur la file d'une machine quelconque. Ce processus est fait quasi-instantanément en raison de la simplicité de ces heuristiques ainsi que le faible temps CPU requis, apportant ainsi un plus à la robustesse du système de production et sa responsivité en cas d'arrivée de nouvelles tâches ou en cas de pannes machines. En tant que tel, les règles de priorité peuvent être considérées comme étant une alternative viable pour faire face à des problèmes d'ordonnancement difficiles comme le Job Shop (cf. Section 1.4.3). Il est d'autant plus intéressant de se focaliser sur les règles de priorité pour l'ordonnancement dynamique où l'information sur les tâches ou sur les machines n'est pas disponible jusqu'à l'arrivée des tâches, comme les dates d'arrivée, les temps d'exécution, ou leurs séquences, rendant ainsi obsolète les techniques classiques de résolution (exactes et méta-heuristiques).

Cependant, la principale difficulté rencontrée lors de l'utilisation des règles de priorité est liée au fait qu'aucune règle ne surpasse les autres pour résoudre différents problèmes, c'est-à-dire que la meilleure règle de priorité peut différer d'un problème à un autre. Afin d'identifier celle-ci toutes règles doivent être simulées et évaluées. Mais comme l'atelier peut contenir un nombre important de machines et que le nombre de règles à tester est lui aussi élevé, il devient difficile, voire impossible dans certains cas, de s'appuyer uniquement sur la simulation pour l'identification de la meilleure règle de priorité. Par ailleurs la simulation devient inutile lorsqu'il est nécessaire de prendre des décisions en temps-réel [22].

Afin de résoudre ce problème, la fouille de données peut être envisagée, car elle est capable d'identifier et de sélectionner la meilleure règle de priorité à appliquer en se basant sur des expérimentations passées. Comme discuté dans la Section 3.4, la fouille de données est largement sollicitée pour les problèmes d'ordonnancement [171], [176], [186]. Elle offre de nouvelles possibilités et ouvre de nouveaux horizons pour l'amélioration des processus d'ordonnancement et de planification ainsi que la responsivité du système en lui permettant de prendre des décisions en temps-réel. Suivant cette perspective, dans cette approche [191] la fouille de données est combinée avec la simulation afin de retrouver les meilleures règles de priorité pour planifier les tâches en standby sur les files d'attente des machines. Concrètement, l'objectif de cette approche est de développer un système qui, en combinant simulation et fouille de données, automatise la sélection des règles de priorité et minimise le temps total d'exécution (makepsan) pour des problèmes d'ordonnancement de type Job Shop. L'approche [191] repose

sur l'hypothèse que le changement de règle d'une machine à une autre, c'est-à-dire l'allocation de différentes règles aux machines, au lieu d'utiliser une règle pour toute les machines, permet d'améliorer la fonction objectif.

4.2. Job Shop

Le but de l'ordonnancement est d'optimiser un ou plusieurs objectifs en allouant efficacement les ressources disponibles aux tâches tout en respectant les différentes contraintes. Pour réaliser une tâche, une ou plusieurs machines sont requises en fonction de la séquence de la tâche et de la configuration du système de production. Lorsqu'une machine est disponible, elle est dédiée pour une tâche durant une certaine période de temps jusqu'à ce que cette dernière soit terminée. Dans le cas du Job Shop, l'atelier est composé de plusieurs machines, doit traiter plusieurs tâches avec plusieurs possibilités de routage et gérer un grand nombre de contraintes ; tout ceci rend le problème d'ordonnancement dans le Job Shop beaucoup plus complexe que pour les autres types d'ateliers. Le Job Shop (cf. Section 1.4.3) a fait l'objet de nombreuse recherches récentes et une panoplie de méthodes de résolution ont été proposées [24], [93], [99], [192], montrant l'intérêt porté par les chercheurs à ce problèmes dû aux défis et difficultés qu'il présente.

Un problème Job Shop comprend M machines et N tâches. Chaque tâche possédant sa propre séquence, où une séquence est une suite prédéterminée (ou non dans le cas d'un Job Shop dynamique) d'opérations devant être opérées sur un certain nombre de machines suivant une certaine succession. Dans la littérature différentes variantes du Job Shop sont étudiées, avec des contraintes supplémentaires liées aux tâches et/ou aux machines, ou encore au coût de transport d'une machine à une autre. Dans cette approche, les hypothèses suivantes sont formulées :

- Les machines ne peuvent exécuter qu'une seule opération à la fois,
- Une tâche ne peut être opérée qu'une seule fois par une machine,
- L'exécution d'une opération ne peut être suspendue et reprise plus tard.

Chaque tâche possède une liste d'opérations prédéfinies $O_i = \{O_{i,0}, O_{i,1}, \dots, O_{i,j}\}, j = \{0, 1, \dots, N_i - 1\}$ à exécuter sur un ensemble de machines défini préalablement. $O_{i,j}$ dénote l'opération j de la tâche i et N_i est le nombre d'opérations de cette tâche.

- Chaque opération $O_{i,j}$ est exécutée sur une machine m_{ij} qui représente son numéro d'identification,
- Chaque opération a un temps d'exécution supérieur à zéro $p_{ij} > 0$,
- Le temps d'exécution de la tâche est donné par $p_i = \sum_{j=0}^{N_i - 1} p_{ij}$.

Quand une tâche est terminée (l'ensemble des opérations requises finies), son temps de fin d'exécution est mis à jour $C_i = t$ où t est l'heure actuelle. L'objectif de l'ordonnancement dans cas-ci est de trouver une solution faisable (qui respecte les différentes contraintes) et qui termine l'exécution des tâches le plutôt possible, c'est-à-dire réduire le temps d'exécution total (makespan) C_{max} décrit par l'Équation 4.1 :

$$C_{max} = max_{i=0,N-1} C_i \quad (4.1)$$

Dans d'autres variante du problème Job Shop il est possible considérer les dates de début d'exécution S_i, les dates d'échéance d_i, ou leurs priorités w_i (cf. Section 1.2.1). Aussi, d'autres objectifs peuvent être pris en considération notamment ceux liés au retard T_{max} ou au décalage L_{max} (cf. Section 1.2.4). Dans cette contribution, les tâches sont supposées arriver au moment $t = 0, \forall\, i \in \{0, 1, \dots, N-1\}, r_i = 0$. Les dates d'échéance, les temps de préparation, et les perturbations ne sont pas pris en considération.

Il a été démontré dans la littérature que le Job Shop est un problème NP-Difficile (NP-Hard), rendant ainsi l'utilisation de méthodes exactes pour la résolution obsolète et inefficace [7]. Ceci est dû notamment au temps de calcul nécessaire par ces méthodes et qui augmente d'une manière exponentielle en fonction de la complexité et de la taille du problème. Selon Kaban et al. [17], pour un problème Job Shop à N tâches et M machines et un nombre fixe d'opérations N_{op}, il existe $\left(\left(M^{N_{op}} \times N_{op}!\right)^N \times (N!)^M\right)$ solutions possibles. Par exemple, si $M = N = N_{op} = 2$ le nombre de solutions est égal à 256 où l'utilisation de méthodes exactes est possible. Mais en augmentant légèrement le nombre de tâches et/ou machines, le nombre de solutions augmente considérablement. Pour $M = N = N_{op} = 3$ le nombre de solutions possibles est égal à 918 330 048, rendant l'utilisation des méthodes exactes infaisable pour des problèmes plus larges.

Cet état des faits a poussé les chercheurs à développer des méthodes (méta-heuristiques) moins performantes mais surtout moins couteuses en temps d'exécution et qui explorent l'espace de recherche d'une manière intelligente. C'est le cas notamment des algorithmes génétiques ou la recherche tabou. D'un autre côté, Chen & Matis [18] affirment que ces méta-heuristiques souffrent aussi de problèmes de passage à l'échelle. C'est-à-dire que le temps de résolution requis par ces méthodes augmente exponentiellement en fonction de la complexité du problème. D'autre part, ces méthodes, en raison de leur rigidité, ne sont pas idéales lors de la production quand un évènement inattendu survient et qu'il est nécessaire de réagir rapidement. Les auteurs concluent finalement qu'une alternative serait de miser sur les heuristiques à complexité linéaire, plus largement connues sous le nom de « *règles de priorité* ».

4.3. Problématique de la sélection des règles de priorité

Les règles de priorités (cf. Chapitre 3) présentent certains atouts comme la rapidité d'exécution et la flexibilité lors de la résolution de problèmes d'ordonnancement dynamique et/ou à forte complexité. En fonction du critère d'optimisation, on peut ne considérer qu'un certain nombre de règles connues pour leurs bonnes performances, néanmoins, il est nécessaire de tester toutes les règles afin d'identifier les meilleures. Pour se faire, la simulation est constamment sollicitée car elle permet de tester différents scénarios où les règles de priorité sont changées afin de trouver celle qui maximise/minimise la fonction objectif. Elle permet aussi d'observer le comportement du système suivant telle ou telle solution, ou lorsqu'une perturbation intervient. Cependant, tester toutes les règles de priorité peut avoir un effet catastrophique sur la responsivité du système de production. Ce dernier devra rester en standby pour une solution, affectant ainsi négativement les performances de l'atelier. Une des solutions possibles serait de

faire appel à un expert de la production avec la capacité et la connaissance nécessaires pour décider de la meilleure heuristique à utiliser pour résoudre le problème d'ordonnancement. L'expert prend cette décision en se basant soit sur sa longue expérience dans le domaine, soit en testant rapidement un nombre réduit d'heuristiques qu'il juge plus intéressantes en utilisant la simulation. L'expert peut être sollicité lorsque le problème à résoudre est à faible complexité, mais lorsque la complexité est très forte (NP-Difficile), l'expérience humaine n'est plus envisageable. La fouille de données peut alors être appelée à remplacer l'expert humain car elle profite à la fois des capacités de l'intelligence artificielle et de la puissance actuelle des ordinateurs et de leurs capacités de calcul.

La fouille de données est souvent considérée dans la littérature comme un atout (cf. Chapitre 3) pour l'automatisation de l'ordonnancement et la mise en place de systèmes d'aide à la décision au profit des responsables de production. Les techniques de fouille de données peuvent être utilisées pour réagir aux pannes machines [95] et minimiser leur impact, pour l'extraction de nouvelles règles de priorité avec un comportement similaire aux méthodes exactes ou méta-heuristiques [24], ou encore pour l'identification et la sélection de règles de priorité en temps-réel. Dans le cas de problèmes d'ordonnancement à forte complexité et où les règles de priorité se présentent comme alternative intéressante, la fouille de donnée permet d'apporter un plus pour mettre en place un système d'aide à la décision basé sur des expériences passées pour la résolution de nouveaux problèmes.

Dans cette perspective, nous proposons un système d'aide à la décision dont le but est d'identifier et d'allouer en temps-réel une règle de priorité pour chaque machine de l'atelier. Contrairement aux approches proposées dans la littérature ayant reposé sur les règles de priorité (cf. Section 2.4) où une règle de priorité est affectée à toutes les machines de l'atelier, le but de cette approche est d'utiliser une règle de priorité différente pour chacune des machines simultanément. Ceci, et en se fiant aux expérimentations, permet de trouver de meilleures solutions tout en réduisant considérablement le temps d'exécution total. Cette approche diffère des autres approches à base de fouille de données comme celle proposée par Metan et al. [22] où les règles de priorité sont choisies indépendamment l'une de l'autre, pouvant mener à des solutions peu performantes.

Afin de mettre en place un modèle de décision en utilisant les arbres de décision, d'importantes quantités de données issues du système de production sont requises. Ces données comportent notamment les descriptions des problèmes d'ordonnancement ainsi que leurs solutions. Afin d'alimenter la base de données avec de bonnes solutions et permettre d'obtenir un système d'aide à la décision efficace, la simulation est utilisée pour l'indentification, dans un premier temps, des meilleurs ensembles de règles de priorité. Comme expliqué précédemment, la simulation est un puissant outil pour tester et apprécier l'impact des différentes heuristiques, mais n'est pas très efficace lorsqu'il s'agit de prise de décision en temps-réel. Cette inefficacité est due en grande partie au temps nécessaire pour reproduire le système de production sur le simulateur (machines, tâches, et leurs caractéristiques), en plus du temps d'exécution requis pour la simulation et l'interprétation des résultats.

4.4. Approche proposée pour la sélection des règles de priorité

Dans cette approche [191], deux principaux objectifs sont mis en avant :

- La réduction du le temps d'exécution total des tâches en utilisant des règles de priorité qui diffèrent d'une machine à l'autre. Contrairement à ce qui se fait dans la littérature [17], [180], où l'on considère uniquement une seule règle, l'utilisation de plusieurs règles de priorité en même temps permet d'améliorer grandement les solutions. L'identification des meilleures règles de priorité se base entièrement sur la simulation. La simulation est un puissant outil largement utilisé dans l'ordonnancement [19], [25], [26], [91], [193]. Elle permet notamment de tester les règles de priorité et d'identifier celle qui assure les meilleures performances. Elle offre aussi la possibilité d'observer, d'analyser et de comprendre le comportement de l'atelier et de ses différentes composantes lorsque plusieurs séquences sont considérées. Dans cette approche, le rôle de la simulation consiste à trouver le meilleur ensemble de règles de priorité pour un certain nombre de problèmes, où chaque règle est allouée à une machine spécifique. Pour identifier le meilleur ensemble de règles en utilisant la simulation il est nécessaire d'explorer et de tester toutes les combinaisons possibles égal à q^M où M est le nombre de machines et q le nombre de règles de priorité considérées. Par exemple, pour un atelier composé de deux machines et où l'on considère cinq règles de priorité, le simulateur doit tester $2^5 = 32$ scénarios, avec un atelier de cinq machines et le même nombre de règles de priorité, le nombre de scénarios est de $5^5 = 3125$. Ce nombre augmente d'une manière exponentielle en fonction de nombre de machines et/ou de règles, menant ainsi rapidement vers une saturation des capacités du simulateur, et celles de l'unité de calcul par extension.
- L'automatisation, et l'accélération du processus d'identification et de sélection des règles de priorité : la difficulté majeure d'utilisation de la simulation pour résoudre des problèmes de grande taille nous a orientée vers la fouille de données. Ainsi, le système proposé doit être capable, en se basant sur des données historisées, de décider de la meilleure règle à utiliser d'une manière autonome. De plus, la solution doit être retournée instantanément et sans passer par des expérimentations ou des simulations supplémentaires qui seraient gourmandes en temps de calcul.

Afin de mettre en place une telle approche, il est nécessaire d'acquérir de grandes quantités de données concernant la résolution de problèmes Job Shop à base de règles de priorité. Une fois cette opération achevée, les techniques de fouille de données et d'intelligence artificielle peuvent être appliquées, aboutissant à un modèle de décision capable de résoudre de nouveaux problèmes d'ordonnancement par l'identification et l'allocation des meilleures règles de priorité en temps-réel.

L'approche proposée, décrite dans la Figure 4.1, peut être découpée en trois parties. Le principal but de la première partie est d'alimenter une base de données. Quant à la seconde, elle crée, évalue, et valide le modèle de décision en utilisant la fouille de données. Enfin, durant la dernière partie, les nouveaux problèmes sont résolus en allouant en temps-réel les règles de

priorité aux machines en utilisant le modèle de décision. Cette dernière partie offre aussi la possibilité d'améliorer les performances du système en recourant au retour d'information.

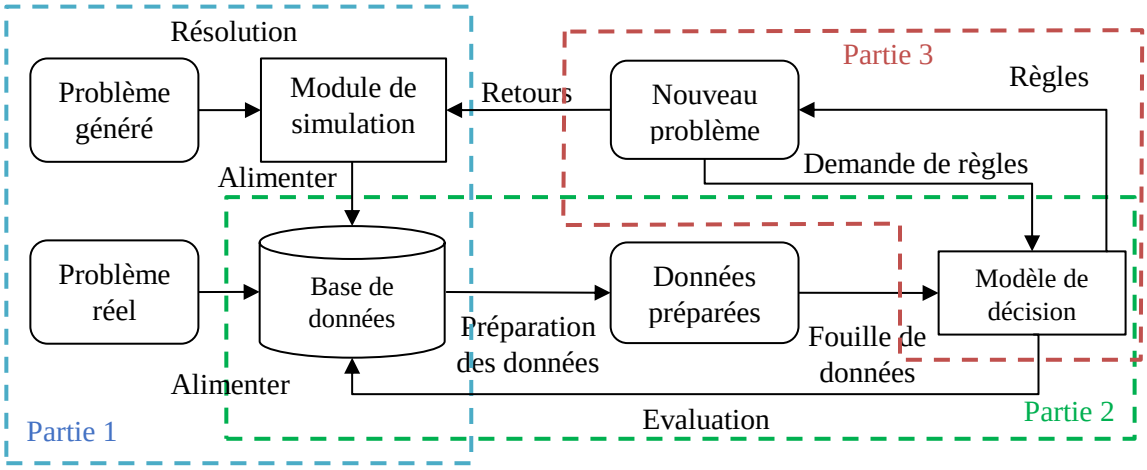

Figure 4.1. Architecture de l'approche proposée

4.4.1. Résolution des problèmes et collecte de données

Durant cette première partie, il nécessaire d'acquérir le maximum de données sous forme d'attributs/classe pour l'apprentissage. Cette procédure peut être accomplie dans un système de production réel [22], [82], [180], appelée « *problème réel* » dans la Figure 4.1. Toutes les données issues du processus de production concernant les tâches et machines peuvent être récupérées, stockées, prétraitées, analysées, et mises à profit afin d'améliorer l'ordonnancement et de résoudre de nouveaux problèmes.

Dans le cas de cette approche, et pour des raisons de non disponibilité de données réelles, un grand nombre de problèmes Job Shop générés est utilisé pour alimenter la base de données et ainsi prouver l'efficacité de l'approche. Durant les expérimentations, la base de données est alimentée par des informations issues de la résolution de problèmes Job Shop générés en se basant sur le travail de Taillard [129]. La simulation est ensuite utilisée pour tester toutes les combinaisons de règles de priorité et celle qui minimise le makespan est utilisée pour alimenter la base de données comme présenté dans la Figure 4.2.

Bien que l'exploration des différentes combinaisons prenne beaucoup de temps, elle reste toutefois faisable car effectuée en hors-ligne et pour un petit nombre de machines. Le système de production n'est donc pas affecté par cette opération.

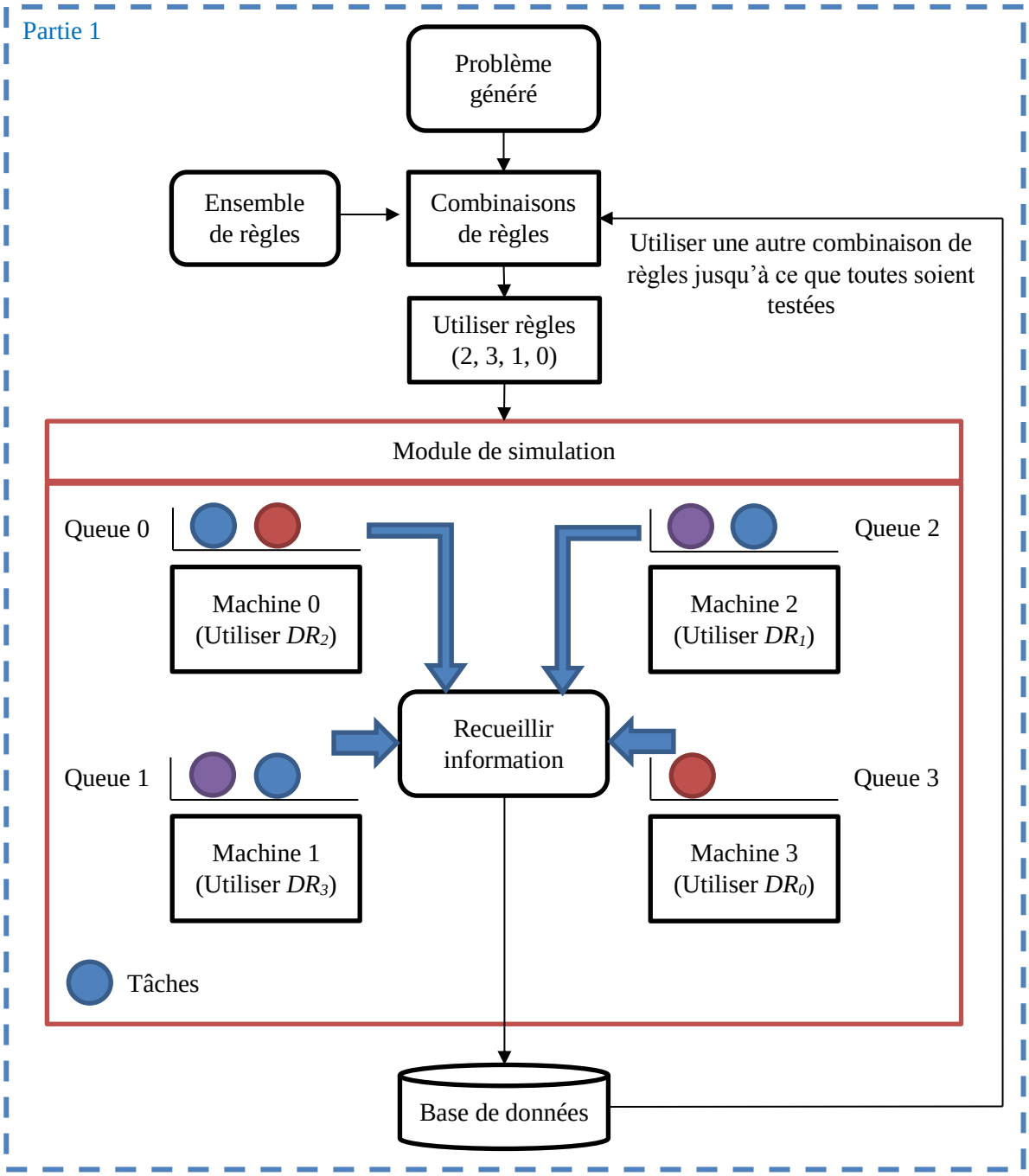

Figure 4.2. Résolution et collecte des données

4.4.2. Modélisation par fouille de données

En ce qui concerne cette partie, un modèle est créé en utilisant un algorithme d'arbres de décision. Le choix des arbres de décision est motivé par leur simplicité et leur large utilisation entre autres (cf. Section 3.4 et Section 3.5). L'arbre de décision utilise les données stockées dans la base de données issues d'expérimentations passées après un prétraitement. Ce prétraitement a comme objectif d'éliminer les données redondantes, incomplètes, ou erronées, le cas échéant. Il consiste aussi à trouver une représentation adéquate des données afin d'obtenir des connaissances utiles. Il est crucial de trouver une bonne représentation car elle est

directement liée à la qualité des résultats obtenus. Une fois le prétraitement terminé, l'arbre de décision peut être alors appliqué pour obtenir le modèle de décision.

Il est aussi utile de noter que lors de l'extraction des connaissances, il est possible de passer par une étape intéressante qui est l'évaluation afin d'améliorer le fonctionnement du modèle de décision. Pour se faire, les données sont divisées en deux groupes. Un pour l'apprentissage et la création du modèle, et le second pour l'évaluation et la validation de ce dernier. Si le modèle obtenu n'atteint pas les objectifs attendus, le processus de prétraitement est alors revu et corrigé de même que les paramètres de l'arbre de décision utilisés. Ce processus est répété autant de fois que nécessaire jusqu'à atteindre les objectifs désirés. La création et la validation du modèle de décision (voir Figure 4.3) se fait aussi hors-ligne permettant ainsi de faire un maximum d'itérations.

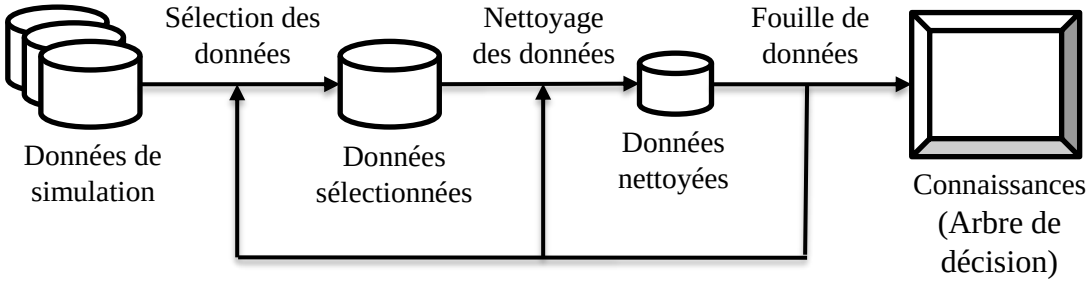

Figure 4.3. Processus d'ECD modifié

Dans ce qui suit nous présentons un exemple afin d'illustrer la seconde partie de l'approche proposée. Soit un problème Job Shop constitué de deux machines M_0 et M_1, et de trois tâches J_0, J_1, et J_2. Pour la résolution du problème on considère deux règles de priorité DR_0 et DR_1. Tout d'abord (comme expliqué dans la première partie) la simulation teste les différentes combinaisons possibles données dans le Tableau 4.1 ci-dessous :

Tableau 4.1. Combinaisons des règles de priorité

Machine	Combination 1	Combination 2	Combination 3	Combination 4
M_0	DR_0	DR_0	DR_1	DR_1
M_1	DR_0	DR_1	DR_0	DR_1

Chacune des trois tâches possède sa propre séquence et un temps d'exécution pour chaque opération et une date d'arrivée égale à zéro comme décrit dans le Tableau 4.2. Par exemple, la tâche J_0 doit être opérée d'abord sur la machine M_0 pour une durée de 15 unités de temps, puis sur la machine M_1 durant 20 unités de temps.

Tableau 4.2. Attributs des tâches

Jobs	Opération 1	Opération 2
J_0	M_0 (15)	M_1 (20)
J_1	M_1 (8)	-
J_2	M_1 (10)	M_0 (5)

Durant les simulations des différentes combinaisons de règles de priorité plusieurs attributs sont sauvegardés (voir Tableau 4.3) dans la base de données à savoir le temps d'attente de la tâche sur la machine WT (*Wait Time*), le temps d'exécution de l'opération p_{ij} (*Processing Time PT*), et le temps d'exécution restant pour terminer la tâche TWT (*Total Work Remaining*). Ces données sont récupérées et sauvegardées à chaque fois qu'une tâche rentre ou quitte la file d'attente de la machine. Une fois toutes les combinaisons testées, un prétraitement est fait pour garder uniquement les données issues des meilleures combinaisons (minimisant le C_{max}) et supprimer le reste (doublons compris). Dans cet exemple illustratif, on suppose que les meilleures combinaisons sont 3 (couple DR_1, DR_0) et 4 (couple DR_1, DR_1) obtiennent les meilleurs résultats. On obtient alors le tableau suivant :

Tableau 4.3. Données collectées

WT	PT	TWR	DR
0	15	35	DR_1
0	5	5	DR_1
0	8	8	DR_1
8	10	15	DR_1
0	15	35	DR_1
0	5	5	DR_1
3	20	20	DR_0
10	8	8	DR_0
0	10	15	DR_0

Par la suite un algorithme d'arbres de décision (C4.5 en l'occurrence) est utilisé pour obtenir les résultats suivants (voir aussi Figure 4.4) :

Si PT < 17.5 **et** WT < 9 **et** TWR < 11.5 **alors** DR_1
Si PT < 17.5 **et** WT < 9 **et** TWR >= 11.5 **et** PT < 12.5 **et** WT < 4 **alors** DR_0
Si PT < 17.5 **et** WT < 9 **et** TWR >= 11.5 **et** PT < 12.5 **et** WT >= 4 **alors** DR_{0_1}
Si PT < 17.5 **et** WT < 9 **et** TWR >= 11.5 **et** PT >= 12.5 **alors** DR_1
Si PT < 17.5 **et** WT >= 9 **alors** DR_0
Si PT >= 17.5 **alors** DR_0

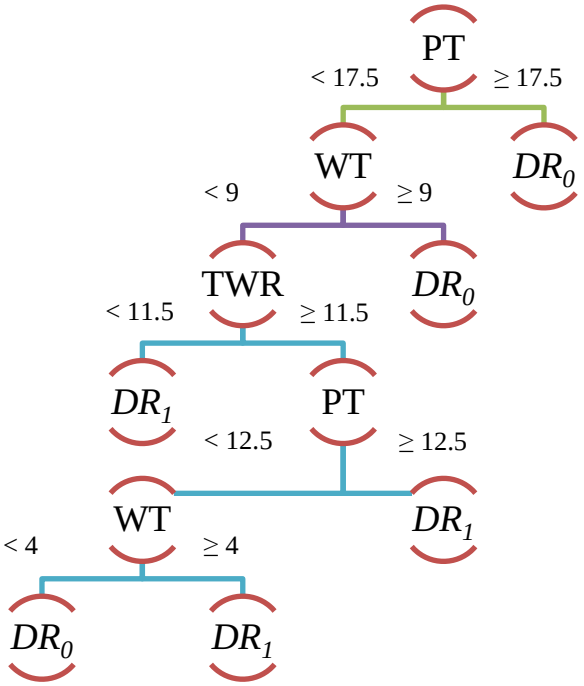

Figure 4.4. Arbre de décision construit

L'arbre de décision (modèle) sera par la suite utilisé lors de la résolution de nouveaux problèmes afin de décider de la meilleure règle de priorité à allouer à chaque machine de l'atelier en fonction des tâches en attente sur leurs files. Ce processus se fait quasi-instantanément et ne requiert aucune simulation supplémentaire.

En testant les différentes combinaisons de règles de priorité, il est très probable de retrouver plusieurs qui optimisent la fonction objectif (même valeur du makespan). Ceci a par conséquent orienté cette approche vers la classification « *Multi-labels* » car plusieurs classes peuvent être associées à la fois à chaque tâche. Les techniques de classification multi-labels sont très sollicitées pour la catégorisation de la musique, la classification des fonctions des protéines, mais aussi dans la classifications des textes [194]. Les méthodes de classification multi-labels sont regroupées en deux familles, « *algorithmes de transformation* » et « *méthodes d'adaptation* ». La première famille rassemble les techniques qui transforment les données en « *single-label* », ou chaque enregistrement associé à plusieurs classes est dupliqué avec une des classes (une classe par enregistrement) (voir Figure 4.5). Par la suite, au lieu de créer un seul arbre de décision, on crée un arbre pour chaque classe. Quant à la seconde famille, elle regroupe des méthodes de fouille de données modifiée qui prennent en considération l'aspect multi-labels des données. Plus de détails sur la classification multi-labels peuvent être trouvés dans les publications de Tsoumakas & Katakis [195], Carvalho & Freitas [196], Modi & Panchal [197], ou Gjorgjevikj et al. [198].

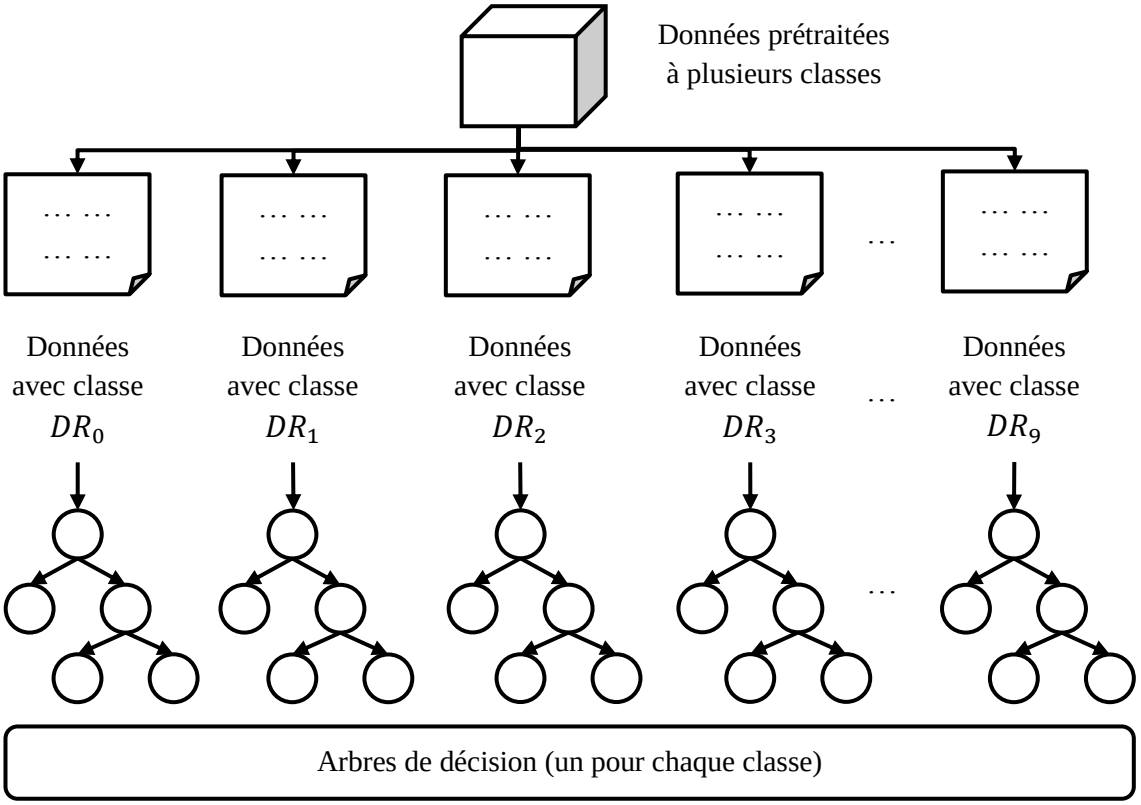

Figure 4.5. Classification multi-labels en utilisant la transformation de données

4.4.3. Sélection des règles de priorité en temps-réel

Enfin, dans cette dernière partie de l'approche proposée (voir Figure 4.6), lorsqu'il est nécessaire de résoudre un nouveau problème Job Shop au lieu de se focaliser sur la simulation et explorer toutes les combinaisons possibles, on fait appel à l'arbre décision pour l'allocation des règles de priorité aux machines. Le modèle de décision est capable d'accomplir cette tâche en temps-réel (cette phase est conduite en ligne). En outre, si le nouveau problème n'est pas présent dans la base de données alors ce dernier est simulé et utilisé pour alimenter la base (revoir Figure 4.2). Le système générera par la suite un nouvel arbre de décision en prenant en considération les nouvelles données et améliorer ainsi son fonctionnement. Cette dernière phase de retour d'information est effectuée en hors-ligne.

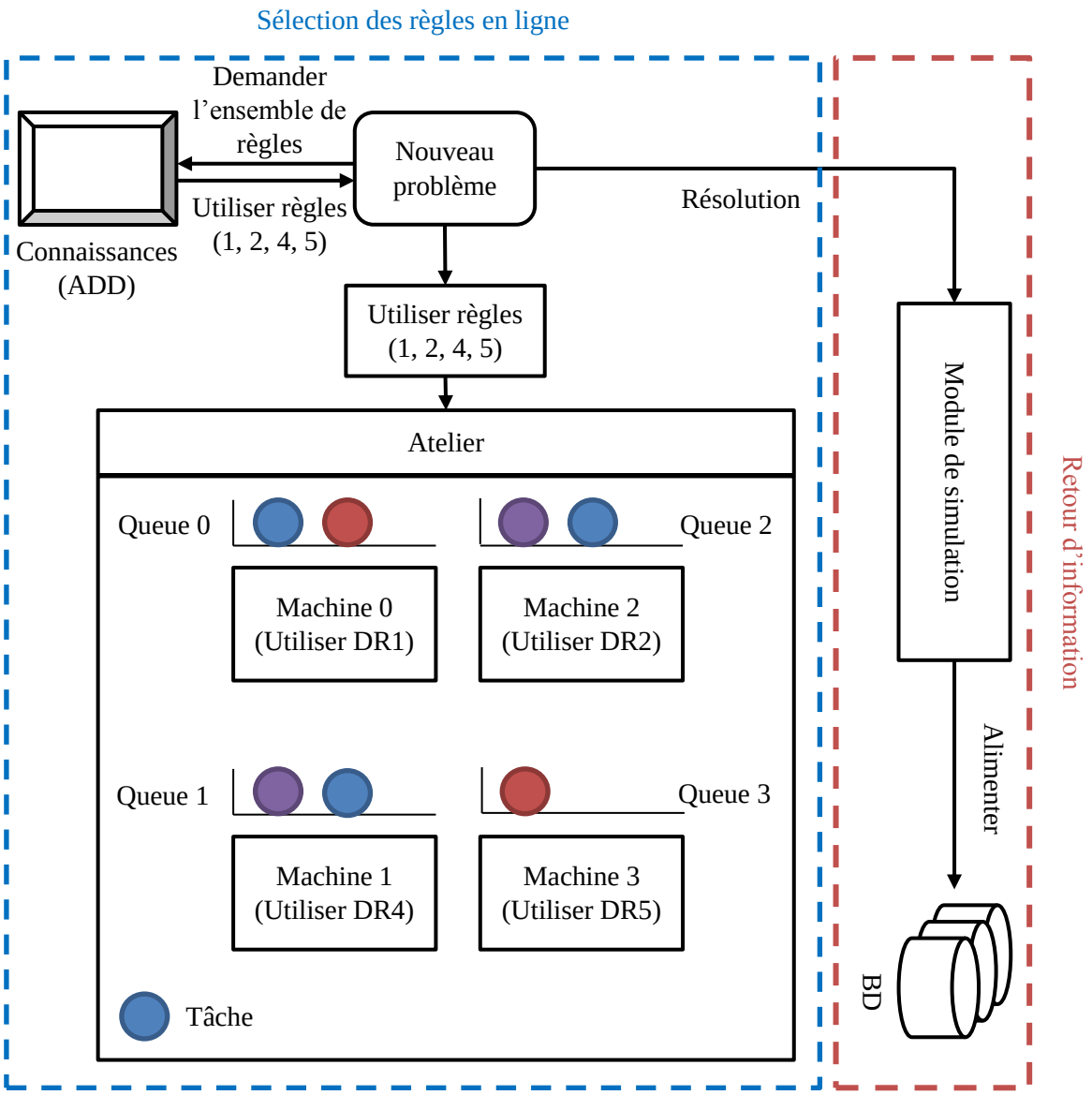

Figure 4.6. Sélection des règles de priorité en temps réel et retour d'information

4.5. Expérimentations

Le but de cette approche est d'identifier en temps-réel une règle de priorité pour chaque machine de l'atelier en utilisant la fouille de données et sans passer par la simulation. Pour se faire, deux groupes de règles de priorité « *statiques* » et « *dynamiques* » sont considérés. Ces règles sont introduites par Kaban et al. [17]. On parle de règles statiques si la priorité de chaque tâche (Z_i assigné à la tâche) ne change pas avec le temps. D'un autre côté, une règle est dynamique si les priorités des tâches évoluent dans le temps. Les règles de priorité proposées sont basées sur les paramètres (attributs) suivants :

- Temps d'exécution (Processing Time PT) : le temps d'exécution requis pour terminer une opération ij,
- Nombre d'opérations restante (Process Sequence PS) : le nombre d'opération restante pour achever la tâche i,

- Somme des temps d'exécution des opération (Total Processing Time TPT) : donné par $p_i = \sum_{j=0}^{N_i-1} p_{ij}$,
- Temps d'attente (Wait Time WT) : le temps d'attente d'une tâche sur une machine (valeur fixée à zéro lorsque la tâche rentre dans une nouvelle queue),
- Temps d'exécution restant (Total Work Remaining TWR) : la somme des temps d'exécution des opérations restantes, opération en cours d'exécution comprise.

Les nouvelles règles de priorité sont obtenues en utilisant l'Équation 4.2 ci-dessous. On obtient alors dix règles de priorité.

$$Z = Attribut1_{ij} \times Attribut2_{ij} \tag{4.2}$$

$Attribut1_{ij}$ et $Attribut2_{ij}$ sont l'un des cinq paramètres et ne peuvent prendre la même valeur à la fois où i représente la tâche et j la machine. Par exemple, la règle obtenue par l'Équation 4.3 consiste à multiplier le temps d'exécution PT_{ij} de l'opération ij par le nombre d'opérations restantes PS_i pour achever cette tâche i. La tâche qui obtient la plus petite valeur de Z est exécutée en premier.

$$Z = PT_{ij} \times PS_i \tag{4.3}$$

L'ensemble des dix règles possibles est donné dans le Tableau 4.4 ci-dessous :

Tableau 4.4. Nouvelles règles de priorité

Attribut	**PT**	**PS**	**TPT**	**WT**	**TWR**
PT	✗	✓	✓	✓	✓
PS	✗	✗	✓	✓	✓
TPT	✗	✗	✗	✓	✓
WT	✗	✗	✗	✗	✓
TWR	✗	✗	✗	✗	✗

Les nouvelles règles de priorité sont données comme suit :

- $PT \times PS$,
- $PT \times TPT$,
- $PT \times WT$,
- $PT \times TWR$,
- $PS \times TPT$,
- $PS \times WT$,
- $PS \times TWR$,
- $TPT \times WT$,
- $TPT \times TWR$,
- $WT \times TWR$.

Durant leurs expérimentations, Kaban et al. [17] démontrent l'efficacité des nouvelles règles de priorité produites en combinant les différents attributs. La comparaison est faite avec des règles répandues dans la littérature, comme FIFO, LIFO, SPT, ou LPT. Dans leurs perspectives, les auteurs suggèrent de considérer différentes règles de priorité, différentes pour chaque machine de l'atelier. Ceci pouvant se faire en testant toutes les combinaisons en utilisant la simulation.

Afin de prouver l'efficacité de l'approche proposée la librairie OR (OR-Library) est considérée [141] qui comporte un certain nombre de problèmes Job Shop. Cependant, le nombre de problèmes dans cette librairie est insuffisant pour les besoins de l'expérimentation et pour produire un modèle de décision efficace. Par conséquence, nous considérons le travail de Taillard [129] sur la génération de problèmes Job Shop à quatre machines. Ce choix du nombre de machines est motivé par l'utilisation de ces problèmes par Mebarki & Shahzad [199], Pierreval & Mebarki [144] ainsi que d'autres auteur cités dans ces deux travaux de recherche. Durant la génération des problèmes Job Shop les hypothèses suivantes sont prises en considération :

- Les dates d'arrivées de toutes les tâches sont égales à 0,
- Les dates d'échéance ne sont pas considérées,
- Les machines ne peuvent exécuter qu'une seule opération à la fois, et l'exécution d'une tâche ne peut être suspendue,
- Les perturbations et les coûts de transport ne sont pas considérés.

L'approche proposée par Taillard [129] permet de générer des problèmes Job Shop de différentes tailles (nombres de tâches et de machines). Dans ces expérimentations, les problèmes Job Shop générés sont composés de quarante tâche $N = 40$ et de quatre machines $M = 4$. Les séquences des tâches sont déterminées aléatoirement avec des temps d'exécution variant entre 0 et 99 suivant une distribution uniforme discrète. Une tâche ne peut visiter une machine qu'une seule fois. Si une tâche ne requiert pas d'être opérée sur une machine, son temps d'exécution est alors fixé à zéro. Dans le Tableau 4.5 est donné un exemple d'un problème 4×4 généré.

Tableau 4.5. Exemple d'un problème Job Shop de taille 4×4

Tâche/Machine	$\boldsymbol{M_0}$	$\boldsymbol{pt_{00}}$	$\boldsymbol{M_1}$	$\boldsymbol{pt_{01}}$	$\boldsymbol{M_2}$	$\boldsymbol{pt_{02}}$	$\boldsymbol{M_3}$	$\boldsymbol{pt_{03}}$
J_0	**3**	41	**2**	5	**1**	32	**0**	87
J_1	**1**	66	**0**	36	**2**	4	**3**	11
J_2	**0**	80	**0**	88	**3**	50	**2**	0
J_3	**1**	55	**2**	12	**0**	0	**3**	33

Durant les expérimentations, 50 problèmes Job Shop de taille 40×4 sont générés. Chaque problème est ensuite résolu en utilisation la simulation pour tester toutes les combinaisons de règles de priorité (voir Équation 4.4) c'est-à-dire, simulé 10.000 fois.

$$q^M = 10^4 = 10.000 \tag{4.4}$$

Une étude comparative est menée avec l'approche de Kaban et al. [17] qui consiste à utiliser une règle pour toutes les machines (appelée ici « *One-DR* »).

4.6. Résultats et discussion

Les résultats des expérimentations montrent que sur les 50 problèmes Job Shop considérés, l'utilisation de plusieurs règles de priorité simultanément permet de minimiser le temps d'exécution total dans 38% des cas. Pour les autres 62% des cas, l'utilisation de règles de priorité multiples atteint le même makespan. Pour plus de détails, une comparaison est menée avec la meilleure et la pire « *One-DR* », c'est-à-dire avec la règle qui maximise le makespan (pire règle) et celle qui le minimise (meilleure règle) pour toutes les machines. En utilisant différentes règles, le temps d'exécution total est inférieur à la meilleure One-DR et oscillant entre 1 à 8%. Alors que comparée avec la pire One-DR l'amélioration varie de 2 à 44%. Dans le Tableau 4.6 on montre quelques résultats de problèmes Job Shop sélectionnés au hasard. Dans la Figure 4.7 l'écart (voir Équation 4.5) entre les résultats de l'approche proposée (AP) et la meilleure One-DR est donné pour chacun des 50 problèmes. L'écart entre l'approche proposée et la pire One-DR est montré dans la Figure 4.8.

$$\text{OneDR}\,(i) - \text{AP}(i) \tag{4.5}$$

Tableau 4.6. Quelques résultats

JSSP N°	Approche de Kaban et al. [17]		Plusieurs règles
	Meilleure	Pire	
0	**2809**	3232	**2809**
3	2604	2977	**2580**
10	2487	3172	**2454**
23	2166	2683	**2160**
42	**2280**	2845	**2280**

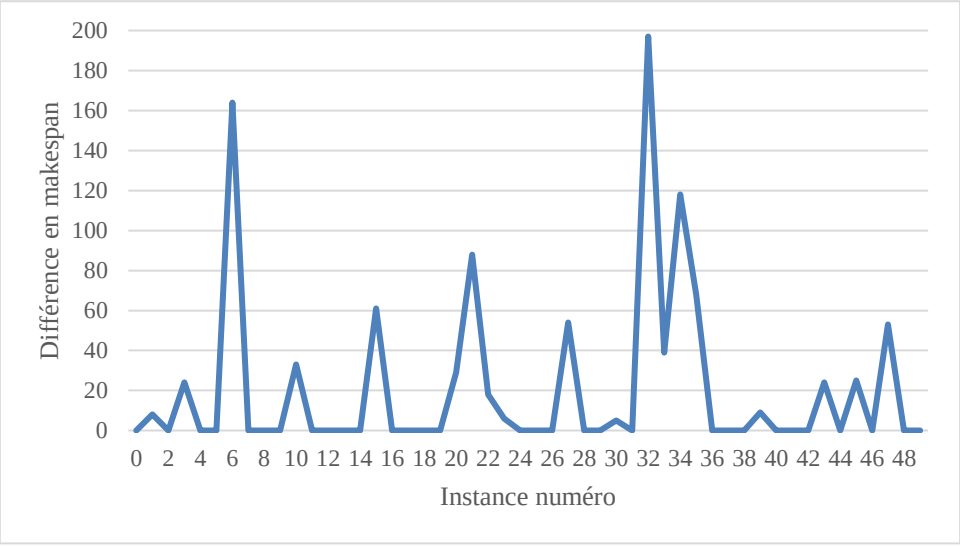

Figure 4.7. Approche proposée comparée avec la meilleure One-DR

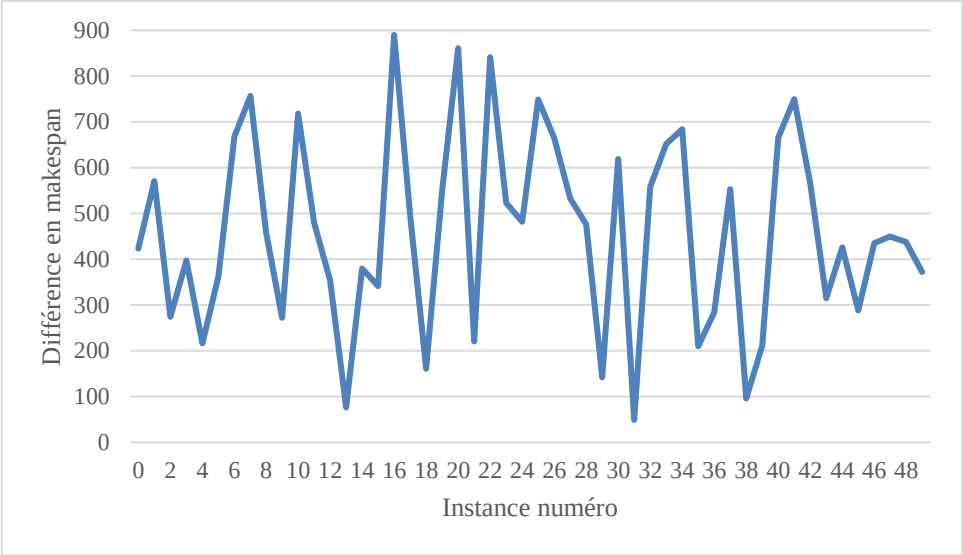

Figure 4.8. Approche proposée comparée avec la pire One-DR

Les 50 problèmes Job Shop sont aussi résolus avec des méthodes exactes en utilisant IBM CPlex. CPlex obtient de meilleurs résultats que l'utilisation de plusieurs règles de priorité avec une moyenne de 1.45% pour 46% des cas. En ce que concerne les 54% problèmes restants, l'approche proposée obtient le même makespan que CPlex prouvant ainsi l'efficacité de l'utilisation de plusieurs règles à la fois. Dans la Figure 4.9 nous présentons l'écart, donné par l'Équation 4.6, entre l'approche proposée et CPlex.

$$AP(i) - \text{CPlex}(i) \tag{4.6}$$

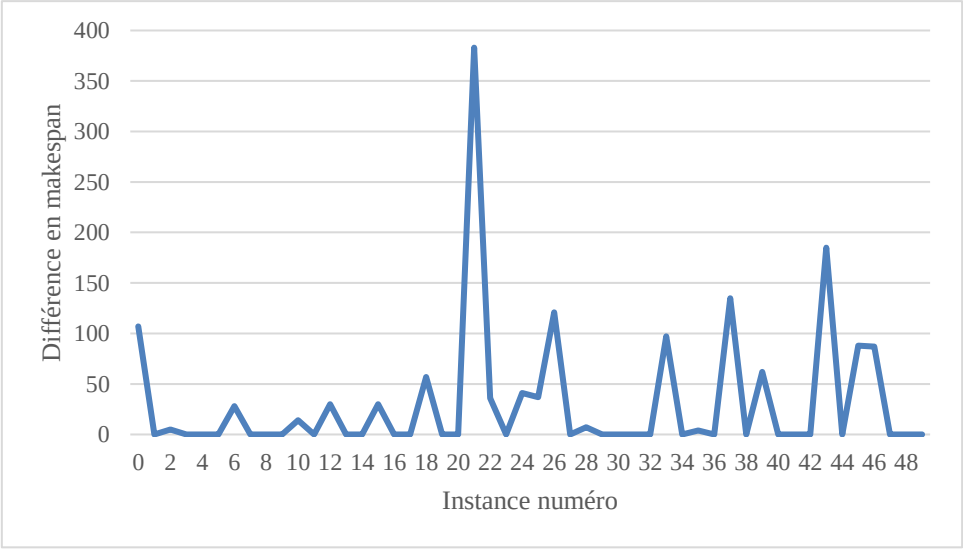

Figure 4.9. Approche proposée comparée avec CPlex

En se basant sur ces données, on peut conclure que l'utilisation des plusieurs règles de priorité à la fois permet d'améliorer considérablement les solutions et par extension le temps d'exécution total. Cependant, le problème d'une telle démarche est directement lié au nombre de combinaisons à tester, 10.000 fois pour les problèmes Job Shop 40×4. Si le nombre de machines ou le nombre de règles de priorité augmentait, le nombre de combinaisons

augmenterait à son tour d'une manière exponentielle (revoir Équation 4.4). Ainsi l'utilisation de la simulation uniquement pour l'exploration des différentes solutions deviendrait impossible et ne peut être envisagée pour l'ordonnancement en temps-réel. Comme discuté dans l'approche proposée (cf. Section 4.4), pour éviter de tester toutes les combinaisons machines/règles il serait intéressant de considérer la fouille de données et de mettre à profit les expériences passées. Le modèle de décision serait donc capable d'identifier et d'allouer en temps réel une règle de priorité à chaque machine de l'atelier en fonction des tâches en attente sur sa file.

Comme décrit dans la Figure 4.2, durant les simulations des différentes solutions, chaque fois qu'une tâche quitte la file d'attente d'une machine ses attributs et ceux des tâches encore sur la file (voir Tableau 4.7) sont enregistrés dans la base de données en plus de la règle de priorité utilisée actuellement par cette machine. Ces attributs sont : PT, PS, TPT, WT, TWR, en plus de l'identifiant de la tâche, et du numéro de machine $\{0, 1, 2, 3\}$. Quant à la règle de priorité elle prend l'une des valeurs suivantes : PT × PS, PT × TPT, PT × WT, PT × TWR, PS × TPT, PS × WT, PS × TWR, TPT × WT, TPT × TWR or WT × TWR.

Tableau 4.7. Attributs sélectionnés

Attribut	**Signification**
ID tâche	Le numéro d'identification de la tâche (facultatif)
PT	Le temps d'exécution de l'opération p_{ij}
PS	Le nombre d'opérations restantes
TPT	La somme des temps d'exécution des opérations
WT	Temps d'attente de la tâche sur la machine
TWR	Temps d'exécution encore nécessaire pour achever la tâche
ID machine	Numéro d'identification de la machine (facultatif)
Règle de priorité	Règle utilisée pour ordonnancer les tâches

Une fois les données récupérées et enregistrées, deux principales opérations de prétraitement sont menées. La première (sélection des données) a pour objectif de garder uniquement les données issues de combinaison de règles qui minimisent au maximum le makespan (bonne réponses) et d'éliminer le reste (mauvaises réponses). L'idée est de créer un modèle capable de choisir uniquement les meilleures règles. La seconde opération élimine les données redondantes, mais aussi les informations inutiles comme les identifiants des tâches et des machines. Les données obtenues sont ensuite divisées en deux groupes, l'un pour l'apprentissage et est constitué de 70% des données, et l'autre pour la validation et l'évaluation regroupe les 30% restants.

Par la suite, l'outil Meka [200] est utilisé pour la classification multi-labels. Plusieurs techniques de transformation de données implémentées dans cet outil sont testées, et les résultats montrent que la méthode RT « *Ranking + Threshold* » combinée avec l'algorithme d'arbres de décision C4.5 maximisent le taux de bonne classification (précision) (voir les résultats dans le Tableau 4.8).

Tableau 4.8. Résultats de la classification multi-labels par classe

Règle	$PT \times PS$	$PT \times TPT$	$PT \times WT$	$PT \times TWR$	$PS \times TPT$	$PS \times WT$	$PS \times TWR$	$TPT \times WT$	$TPT \times TWR$	$WT \times TWR$
Préc. (%)	**74**	**84**	**87**	**80**	**72**	**66**	**61**	**75**	**75**	**78**

Le modèle de décision obtenu possède un taux de classification global de 75.07% pour sélectionner et allouer au moins une bonne règle de priorité. Contrairement à la simulation qui requière un temps d'exécution assez conséquent, le modèle de décision est capable de renvoyer la meilleure règle de priorité instantanément (quelques millisecondes). Chaque tâche en attente sur une file est classée (en se basant sur ses attributs PT, PS, ...) en utilisant les dix arbres de décision résultant en une ou plusieurs classes associées à cette tâche. Afin de décider la règle à appliquer à la machine il suffit simplement de trouver la règle la plus populaire parmi les tâches (plus grand nombre de tâches associées à une classe donnée).

En outre, le modèle obtenu peut s'améliorer continuellement (revoir Figure 4.6) en utilisant un retour d'informations. Lorsqu'on doit résoudre un nouveau problème Job Shop, le modèle de décision va retourner une solution en temps-réel. Une fois la production terminée, ce problème est repris, et toutes les combinaisons sont testées via simulation, tout en alimentant la base de données. En utilisant cette nouvelle instance, un nouveau modèle de décision est généré qui devrait retourner de meilleurs résultats par rapport à celui qui l'a précédé.

4.7. Conclusion et perspectives

Le but de cette approche est de proposer un système capable d'identifier et d'allouer en temps-réel la meilleure règle de priorité pour chaque machine de l'atelier Job Shop. Ce faisant, le temps d'exécution total s'en retrouve réduit comparé à l'utilisation d'une seule règle pour toutes les machines. Les résultats de cette approche se rapprochent même de ceux des méthodes exactes trouvés en utilisant CPlex pour des problèmes de petite taille. Cependant, et contrairement aux méthodes exactes, l'approche proposées est capable de résoudre des problèmes à forte complexité ou de grande taille et en temps-réel.

Par ailleurs, l'utilisation d'une telle approche permet de changer dynamiquement les règles de priorité qui peut se révéler efficace en cas de perturbations comme l'arrivée de nouvelles tâches. En outre, le modèle de décision sélectionne les règles de priorité en temps-réel et sans recourir à la simulation. Il faut ajouter à cela la capacité de système de s'améliorer continuellement via retours d'informations à chaque fois qu'un nouveau problème d'ordonnancement est rencontré. Les avantages de cette approche sont en grande partie dus à la fouille de données et sa capacité de créer des modèles pour aider le responsable de l'atelier de production à prendre des décisions en temps-réel.

En ce qui concerne les perspectives, il serait intéressant d'explorer d'autres règles de priorité afin de trouver de meilleures solutions. Les expérimentations peuvent aussi être étendues afin de couvrir des problèmes encore plus complexes que ceux traités dans les expérimentations où les méthodes exactes ou les méta-heuristiques ne sont pas effectives. Finalement, un dernier

aspect et non des moindres, serait de mener plus d'expérimentations sur la partie fouille de données ; considérer des techniques autres que les arbres de décision, ou s'intéresser aux adaptations d'algorithmes pour la classification multi-labels.

CHAPITRE

5. Algorithmes Génétiques, Fouille de Données, et Simulation pour l'Allocation de Règles de Priorité en Temps-Réel

Résumé : Dans la production, les expériences réalisées ultérieurement ainsi que l'expérience acquise par les opérateurs humains et les responsables de production peuvent être mis à profit afin d'optimiser la planification et l'ordonnancement. Pour se faire, la fouille de données peut être sollicitée afin de transformer ces données et expériences en connaissances utiles pour l'amélioration de la production et la prise de décision en temps réel. Le but de cette approche est de combiner règles de priorité, algorithmes génétiques, fouille de données, et simulation pour allouer des règles aux machines en fonction des tâches en attente sur leurs files. Les expérimentations sont effectuées sur des problèmes Job Shop avec le temps d'exécution total comme critère d'optimisation. Les résultats démontrent que l'approche proposée est une alternative viable et efficace pour la résolution du Job Shop.

Mots-clés : Règles de Priorité · Fouille de Données · Arbres de Décision · Algorithmes Génétiques · Simulation · Job Shop · Ordonnancement en Temps-Réel · Makespan

5.1. Introduction

Pour les entreprises, en dépit de leur taille ou leur domaine d'activité, la connaissance est un atout majeur pour leur développement et leur succès. Cette connaissance se retrouve dans tous les domaines d'activité de l'entreprise, le marketing, les achats, le design, la production, la maintenance des équipements, la logistique, etc. Néanmoins, cette connaissance reste difficile à identifier, à unifier et à mettre à profit en raison de la complexité et du volume des données à traiter. Mais une fois acquise elle peut notamment servir à mieux cerner les besoins des clients et leurs habitudes, déterminer leurs besoins réels, identifier le meilleur fournisseur pour l'achat de matières premières, ou pour l'accomplissement d'opérations liées à la production. En mettant en place et en exploitant cette connaissance, l'entreprise peut améliorer son processus de production et réduire ses prix de revient, assurant ainsi sa survie et son succès dans un environnement ultra compétitif [201].

Cette philosophie d'utilisation des connaissances s'applique aussi aux problèmes d'ordonnancement dont le Job Shop. Pour résoudre ce type de problème, il existe dans la littérature un grand nombre de méthodes regroupées en deux familles communément appelées méthodes exactes et méthodes heuristiques/méta-heuristiques. Lorsque les problèmes à résoudre sont de petite taille avec un environnement statique où les attributs des tâches sont connus au préalable, il est intéressant de considérer les méthodes exactes, comme le Branch & Bound, pour leur résolution [12]. Cependant, lorsqu'il s'agit de problèmes de grande taille, à forte complexité, ou lorsqu'il s'agit d'ordonnancement dynamique les méthodes exactes deviennent inopérantes en raison des temps de calcul requis [44]. De ce fait, les heuristiques et les méta-heuristiques sont considérées comme de bonnes alternatives en raison de leurs simplicités, leurs faibles temps de calcul, ainsi que leurs résultats relativement bons.

Parmi les méthodes les plus utilisées pour l'ordonnancement, on trouve les règles de priorité (cf. Chapitre 2). Elles sont très sollicitées dans la littérature [23] en raison de leur efficacité, simplicité, et capacité de définir très rapidement une priorité pour chaque tâche sur la file d'attente d'une machine. Cependant, dans la pratique on énumère un grand nombre de règles de priorité (cf. Section 2.5) et aucune ne peut surpasser les autres pour résoudre différents problèmes. Afin de déterminer celle qui optimise au mieux la fonction objectif, on fait généralement appel à la simulation pour tester les différentes règles de priorité. Le plus souvent on associe une seule règle à toutes les machines de l'atelier pour ordonnancer les tâches. Mais comme démontré dans l'approche présentée dans le Chapitre 4, l'utilisation dans le même temps de règles différentes pour les machines permet d'améliorer les solutions obtenues [201]. Par conséquent, la sélection des règles de priorité à utiliser devient plus complexe et requiert un temps d'exécution plus important, car le nombre de solutions possibles passe de q à q^M, où q est le nombre de règles de priorité et M le nombre de machines. En outre, Metan et al. [22] affirment que le changement de règles tout au long du processus d'ordonnancement, connu en anglais sous le terme « *multi-pass scheduling* », permet aussi d'obtenir de meilleures solutions mais en contrepartie le nombre de solutions possibles et la complexité du problème augmentent.

Le but de cette approche est d'employer la fouille de données afin d'identifier la meilleure règle de priorité à allouer à chaque machine en se basant les caractéristiques des tâches en attente de

cette ressource. L'utilisation du data mining, comme c'est le cas pour tout autre système d'aide à la décision, devrait accélérer, automatiser, simplifier la sélection des règles de priorité, et l'accomplir en temps-réel. La fouille de données génère un modèle de décision en se basant sur un ensemble de données et d'expériences passées, qui après un prétraitement les utilise pour la phase d'apprentissage. Afin de mettre en place une telle source de données, un algorithme génétique est proposé. L'AG résout les problèmes Job Shop en créant et en faisant évoluer des ensembles de règles de priorité, puis en les assignant aux ressources. L'algorithme génétique est combiné avec un module de simulation qui reproduit les problèmes Job Shop, collecte les données, et alimente une base de données avec des informations sur le comportement des tâches lors de la simulation. En plus, le simulateur teste les combinaisons de règles de priorité et renvoie le temps d'exécution total à l'AG (évaluation des solutions). Les données obtenues stockées dans la base de données sont par la suite prétraitées et préparées en vue de la création d'un arbre de décision qui suggèrera la meilleure règle de priorité à utiliser à chaque fois qu'une machine est disponible avec des tâches sur sa file d'attente.

L'avantage de cette approche réside dans le fait qu'elle ne requiert pas de simulation lors de la résolution de nouveaux problèmes Job Shop et, ce faisant, permet d'ordonnancer les tâches en temps-réel tout en produisant des solutions réalisables et de bonne qualité. Cette approche permet aussi de gérer les évènements imprévus en changeant d'une manière continue les règles de priorité et de répondre ainsi au mieux à cette perturbation. En utilisant la fouille de données, il devient possible de résoudre des problèmes d'ordonnancement à très forte complexité (NP-Difficiles) très rapidement, contrairement aux méthodes exactes, méta-heuristiques, ou approches basées uniquement sur la simulation, où le temps requis augmente fortement.

5.2. Les algorithmes génétiques pour le Job Shop : état de l'art

L'algorithme génétique développé par Holland et son équipe [117] en 1975 est une méthode évolutive (cf. Section 1.7.2.3) capable de simuler la génétique et la sélection naturelle. L'AG est composé d'une population initiale de chromosomes, connus aussi sous le terme de « *solutions encodées* ». Chaque chromosome est une combinaison de plusieurs gènes. Dans le cas des problèmes d'ordonnancement, un chromosome est une séquence (ordre de passage des tâches ou opérations), et un gène représente une opération [35], [190]. Chaque chromosome est associé avec une « *fitness* » ou une valeur de la fonction objectif, comme le retard, ou le temps d'exécution total. En utilisant cette valeur et se basant sur une méthode de sélection, les meilleurs chromosomes sont identifiés et choisis pour se reproduire et créer de nouveaux chromosomes. Après plusieurs générations ou après atteindre la valeur de la fonction objectif désirée, l'algorithme génétique est stoppé, le meilleur chromosome identifié, interprété, adapté, et utilisé dans le système de production.

En détails, l'algorithme génétique comprend plusieurs opérations distinctes à savoir, la création de la population initiale, l'évaluation des chromosomes, la sélection, le croisement, et la mutation. Lors de la génération de la population initiale, il est possible de générer les solutions aléatoirement [86], mais dans le cas de problèmes d'ordonnancement à plusieurs machines (comme le Job Shop ou le Flow Shop), il est nécessaire de réaliser une tâche complémentaire qui a pour but de vérifier et de corriger les solutions afin de respecter toutes les contraintes.

Cette opération peut s'avérer lente et couteuse en temps de calcul, en raison de la nécessité de vérifier chaque chromosome créé lors de génération de la population initiale, le croisement, ou la mutation. Afin de minimiser l'impact de cette opération et afin d'améliorer la qualité des solutions de la population initiale, d'autres chercheurs se sont intéressés à l'intégration d'heuristiques ou de méta-heuristiques avec l'AG. Cette combinaison permet ainsi à l'AG d'éviter les optimums locaux et d'accélérer la convergence vers une solution optimale. Dans cette optique, Liu et al. [46] intègrent des règles de priorité pour la génération de la population initiale pour un problème Single Machine. Selon les auteurs, cette démarche permet d'améliorer nettement les résultats de l'algorithme génétique.

Par la suite, tous les chromosomes, créés lors de la génération de la population initiale ou bien lors du croisement ou de la mutation, sont évalués et leur survie déterminée dans la génération suivante. L'évaluation attribue une valeur à chaque solution en fonction du critère d'optimisation du problème d'ordonnancement, le makespan par exemple. L'évaluation est suivie par la sélection qui garantit la survie des meilleurs chromosomes au dépend des pires solutions de la population. Plusieurs paramètres sont pris en considération durant cette étape, comme le nombre de chromosomes à sélectionner, la probabilité de sélection, le processus de classement des chromosomes, ou le ratio croisement/mutation. Par exemple, Hasan et al. [202] considèrent une sélection élitiste, avec un chromosome parent choisi parmi les 15% meilleurs chromosomes et le second est identifié en utilisant un tournoi (classement) entre deux chromosomes tirés aléatoirement de la population. Ces mêmes auteurs dans un autre travail de recherche [85] utilisent une sélection par tournoi pour identifier les deux parents. Tandis que Kurdi [190] propose une méthode de sélection par migration « *migration selection method* » qui repose sur une sélection par roulette. Cette procédure sélectionne plusieurs chromosomes issus de plusieurs populations (sous-populations) qui après croisement et mutation sont migrés vers d'autres sous-populations. Le but de cette approche est de diversifier les différentes sous-populations de l'algorithme génétique.

Une fois les chromosomes identifiés, l'AG explore l'espace de recherche en générant de nouvelles solutions à travers le croisement et la mutation. Le croisement crée les nouveaux chromosomes (fils) en se basant sur les chromosomes parents (échange partiel d'information). Dans la littérature on dénombre un grand nombre de variantes de croisement. Par exemple, Wang & Zheng [74] proposent une procédure de mutation qui consiste à diviser les tâches aléatoirement en deux groupes A_1 et A_2. Le premier chromosome fils est créé en recevant les tâches présentes dans le groupe A_1 et en suivant l'ordre d'apparition de ces tâches du premier chromosome parent et continue avec les tâches dans A_2 suivant l'ordre des tâches du second chromosome parent. Quant au second fils, il est créé en utilisant la même procédure mais en inversant les rôles de A_1 et A_2. C'est-à-dire, commencer par copier les gènes dans A_2 suivant l'ordre du premier parent, et de continuer avec les tâches du groupe A_1 selon l'ordre du second parent. D'autres contributions se sont focalisées sur des procédures de croisement plus connues comme le croisement à un point (*one-point*), le croisement à deux points (*two-points*) [85], [202], ou le croisement basé sur l'ordre (*order-based*) [190]. Ce dernier consiste à déterminer aléatoirement deux points de coupure (*cut-points*) et les gènes se trouvant entre ces deux points sont copiés vers les chromosomes fils. Chaque chromosome fils est ensuite complété (remplir

les gènes manquants à gauche et à droite) en utilisant les gènes de l'un des parents. En ce qui concerne la mutation, elle tente de diversifier la population et d'éviter la convergence prématurée de la population en faisant évoluer les chromosomes. Cette procédure consiste par exemple à changer aléatoirement les valeurs des gènes ou en échangeant les positions de deux gènes choisis aléatoirement. Par exemple, Kurdi [190] dans son algorithme génétique utilise trois procédures de mutations par échange (*swap*), insertion, ou inversion.

Pour la résolution du problème Job Shop, plusieurs approches sollicitant l'algorithme génétique ont été proposées dans la littérature. On cite notamment Kuczapski et al. [84] qui concentrent leurs efforts sur la génération de la population initiale afin d'améliorer les résultats de l'AG. Les auteurs incluent ainsi lors de la génération des règles de priorité composées « *Composite Dispatching Rules* » basées sur une pondération des règles de priorité « *weighted sum of priority rules* ». Avec une population initiale composée de bonnes solutions, l'algorithme génétique a plus de chance d'atteindre de meilleures solutions (proche de l'optimum) plus rapidement comparé aux AG reposant uniquement sur une population générée aléatoirement. Dans la même perspective, Ritwik & Deb [86] optimisent les résultats de l'algorithme génétique pour la résolution du problème Job Shop en améliorant la représentation basée sur les opérations (*operation-based encoding*). Les auteurs mènent ainsi une étude comparative sur plusieurs problèmes Job Shop en changeant le nombre de tâches et/ou de machine prouvant la supériorité de la nouvelle représentation face à d'autres présentées dans la littérature.

Hasan et al. [85] proposent une hybridation de l'AG avec une technique de recherche locale appelée « *Shifted Gap-Reduction* » (*SGR*) pour la réduction du gap (temps d'inactivité des machines) entre les tâches afin d'optimiser le temps d'exécution total dans un problème Job Shop. Le but du SGR est d'améliorer les solutions créées durant la génération de la population initiale en réduisant ou en éliminant tous les gaps en faisant avancer d'autres opérations (décalage à gauche). Cette opération doit ainsi se faire tout en respectant les contraintes liées aux tâches et machines. L'AG proposé repose sur une représentation par génotype, une sélection par tournoi, un croisement à deux points, et d'une mutation « *bit-flip* » (changement de valeur d'un gène). Les auteurs suggèrent aussi une amélioration de leur AG pour réagir aux pannes machines (ordonnancement prédictif et ordonnancement réactif). Les expérimentations sont menées sur un problème Job Shop composé de 15 tâches et 15 machines.

Dans le même ordre d'idées, Cheng et al. [203] proposent un algorithme évolutionnaire hybride « *Hybrid Evolutionary Algorithm* » incluant un algorithme de recherche tabou. L'objectif du Job Shop à résoudre est de réduire le temps d'exécution total. D'après les auteurs, la combinaison AG/TS permet de produire de meilleures solutions réduisant ainsi le makespan. Cette approche est testée sur plusieurs benchmarks avec nombre de tâche variant de 6 à 30 et un nombre de machines oscillant entre 5 à 15. Kurdi [190] propose un algorithme « *Hybrid Island Model Genetic Algorithm* » génétique composé de plusieurs populations (ou sous-populations) avec comme but de retarder la convergence rapide de l'AG et d'explorer un maximum de solutions. Les chromosomes à faire évoluer sont tirés des différentes sous-populations en utilisant une sélection par tournoi. L'auteur utilise un croisement « *order-based* » et trois procédures de mutations par échange, insertion, ou inversion. L'objectif du Job Shop est de minimiser le temps d'exécution total, alors que les expérimentations sont menées

sur 76 problèmes benchmarks. L'AG proposé est comparé avec 15 autres algorithmes issus de la littérature démontrant sa supériorité.

5.3. Problématique de l'allocation des règles de priorité

Comme annoncé dans le Chapitre 4, trouver la meilleure règle de priorité peut s'avérer lent et couteux en temps de calcul car aucune règle ne surpasse les autres dans différents scénarios ou pour différents problèmes. Pour identifier la meilleure, la simulation est considérée pour tester toutes les règles susceptibles d'obtenir de bons résultats et identifier celle qui optimise la fonction objectif. Plus il y a de règles considérées, plus il faudra de temps de calcul pour les tester, rendant l'identification et la sélection plus lent. En plus, si pour chaque machine une règle de priorité différente est envisagée, alors le nombre de combinaisons de q à q^M, nécessitant ainsi encore plus de temps de calcul et d'évaluation [201]. Par exemple, pour un problème Job Shop composé de 6 machines et où on considère 10 règles de priorité, le nombre de combinaisons à tester est de 10^6 rendant l'utilisant d'une technique de force brutale (*brute-force*) très lente et couteuse.

Comme alternative, il est possible de recourir à la fouille de données comme suggéré dans la littérature [22], [99] afin d'accélérer la sélection de règles de priorité. La fouille de données se repose sur des données issues d'expérimentations passées et est capable de créer un modèle de décision qui utiliserait les caractéristiques des tâches et/ou machines afin d'allouer ou de changer de règle de priorité. Son principal objectif est de minimiser/maximiser la fonction objectif et de prendre en considération les évènements inattendus tel que les pannes machine ou l'arrivée de nouvelles tâches. En outre, et contrairement aux approches basées sur la simulation [19], la fouille de données peut retrouver la meilleure règle de priorité en temps-réel évitant tout temps d'attente.

Le data mining repose sur des données utiles et utilisables qui peuvent être transformées en connaissance. Dans cette perspective, Habib Zahmani et al. [99] créent une base de données en simulant plusieurs problèmes Job Shop avec les meilleurs ensembles de règles de priorité. Les auteurs testent toutes les 256 combinaisons possibles en menant les expérimentations sur des problèmes composés de 4 machines et en considérant 4 règles de priorité (FIFO, LIFO, SPT, et LPT). Néanmoins, si le nombre de règle ou de machines considérés venait à augmenter l'espace de recherche augmenterait aussi d'une manière rapide rendant l'utilisation de la force brute impossible. Dans cette optique, un algorithme génétique est proposé dans cette approche afin de résoudre les problèmes Job Shop en affectant des règles de priorité aux machines de l'atelier. Le but de l'AG est d'explorer l'espace de recherche d'une manière intelligente et efficace. Une telle approche a déjà été proposée dans la littérature par Korytkowski et al. [19] où un algorithme génétique identifie les meilleures règles de priorité. Chaque chromosome est une liste de valeur entières oscillant 0 et 6, et où une valeur désigne l'une des règles de priorité considérées. L'AG développé repose sur un croisement à deux points, une mutation par échange, et une combinaison de sélections par tournoi et par roulette (en anglais « *roulette wheel selection* »). Le critère d'optimisation considéré consiste à minimiser le retard moyen et la moyenne du temps d'écoulement. Cependant, une telle approche ne peut être considérée pour l'ordonnancement en temps-réel car l'AG a besoin d'une certaine période de temps pour trouver

une bonne solution, qui se répercute négativement sur la fonction objectif dans l'ordonnancement en ligne. Afin de surmonter cette difficulté nous proposons d'utiliser la fouille de données pour identifier en temps-réel les meilleures règles de priorité à utiliser.

5.4. Approche proposée pour l'allocation des règles de priorité

Le système proposé dans cette approche est capable d'attribuer différentes règles de priorité aux machines à partir d'un ensemble de règles candidates tout au long de l'horizon de planification. Cette approche comporte un algorithme génétique, un simulateur, et les arbres de décision. La structure générale de ce système est découpée trois principales tâches et est donnée dans la Figure 5.1. La première tâche a pour but de résoudre des problèmes Job Shop et de créer une base de données dédiée à l'apprentissage. Le but de la seconde tâche est d'analyser, de préparer les données, et d'utiliser un algorithme d'arbres de décision pour générer un modèle de décision. Enfin, durant la troisième tâche et en utilisant le modèle créé, de nouveaux problèmes Job Shop sont résolus en attribuant les règles de priorité aux machines en temps-réel.

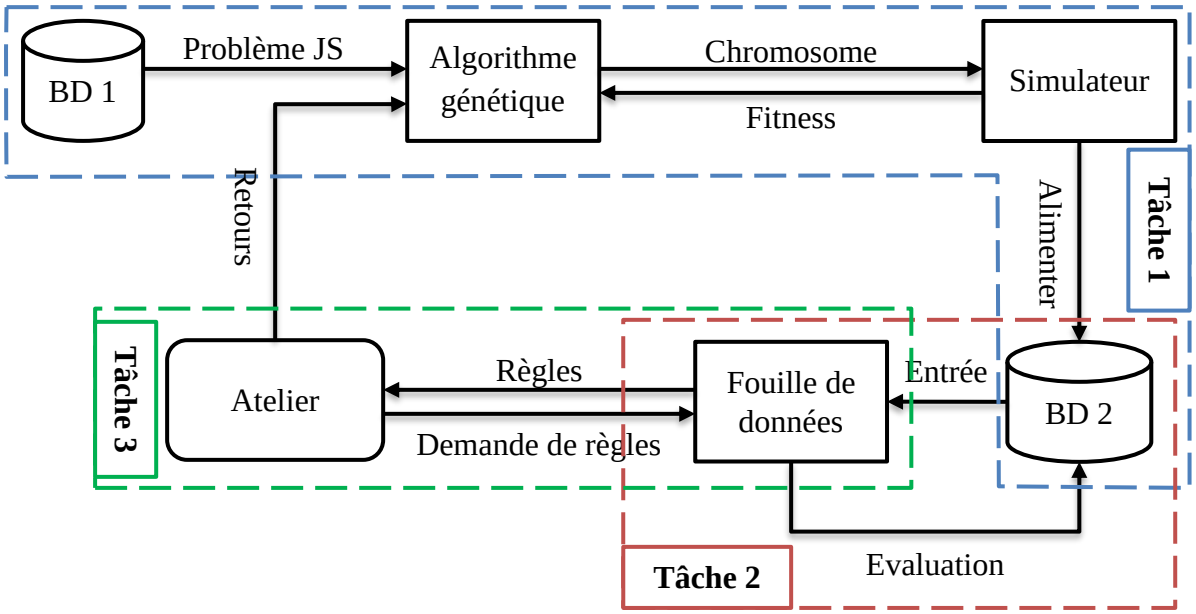

Figure 5.1. Structure générale du système proposé

5.4.1. Tâche 1 : Résolution de problèmes pour l'apprentissage

Durant cette première tâche, un algorithme génétique est couplé avec un module de simulation pour résoudre les problèmes Job Shop comme décrit dans la Figure 5.2. Le but cet AG est d'identifier le ou les meilleurs ensembles de règles de priorité qui minimisent le temps d'exécution total. Quant à la simulation, son rôle est de tester, d'évaluer, et de renvoyer les makespan (fitness) de chaque combinaison de règles (chromosome). Tous les problèmes Job Shop sont enregistrés dans la première base de données « *JS (BD 1)* » qui par la suite sont résolus en utilisant l'AG et le simulateur. Durant la simulation toutes les données sont récupérées et sauvegardées dans la deuxième base « *Données (BD 2)* ». L'ensemble des opérations de cette première tâche sont effectuées hors ligne.

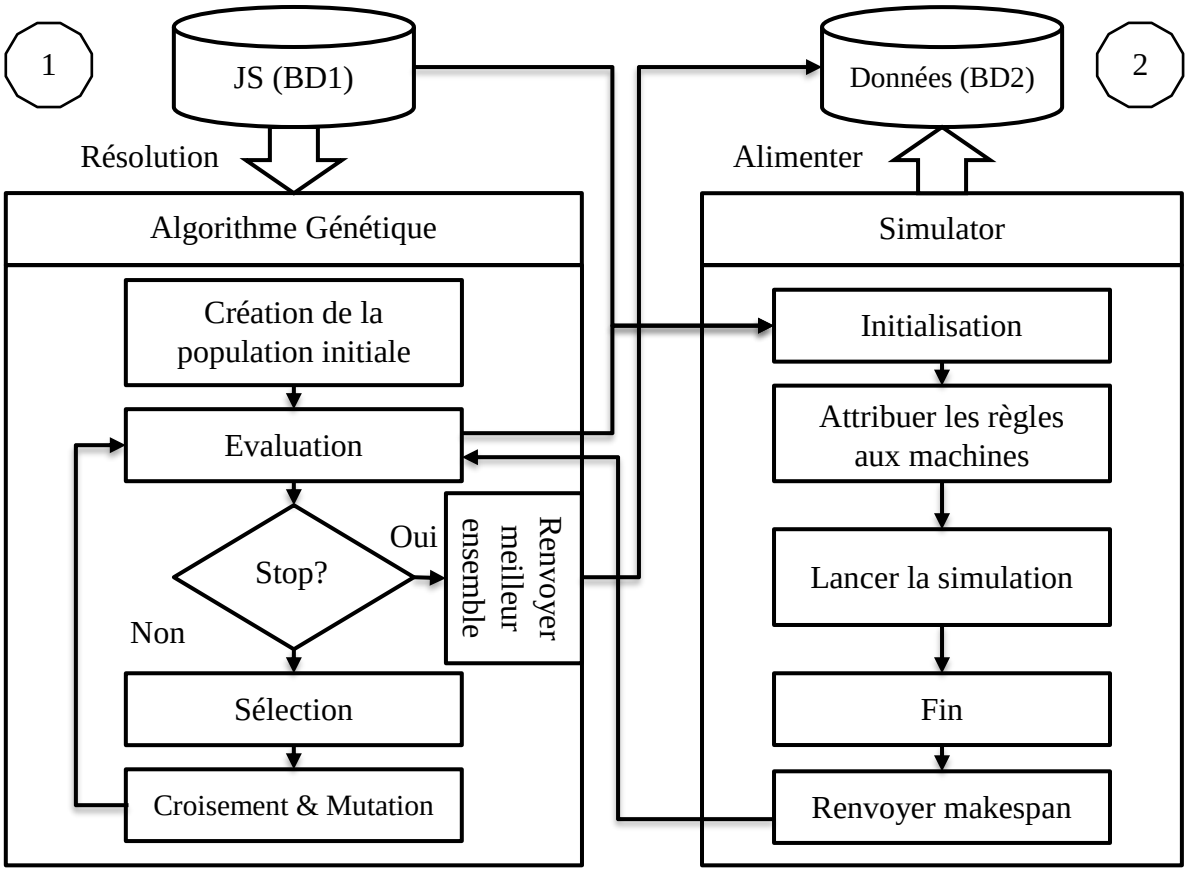

Figure 5.2. Tâche 1 : Résolution des JS en utilisant l'AG et la simulation

5.4.1.1. Algorithme génétique

Le but de cet AG est d'explorer l'espace de recherche, c'est-à-dire l'ensemble de règles de priorité, d'une manière efficace en se basant sur les opérateurs génétiques, au lieu de tester toutes les combinaisons possibles comme fait dans littérature [99] pour des petits problèmes. Dans cette approche, l'objectif est de considérer des problèmes encore plus complexes composés de plus de tâches, de machines, et en considérant plus de règles de priorité.

- **Encodage et règles de priorité**

L'algorithme génétique doit reposer sur un encodage efficace permettant de représenter toutes les solutions possibles, et par conséquent de les atteindre. Cette représentation doit aussi permettre l'utilisation des opérateurs de croisement et de mutation. L'AG proposé est constitué d'une population P_j de P chromosomes (solutions encodées), où j est le numéro de la génération courante. P_0 désigne alors la population initiale. Pour le Job Shop un simple encodage peut être utilisé comme suggéré par Korytkowski et al. [19] où un chromosome est un vecteur de M valeurs entières et chacune de ces valeurs varie entre 0 et $q - 1$. Les règles de priorité considérées dans cette approche sont présentées dans les travaux de Kaban et al. [17], et de Habib Zahmani et al. [201] (cf. Section 4.5 pour plus de détails). L'encodage des chromosomes utilise les valeurs (gènes) suivantes :

- 0 : $PT \times PS$,
- 1 : $PT \times TPT$,
- 2 : $PT \times WT$,
- 3 : $PT \times TWR$,
- 4 : $PS \times TPT$,
- 5 : $PS \times WT$,
- 6 : $PS \times TWR$,
- 7 : $TPT \times WT$,
- 8 : $TPT \times TWR$,
- 9 : $WT \times TWR$.

Les chromosomes de la population initiale sont créés aléatoirement en tenant compte du fait qu'une règle de priorité peut être attribuée à plusieurs machines en même temps. Les chromosomes en doubles dans la population initiale sont éliminés afin d'assurer la diversité de la population. Par la suite, tous les chromosomes sont évalués en utilisant le simulateur et se voient affecter une valeur de makespan (C_{max}).

- **Croisement et mutation**

Une fois la population initiale créée et évaluée, une sélection par tournoi est utilisée pour définir les chromosomes à faire évoluer en utilisant le croisement ou la mutation. Cette sélection consiste à choisir parmi t chromosomes, puisés aléatoirement dans la population, celui qui possède le temps d'exécution total le plus petit. Il est alors retenu pour le croisement ou la mutation. Dans cette approche un croisement à un point est utilisé, où les chromosomes parents sont découpés à la position $\lfloor (M/2)^{ème} \rfloor$. Ensuite, une partie de chacun des deux parents est utilisée pour créer les chromosomes fils. Pour illustrer ce croisement, on suppose un problème Job Shop composé de 6 machines et où l'on considère 10 règles de priorité numérotées de 0 à 9. En utilisant la sélection par tournoi, deux parents sont choisis de la population actuelle P_j afin de créer les nouveaux chromosomes comme décrit dans la Figure 5.3.

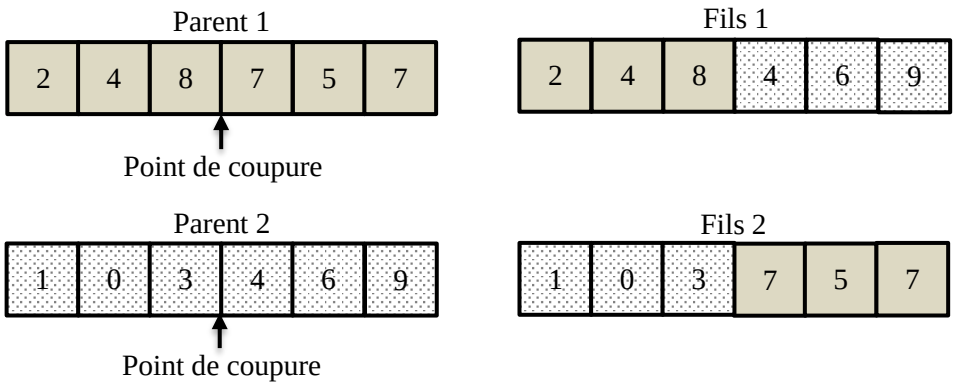

Figure 5.3. Croisement à un point

En ce qui concerne la mutation, une fois qu'un chromosome est identifié via la sélection par tournoi, un gène est sélectionné au hasard est sa valeur changée aléatoirement comme décrit dans la Figure 5.4.

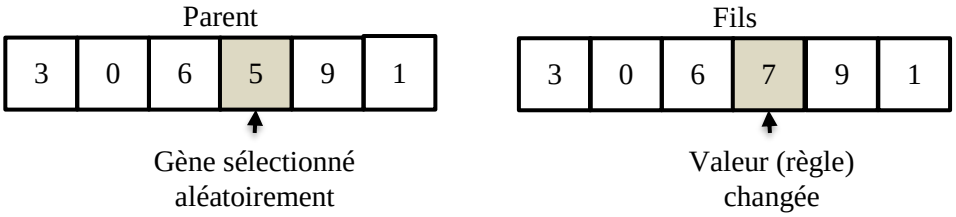

Figure 5.4. Mutation

5.4.1.2. Module de simulation

Le rôle de la simulation est de reproduire les problèmes Job Shop et de mesurer les performances des différentes combinaisons de règles de priorité (revoir Figure 5.2). L'algorithme génétique invoque le module de simulation à chaque fois qu'un nouveau chromosome est créé, durant la génération de la population initiale, croisement, ou mutation. Ce module reçoit ainsi la description complète du problème Job Shop, c'est-à-dire, le nombre de tâche et de machines ainsi que les séquences des tâches stockés dans la première base de données (*JS*). Il reçoit aussi de la part de l'algorithme génétique la combinaison de règles de priorité à tester. Le simulateur reproduit ainsi le problème Job Shop, affecte une règle de priorité à chacune des machines (en utilisant le chromosome), et commence l'exécution des tâches. Durant cette opération, à chaque fois qu'une machine est disponible et qu'une tâche est tirée, suivant l'ordre donnée par la règle de priorité, de sa file d'attente pour être lancée, ses attributs sont recueillis. Ces attributs sont : PT, PS, TPT, WT, et TWR et sont stockés dans la seconde base de données (*Données*) ainsi que la règle de priorité utilisée par cette machine. Une fois que toutes les tâches sont terminées, la simulation s'achève alors et le temps d'exécution total (C_{max}) correspondant est retourné à l'algorithme génétique.

Il est utile de noter que plusieurs instances du simulateur peuvent être lancées en même temps permettant ainsi de tester diverses combinaisons de règles de priorité en parallèle, et à l'algorithme de génétique de fonctionner plus rapidement, particulièrement lors de l'évaluation de la population initiale.

5.4.1.3. Bases de données

Dans cette approche, deux bases de données sont utilisées. La première (*JS*) contient les données relatives aux problèmes Job Shop, c'est-à-dire, le nombre de machines, le nombre de tâches, les séquences des tâches, les temps d'exécution, ainsi que le numéro de machine où une opération doit être exécutée. Cette base de données fournit les informations des problèmes Job Shop au module de simulation afin de le reproduire et de le simuler. D'une autre part, la seconde base (*Données*) est alimentée avec les données issues du simulateur, à savoir, le numéro de la tâche, le temps d'exécution, le temps d'attente, la somme des temps d'exécution, le nombre d'opération restantes, et enfin le temps d'exécution encore requis pour achever la tâche. Dans

cette base de données on trouve aussi le numéro de la machine où la tâche est opérée ainsi que la règle de priorité utilisée pour ordonnancer les tâches sur la file d'attente.

5.4.2. Tâche 2 : Fouille de données pour la prise de décision

Au cours de cette tâche (voir Figure 5.5), l'algorithme C4.5 est utilisé pour créer un arbre de décision en hors-ligne en se basant des expérimentations passées contenues dans la seconde base de données. Le choix des arbres de décision dans cette approche est motivé par leur utilisation répandue dans la littérature, leur simplicité, et leur facilité d'interprétation [22], [56], [201]. Cependant, avant de passer à la phase d'extraction de connaissances, il est impératif de procéder à un prétraitement afin de formater, structurer, et préparer les données. Cette opération est d'une importance capitale pour générer un modèle de décision précis et efficace. Dans le processus d'extraction de connaissances à partir de données (cf. Section 3.2), plusieurs opérations préparatoires sont nécessaires comme la sélection, le nettoyage, ou la transformation des données. Néanmoins, dans l'approche proposée, c'est uniquement la suppression des données dupliquées qui est réalisée parce que les données disponibles dans BD2 sont complètes, ne contiennent aucune valeur manquante, et sont correctes. Une fois le modèle de décision généré, une évaluation est effectuée afin de garantir le bon comportement de ce modèle et de garantir la bonne qualité de ses résultats.

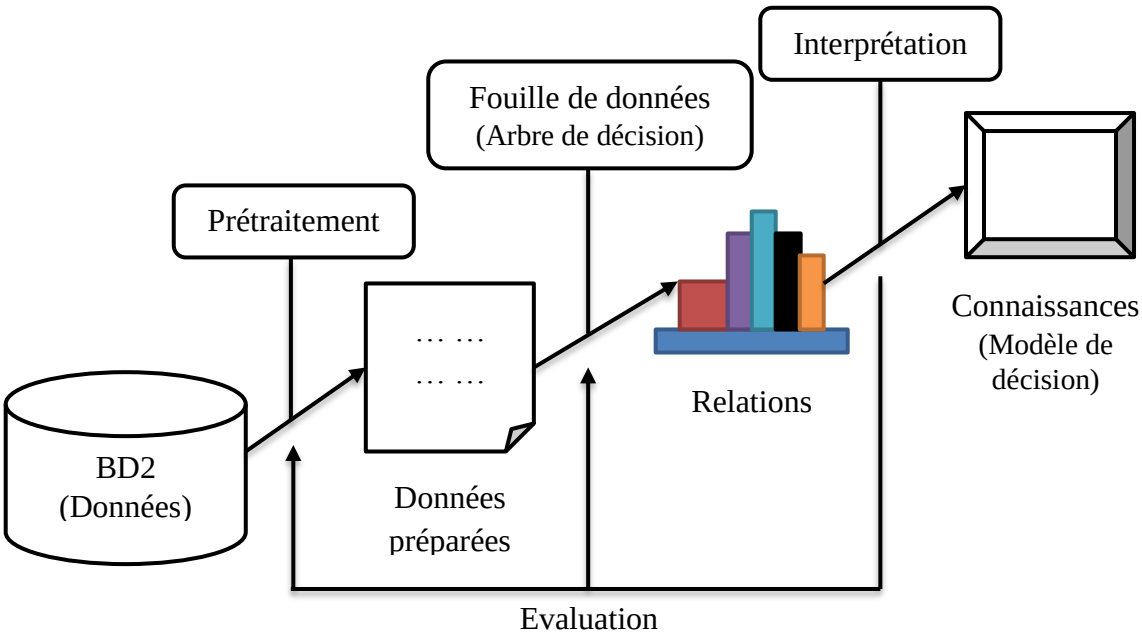

Figure 5.5. Tâche 2 : Extraction de connaissances

Comme expliqué auparavant, à chaque fois qu'une machine est disponible et qu'une tâche est sélectionnée pour exécution, ses attributs sont recueillis et stockés dans la seconde base de données ainsi que la règle de priorité utilisée. Dans le Tableau 5.1 ci-dessous, un exemple des données collectées est illustré.

Tableau 5.1. Exemple de données de simulation recueillies

ID Tâche	ID Machine	PT	PS	TPT	WT	TWR	Règle
9	1	69	5	748	94	215	PTPS
14	10	57	15	665	0	665	PSTPT
5	12	88	13	790	230	700	PTPS
9	3	89	8	748	61	470	WTTWR
7	13	35	10	621	136	418	PSTPT

Les attributs « *ID Tâche* » et « *ID Machine* » sont affichés dans ce tableau uniquement à titre d'information et sont supprimés ainsi que les données (enregistrements) dupliquées lors de la phase de prétraitement. Ensuite, en utilisant l'algorithme C4.5 l'arbre de décision est construit (voir Figure 5.6 ci-dessous) :

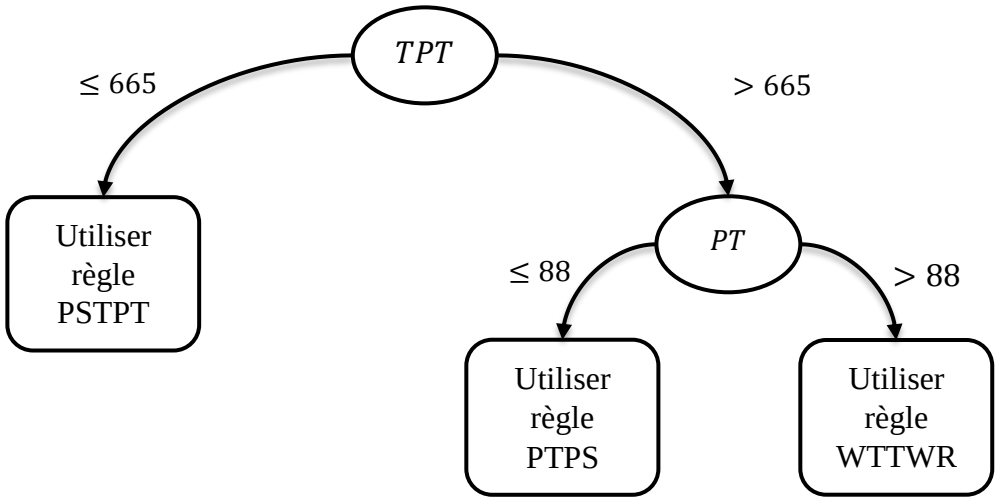

Figure 5.6. Arbre de décision simplifié

L'arbre de décision obtenu est testé et évalué afin d'améliorer son efficacité. Plusieurs mesures de performance sont considérées lors de cette étape, comme le taux d'apprentissage, la précision, et le rappel. Si les résultats de l'évaluation sont jugés non satisfaisants, la procédure est répétée en modifiant les paramètres de l'algorithme C4.5 jusqu'à atteindre les résultats désirés.

5.4.3. Tâche 3 : Allocation de règles de priorité en temps-réel

Cette troisième tâche (voir Figure 5.7) peut être découpée en deux sous-tâches. Le but de la première est résoudre les nouveaux problèmes Job Shop en temps-réel en utilisant l'arbre de décision afin d'allouer les règles de priorité aux machines en fonction des tâches en attente sur leurs files. Durant cette sous-tâche, les module de contrôle surveille et supervise continuellement le système de production. A chaque fois qu'une tâche parmi un ensemble de tâches en attente doit être sélectionnée pour être opérée, le contrôleur invoque l'arbre de décision pour obtenir la meilleure règle de priorité à utiliser. Pour chaque tâche en attente sur la file, l'arbre de décision retourne une règle de priorité à utiliser. Ensuite, le contrôleur choisit la meilleure règle de priorité en fonction de sa popularité (un vote) et qui sera allouée à la

machine. Si aucune règle de priorité n'est plus populaire que les autres (vote ex aequo), une règle est choisie aléatoirement parmi celles suggérées par l'arbre de décision. Si une seule tâche est présente sur la file d'attente de la machine et que cette dernière est disponible, alors la tâche est directement sélectionnée sans passer par l'arbre de décision.

Lors de la seconde sous-tâche les nouveaux problèmes Job Shop sont résolus en hors-ligne en utilisant l'algorithme génétique suivant le même procédé présenté dans la Figure 5.2, où le but est d'identifier les meilleurs ensembles de règles de priorité. Par la suite, les données obtenues lors de la simulation sont rajoutées (alimentation de la deuxième base de données) aux anciennes données afin de créer un meilleur arbre de décision (revoir Figure 5.1, Figure 5.5, et Figure 5.6) pour résoudre les problèmes d'ordonnancement d'une manière plus efficace.

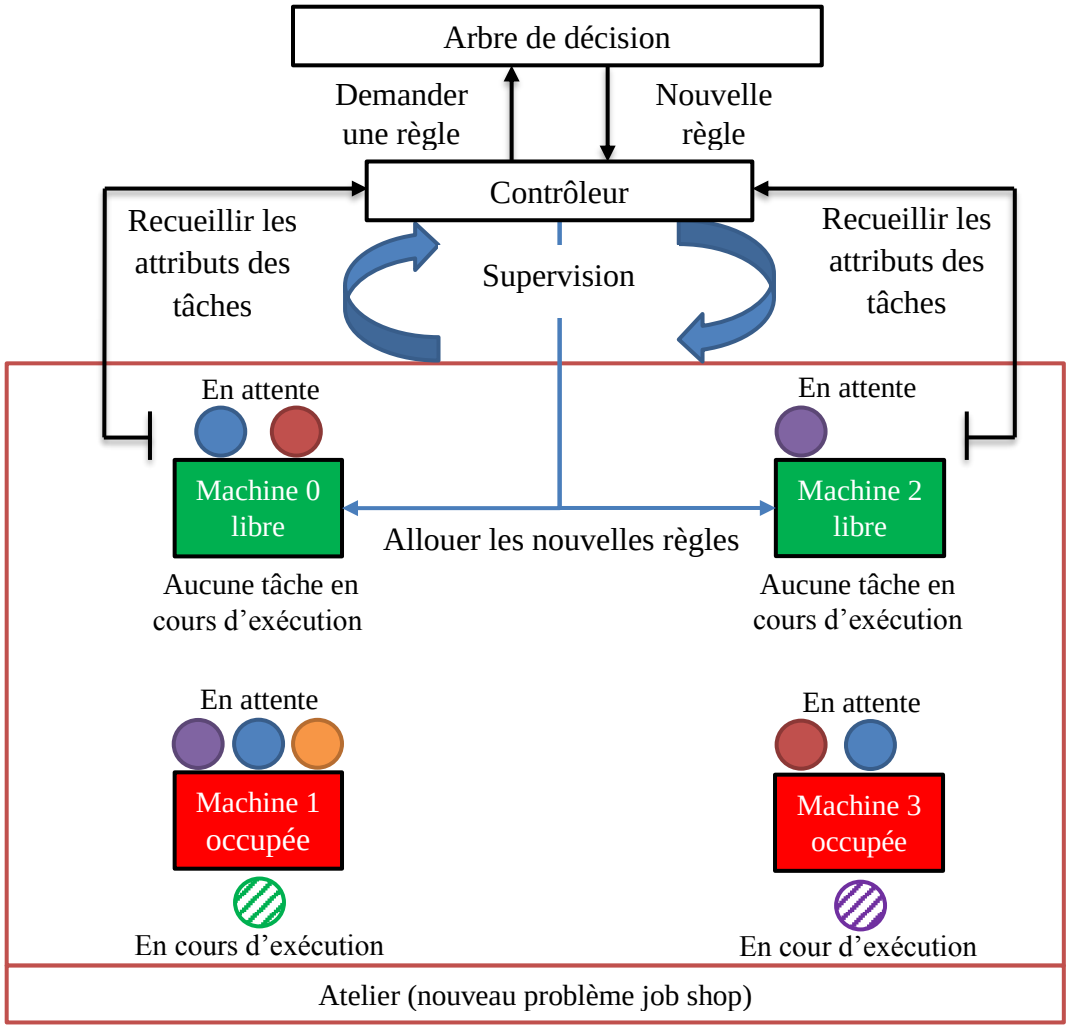

Figure 5.7. Allocation de règles de priorité en temps-réel

L'allocation de règles de priorité en temps-réel à base d'arbres de décision ne requiert pas beaucoup de temps d'exécution contrairement aux autres méthodes de résolution. Cette approche peut aussi être adaptée afin de faire face aux perturbations en changeant d'une manière dynamique les règles de priorité en fonction des attributs des tâches en attente sur les files des machines. L'approche peut aussi être considérée pour résoudre des problèmes encore plus complexes composés d'un grand plus grand nombre de tâches et/ou machines sans pour autant

augmenter considérablement le temps d'exécution requis contrairement aux heuristiques et méta-heuristiques.

5.5. Expérimentations

Dans cette section, nous présentons les problèmes Job Shop, les paramètres de l'algorithme génétique proposé, et la construction de l'arbre de décision. Ces expérimentations sont codées en langage Java et menées sur un ordinateur avec un processeur Core i7 2.4 GHz et doté de 8 GO de RAM.

5.5.1. Problèmes Job Shop

Dans la littérature on recense plusieurs jeux de données de problèmes Job Shop proposés par différents chercheurs [92], [130], [141] afin de comparer et mesure les performances des méthodes de résolutions. Ces jeux de données comportent un large éventail de problèmes Job Shop avec différents niveaux de complexité et considèrent différents objectifs. Durant ces expérimentations, nous considérons les problèmes Job Shop issus de la fameuse OR-Library proposée par Beasley [141] composée de problèmes regroupés en 8 catégories avec 15 et 20 machines et un nombre de tâches allant de 15 à 100 comme présenté dans le Tableau 5.2 ci-dessous.

Tableau 5.2. Problèmes Job Shop considérés

Catégorie	Nombre de tâches	Nombre de machines
15×15	15	15
20×15	20	15
20×20	20	20
30×15	30	15
30×20	30	20
50×15	50	15
50×20	50	20
100×20	100	20

Chacune de ces 8 catégories est composée de 10 instances énumérées de 1 à 10, portant le nombre total d'instances dans cette librairie à 80. Ces instances sont adressées dans ce papier en utilisant la codification $N \times M\_k$ où k est le numéro de l'instance (suivant l'ordre d'apparition dans la librairie), par exemple, « 15×15_3 » est la troisième instance de la catégorie de problèmes 15×15.

Beasley [141] génère les problèmes Job Shop aléatoirement y compris les séquences des tâches. Les temps d'exécutions des opérations varient entre 1 à 99 suivant une distribution uniforme. Une tâche doit nécessairement passer par l'ensemble des machines $\forall\, i \in \{0, 1, \dots, N-1\}, N_i =$ M. Les dates de création et d'arrivée des tâches sont fixées à 0 ; le temps de transport ainsi que les pannes machines sont négligés. Dans l'OR-Library, chaque problème possède une borne supérieure et une borne inférieure qui sont mis à jour continuellement. Ces bornes sont utilisées dans cette approche à des fins de comparaison.

5.5.2. Algorithme génétique

La taille de la population de l'algorithme génétique proposé est fixée à 1000 et ne change pas d'une génération à une autre, et où les nouveaux chromosomes replacent ceux avec le makespan le plus élevé. Durant la génération de la population initiale, les chromosomes similaires sont éliminés afin d'assurer l'hétérogénéité de la population. Ensuite lors de chaque génération, 10% de la population est sélectionné en utilisant la sélection par tournoi c'est-à-dire 100 chromosomes sont choisis pour le croisement et la mutation avec un ratio de 50/50. L'algorithme génétique est arrêté après 25 itérations ou après que le meilleur chromosome de la population reste inchangé après 5 générations consécutives.

5.5.3. Simulateur

Les simulations des problèmes Job Shop sont menées en utilisant l'environnement programmable multi-agent de modélisation « *NetLogo* » développé par Wilensky [204]. NetLogo offre un ensemble d'outils pour la modélisation et la simulation, et peut être facilement adapté pour reproduire les environnements de production. Il fournit un langage de programmation intuitif bien documenté avec une interface utilisateur graphique simplifiée. Dans cette approche un simulateur basé sur NetLogo est mis en place qui peut reproduire n'importe quel problème Job Shop indépendamment de sa taille (nombre de tâches ou de machines). Ce simulateur comporte une interface de supervision et de contrôle, qui permet de suivre en temps-réel le comportement des tâches et machines, les tailles des files d'attentes, entre-autres. Grâce aux outils de NetLogo, le simulateur développé offre la possibilité de lancer plusieurs simulations en parallèle ce qui peut s'avérer très utile lors de la résolution en utilisant l'algorithme génétique le but étant d'accélérer la procédure d'évaluation de l'AG. Durant ces expérimentations, le nombre maximal de simulations en parallèle est fixé à 10 maximisant ainsi l'utilisation du processeur.

5.5.4. Fouille de données

La partie de fouille de données et basée sur le classifieur J48 pour l'extraction de connaissances à partir de la seconde base de données après un processus de prétraitement. J48 est une implémentation de l'algorithme C4.5 disponible dans le logiciel de fouille de données « *Weka* » développé par Hall et al. [178]. Les expérimentations impliquant l'arbre de décision sont menées en deux parties. Durant la première, à partir des 10 instances de chaque catégorie, 7 sont dédiées à l'apprentissage, alors que les 3 restantes sont résolues en utilisant l'arbre de décision obtenu. Quant à la deuxième partie, une catégorie, constituée de 10 instances de problèmes tous de même taille, est utilisée pour l'apprentissage et les 70 instances restantes (10 instances pour les 7 autres catégories) sont résolues en utilisant l'arbre de décision. Le but de ce découpage est de montrer l'impact de différents ensembles d'apprentissage sur les résultats (le temps d'exécution total), mais aussi afin de démontrer que les données de n'importe quelle catégorie de problèmes peuvent être exploitées pour résoudre de nouveaux problèmes. Après plusieurs tests, le nombre minimal d'objets du J48 est fixé à 1, et l'attribut « *unpruned* » est mis à « *true* ».

5.6. Résultats et discussion

Les résultats des expérimentations sont présentés en deux temps. Premièrement, on présente les résultats de l'algorithme génétique, les meilleurs ensembles de règles de priorité trouvés et une comparaison avec les bornes supérieures et inférieures. Deuxièmement, nous présentons les résultats de l'arbre de décision pour l'allocation dynamique de règles de priorité et nous menons une étude comparative avec l'algorithme génétique.

5.6.1. Algorithme génétique

Les résultats de l'algorithme génétique proposé sont présentés à partir du Tableau 5.3 jusqu'au Tableau 5.10. Les colonnes de chacun de ces tableaux contiennent les informations suivantes :

- Pb. : est le numéro du problème (suivant l'ordre d'apparition dans l'OR-Library),
- C_{max} AG : est le temps d'exécution total de la meilleure solution (ensemble de règles de priorité) trouvée par l'AG,
- Sol. Un. : représente le nombre de solutions uniques trouvées lors de la recherche,
- Meilleur C_{max} : sont les meilleurs bornes inférieures et supérieures (si disponible) pour un problème d'ordonnancement donné contenues dans l'OR-Library,
- Ecart C_{max} : est l'écart (donné en pourcentage, voir Équation 5.1 et Équation 5.2) en termes de makespan (C_{max}) entre la solution obtenue par l'AG et la meilleure solution connues, c'est-à-dire, les bornes supérieures (Sup.) et les bornes inférieures (Inf.),
- Bst. Ens. : est le nombre des meilleurs ensembles de règles de priorité (solutions) trouvées par l'AG (celle qui donnent le même meilleur C_{max}),
- Ensemble règles de priorité : est une liste de valeurs entières, où chaque valeur est le numéro de la règle utilisée pour une machine. Le nombre d'entiers varie de 15 à 20 en fonction du nombre de machines dans le problème. Dans le cas où il existe plusieurs bonnes solutions, une seule, choisie aléatoirement, est affichée.

Tableau 5.3. Résultats des problèmes 15×15

Pb.	**C_{max} GA**	**Sol. Un.**	**Meilleur C_{max}**		**Ecart C_{max} (%)**		**Bst Ens**	**Ensemble règles de priorité**
			Inf.	**Sup.**	**Inf.**	**Sup.**		
1	1390	1500	1231	-	12.92	-	1	404456405432641
2	1362	2081	1244	-	9.49	-	2	316934354629114
3	1430	1599	1218	-	17.41	-	1	199425629115863
4	1473	1499	1175	-	25.36	-	19	741769073224245
5	1459	1500	1224	-	19.20	-	1	764785615981875
6	1496	1498	1238	-	20.84	-	2	338929699698999
7	1397	1893	1227	-	13.85	-	5	460015317532495
8	1409	1497	1217	-	15.78	-	3	793800587798402
9	1522	1895	1274	-	19.47	-	1	211550914784320
10	1490	1897	1224	-	21.73	-	1	125549953918464

Par exemple, lors de la résolution de la première instance du problème job shop 15×15 l'algorithme génétique explore 1500 solutions uniques parmi lesquelles une seule meilleure solution avec un makespan de 1390. L'écart entre les solutions de l'algorithme proposé et les meilleures valeurs de makespan connues est donné par :

$$Ecart_{Inf.} = \big(AG(i) - Inf(i)\big)/Inf(i) \quad (5.1)$$

$$Ecart_{Sup.} = \big(AG(i) - Sup(i)\big)/Sup(i) \quad (5.2)$$

Où $AG(i)$ est le C_{max} obtenue par l'algorithme génétique proposé lors de la résolution d'un problème i, $Inf(i)$ est $Sup(i)$ sont les bornes inférieure et supérieure de ce problème dans l'OR-Library.

Tableau 5.4. Résultats des problèmes 20×15

Pb.	C_{max} GA	Sol. Un.	Meilleur C_{max}		C_{max} Gap (%)		Bst Ens	Ensemble règles de priorité
			Inf.	Sup.	Inf.	Sup.		
1	1670	1598	1323	1357	26.23	23.07	1	116488458947817
2	1664	1893	1351	1367	23.17	21.73	2	409172709921006
3	1629	2091	1282	1342	27.07	21.39	1	011994721322344
4	1598	2292	1345	-	18.81	-	1	177492618134978
5	1669	1899	1304	1339	27.99	24.65	1	389616691603302
6	1752	1500	1304	1360	34.36	28.82	1	776036347007880
7	1732	1697	1462	-	18.47	-	1	198239252597273
8	1671	2294	1369	1396	22.06	19.70	3	499114920067153
9	1675	1500	1304	1332	28.45	25.75	2	363123494194278
10	1665	1792	1318	1348	26.33	23.52	1	540327719427009

Tableau 5.5. Résultats des problèmes 20×20

Pb.	C_{max} GA	Sol. Un.	Meilleur C_{max}		Ecart C_{max} (%)		Bst Ens	Ensemble règles de priorité
			Inf.	Sup.	Inf.	Sup.		
1	2044	1794	1573	1642	29.94	24.48	1	89761489676037035431
2	1926	1497	1542	1600	24.90	20.38	1	73156009356399030728
3	1881	1500	1474	1557	27.61	20.81	1	31278022618357188001
4	1966	1700	1606	1644	22.42	19.59	4	02237948654449174345
5	1951	1695	1518	1595	28.52	22.32	2	11396573650695980013
6	2062	1496	1558	1643	32.35	25.50	1	38849437867513992872
7	1951	1893	1617	1680	20.66	16.13	3	31460280463297502977
8	1965	1994	1591	1603	23.51	22.58	2	06532851934358982837
9	1967	1500	1525	1625	28.98	21.05	1	19475763181753901959
10	1822	2294	1485	1584	22.69	15.03	1	18716402238481804885

Tableau 5.6. Résultats des problèmes 30×15

Pb.	C_{max} GA	Sol. Un.	Meilleur C_{max}		Ecart C_{max} (%)		Bst Ens	Ensemble règles de priorité
			Inf.	Sup.	Inf.	Sup.		
1	2146	1497	1764	-	21.66	-	1	640837146212255
2	2367	1500	1774	1785	33.43	32.61	1	691837472025740
3	2370	1498	1778	1791	33.30	32.33	1	819502341731659
4	2375	1500	1828	1829	29.92	29.85	1	887871244599718
5	2295	1598	2007	-	14.35	-	2	586157491252456
6	2331	1497	1819	-	28.15	-	2	624369083551452
7	2231	1500	1771	-	25.97	-	1	224976014191127
8	2215	1496	1673	-	32.40	-	1	143529513287912
9	2368	2193	1795	-	31.92	-	1	589888755750335
10	2223	1796	1631	1669	36.30	33.19	2	157630771699647

Tableau 5.7. Résultats des problèmes 30×20

Pb.	C_{max} GA	Sol. Un.	Meilleur C_{max}		Ecart C_{max} (%)		Bst Ens	Ensemble règles de priorité
			Inf.	Sup.	Inf.	Sup.		
1	2596	1600	1876	2005	38.38	29.48	2	57149180536628240056
2	2618	1491	1867	1937	40.22	35.16	1	54087194525643146355
3	2461	1896	1809	1848	36.04	33.17	1	64174682870210301752
4	2619	1498	1927	1979	35.91	32.34	2	41589341706717672134
5	2567	1500	1997	2000	28.54	28.35	2	55029769051950731288
6	2723	1600	1940	2004	40.36	35.88	1	81400023384407218776
7	2596	1600	1789	1894	45.11	37.06	1	82949354091582842994
8	2661	1791	1912	1943	39.17	36.95	2	90594018901293986362
9	2528	1894	1915	1961	32.01	28.91	1	14840930544702438997
10	2523	1799	1807	1924	39.62	31.13	3	47288466865728714428

Tableau 5.8. Résultats des problèmes 50×15

Pb.	C_{max} GA	Sol. Un.	Meilleur C_{max}		Ecart C_{max} (%)		Bst Ens	Ensemble règles de priorité
			Inf.	Sup.	Inf.	Sup.		
1	3301	1500	2760	-	19.60	-	1	372364142350342
2	3317	1498	2756	-	20.36	-	3	300405329157661
3	3221	1500	2717	-	18.55	-	1	374150681097186
4	3334	2384	2839	-	17.44	-	1	046094008875848
5	3265	1497	2679	-	21.87	-	1	400913588273258
6	3412	1794	2781	-	22.69	-	1	624965158881997
7	3474	1498	2943	-	18.04	-	1	262407862476978
8	3604	1899	2885	-	24.92	-	1	577209867198295
9	3271	2191	2655	-	23.20	-	1	195259950151303
10	3315	1893	2723	-	21.74	-	1	480302912801163

Tableau 5.9. Résultats des problèmes 50×20

Pb.	C_{max} GA	Sol. Un.	Meilleur C_{max}		Ecart C_{max} (%)		Bst Ens	Ensemble règles de priorité
			Inf.	Sup.	Inf.	Sup.		
1	3771	1500	2868	-	31.49	-	1	36619408184480834671
2	3702	1700	2869	-	29.03	-	1	21843833437925220632
3	3600	1497	2755	-	30.67	-	1	21264703356511281207
4	3466	1500	2702	-	28.28	-	1	29027575194129379928
5	3404	2591	2725	-	24.92	-	1	01421993393153843895
6	3504	2492	2845	-	23.16	-	1	21825331639560173868
7	3600	1798	2825	-	27.43	-	3	43037205027273528234
8	3543	1500	2784	-	27.26	-	1	02091957960424510698
9	3802	1600	3071	-	23.80	-	1	67006628513568434134
10	3741	1696	2995	-	24.91	-	1	97422280927445352536

Tableau 5.10. Résultats des problèmes 100×20

Pb.	C_{max} GA	Sol. Un.	Meilleur C_{max}		Ecart C_{max} (%)		Bst Ens	Ensemble règles de priorité
			Inf.	Sup.	Inf.	Sup.		
1	6175	1898	5464	-	13.01	-	1	04917648851800699580
2	6101	1896	5181	-	17.76	-	1	15273177859143946327
3	6533	1500	5568	-	17.33	-	1	11128961684084628592
4	6396	2297	5339	-	19.80	-	1	63066266784842543312
5	6254	1492	5392	-	15.99	-	1	46657562188564657312
6	6362	1888	5342	-	19.09	-	1	43384641889048776015
7	6403	1897	5436	-	17.79	-	1	07177951277348580000
8	6283	1498	5394	-	16.48	-	2	80519442126276868104
9	6214	1600	5358	-	15.98	-	1	74940951631785172841
10	6211	1500	5183	-	19.83	-	1	30068632906987748476

Il est important de relever le nombre de solutions uniques trouvées par l'algorithme génétique lors de la résolution donné dans ces tableaux. Pour les problèmes Job Shop constitués de 15 machines, ce nombre varie entre 1496 et 2384 pour nombre total de solutions de l'ordre de $q^M = 10^{15}$. Alors que pour les problèmes à 20 machines le nombre de solutions uniques est compris dans une fourchette de 1491 à 2591 et où le nombre total de solutions est égal à 10^{20}.

Pour une meilleure compréhension, nous regroupons les résultats de l'algorithme génétique proposé dans le Tableau 5.11 ci-dessous. Ce tableau contient la moyenne du temps d'exécution total obtenu par l'AG, les moyennes des bornes supérieures et inférieures, ainsi que l'écart moyen.

Tableau 5.11. Résumé des résultats de l'algorithme génétique

Catégorie de problèmes	Moy. C_{max} AG	Moy. Meilleur C_{max}		Moy. Ecart C_{max} (%)	
		Inf.	Sup.	Inf.	Sup.
15×15	1442.8	1227.2	-	17.60	-
20×15	1672.5	1336.2	1355.1	25.29	23.58
20×20	1953.5	1548.9	1617.3	26.16	20.79
30×15	2292.1	1784.0	1768.5	28.74	31.99
30×20	2589.2	1883.9	1949.5	37.54	32.84
50×15	3351.4	2773.8	-	20.84	-
50×20	3613.3	2843.9	-	27.10	-
100×20	6293.2	5365.7	-	17.31	-
Écart moyen (%)				**25.07**	**26.65**

En se basant sur ces résultats, on peut conclure que l'utilisation des règles de priorité combinées avec un algorithme génétique est une alternative intéressante car elle permet de produire des solutions de bonne qualité où l'AG a un écart moyen de l'ordre de 25.07% comparé aux meilleures solutions connues. Cependant, l'identification des meilleures règles de priorité en utilisant l'algorithme génétique reste relativement couteuse en temps d'exécution bien que le nombre maximal de générations (25 générations seulement) et la taille de la population (100 chromosomes) soient petits. Par exemple, pour résoudre les problèmes 100×20 (problèmes les plus difficiles) le temps d'exécution s'élève à 28 heures, environ 2.8 heures par problème. Cet état de fait est en grande partie lié au module de simulation, où une évaluation d'un chromosome nécessite 0.8 à 7 secondes en fonction de la taille du problème. D'un autre côté, il est à noter qu'il est possible d'améliorer les résultats de l'AG et de réduire l'écart avec les meilleures solutions connues en augmentant la taille de population ainsi que le nombre de générations aux dépends du temps de calcul requis.

5.6.2. Fouille de données

Comme décrit dans la Section 5.4.1, lors de la résolution de problèmes Job Shop les attributs des tâches sont sauvegardés dans la seconde base de données. Ces attributs des tâches en plus de la règle de priorité utilisée sont collectés à chaque fois qu'une tâche est choisie pour être exécutée sur machine quelconque. Il est utile de noter que seules les meilleures combinaisons de règles de priorité sont considérées lors de l'apprentissage et le reste est éliminé. La création de l'arbre de décision se fait à base de plusieurs jeux de données durant deux principales expérimentations.

5.6.2.1. Apprentissage 70

Lors de la première expérimentation « *Apprentissage 70* » (pour 70% des données dédiés à l'apprentissage), les instances de chaque catégorie de problèmes, 15×15, 20×15, 20×20, 30×15, 30×10, 50×15, 50×20, et 100×20 sont divisées en deux groupes. Le premier contient 7 problèmes (de 1 à 7) et le second les 3 restants (de 8 à 10). Le premier groupe est utilisé pour

la phase d'apprentissage et de construction de l'arbre de décision. Quant aux problèmes du second, ils sont résolus en utilisant l'arbre de décision construit et permettent d'évaluer son comportement lorsqu'il est conforté à de nouveaux problèmes.

Après la création de l'arbre de décision plusieurs mesures sont calculées qui permettent d'apprécier et d'analyser les performances de ce dernier ainsi que de s'assurer de la bonne classification des tâches en utilisant l'ensemble d'apprentissage. Ces mesures sont :

- Vrai Positif (VP) : le nombre de résultats positifs correctement classés sur le nombre total de positifs pour chaque classe (règle de priorité),
- Faux Positif (FP) : le nombre de résultats négatifs incorrectement classés sur le nombre total de négatifs pour chaque classe (règle de priorité),
- Précision : nombre d'instances correctement classées sur le nombre total d'instances prédites par classe,
- Rappel : nombre d'instances correctement classées sur le nombre total d'instances positives par classe,
- F-Mesure : est une mesure qui permet d'évaluer l'arbre de décision et est données par l'Équation 5.3 :

$$\mathrm{F-Mesure} = \frac{2 \times \mathrm{Rappel} \times \mathrm{Précision}}{\mathrm{Rappel} + \mathrm{Précision}} \tag{5.3}$$

Les valeurs de VP, FP, précision, rappel, et F-Mesure pour chaque catégorie de problèmes sont affichés dans le Tableau 5.12.

Tableau 5.12. Mesure de performance de l'ADD en utilisant 70% des données

Catégorie de problèmes	Moy. VP (%)	Moy. FP (%)	Moy. Précision (%)	Moy. Rappel (%)	Moy. F-Mesure (%)
15×15	57.10	4.69	57.76	57.10	56.94
20×15	95.85	0.45	96.17	95.85	95.99
20×20	85.08	1.60	85.51	85.08	84.97
30×15	95.37	0.52	95.78	95.37	95.56
30×20	95.49	0.52	95.58	95.49	95.53
50×15	86.15	1.53	86.78	86.15	86.32
50×20	96.79	0.37	97.00	96.79	96.89
100×20	96.46	0.39	96.54	96.46	96.49
Moyenne	**88.54**	**1.26**	**88.89**	**88.54**	**88.59**

Le processus de création de l'arbre de décision est répété à plusieurs reprises en changeant les paramètres de l'algorithme afin d'améliorer l'ensemble des mesures de performance. La création de l'arbre de décision est une tâche cruciale qui a une grande influence sur la qualité des solutions obtenues lors de la résolution des problèmes d'ordonnancement. A partir du Tableau 5.12, on peut déduire que les problèmes d'ordonnancement avec un grand nombre de

tâches améliorent nettement les mesures de performance, ceci est dû au nombre d'enregistrements obtenus en considérant ces problèmes. Par exemple, pour les catégories de problèmes 50×20 et100×20, la valeur de F-Mesure est égale à 96.89 et 96.49% respectivement. L'arbre de décision créé et ensuite utilisé pour résoudre les nouveaux problèmes d'ordonnancement comme expliqué dans la Figure 5.7. A chaque fois qu'une machine est libre et qu'il existe au moins deux tâches en attente, le module de contrôle invoque l'arbre de décision pour classer ces tâches et obtenir une règle de priorité correspondante à chacune des tâches. Ensuite, la meilleure est identifiée via un vote, et est allouée à la machine. Si une seule tâche est en attente pour la machine, elle est directement sélectionnée sans recourir à l'arbre de décision. Pour illustrer les résultats de l'arbre de décision, l'écart entre celui-ci et l'algorithme génétique proposé est donné dans le Tableau 5.13 ci-dessous en se basant l'Équation 5.4. $ADD(i)$ est la valeur du makespan obtenu en utilisant l'arbre pour résoudre un problème i.

$$Ecart_{ADD} = \big(ADD(i) - GA(i)\big)/GA(i) \tag{5.4}$$

Tableau 5.13. Résolution des problèmes en utilisant 70% des données pour l'apprentissage

Catégorie de problèmes	**Taux d'apprentissage (%)**	**Problèmes résolus avec l'ADD**	**C_{max} ADD**	**C_{max} GA**	**Ecart C_{max} (%)**
		15×15_8	1543	1409	9.51
15×15	57.80	15×15_9	1908	1522	25.36
		15×15_10	1549	1490	3.96
		20×15_8	1940	1671	16.10
20×15	96.04	20×15_9	1957	1675	16.84
		20×15_10	1954	1665	17.36
		20×20_8	2098	1965	6.77
20×20	85.68	20×20_9	2336	1967	18.76
		20×20_10	2179	1822	19.59
		30×15_8	2684	2215	21.17
30×15	95.39	30×15_9	2881	2368	21.66
		30×15_10	2729	2223	22.76
		30×20_8	3145	2661	18.19
30×20	95.34	30×20_9	2927	2528	15.78
		30×20_10	2849	2523	12.92
		50×15_8	4130	3604	14.59
50×15	86.29	50×15_9	3563	3271	8.93
		50×15_10	3922	3315	18.31
		50×20_8	3770	3543	6.41
50×20	96.77	50×20_9	4463	3802	17.39
		50×20_10	4060	3741	8.53
		100×20_8	6971	6283	10.95
100×20	96.47	100×20_9	6776	6214	9.04
		100×20_10	6740	6211	8.52
Ecart moyen (%)					**14.56**

Durant ces expérimentations, l'apprentissage en utilisant les problèmes 100×20_1 à 100×20_7 donne l'écart le plus faible de l'ordre de 9.50% alors que l'écart en utilisant les instances de problèmes constitués de 30 tâches et de 15 machines est de 21.87% en comparaison avec les résultats de l'algorithme génétique. Alors que la comparaison de l'arbre de décision avec les bornes supérieures et inférieures donne un écart moyen de 44.23 et 46.58% respectivement.

5.6.2.2. Apprentissage 100

Durant cette deuxième partie des expérimentations, les instances de chaque catégorie de problèmes (c'est-à-dire les 10 instances) sont utilisées pour l'apprentissage (les mesures de performances de l'arbre de décision sont données dans le Tableau 5.14). Les autres 70 problèmes restants (10 instances des 7 catégories restantes) sont résolus en utilisant les différents arbres de décision construits. Les résultats sont donnés dans les tableaux de 5.15 à 5.22 et un résumé récapitulatif en est donné dans le Tableau 5.23.

Tableau 5.14. Mesure de performance de l'ADD en utilisant 100% des données

Catégorie de problèmes	**Moy. VP (%)**	**Moy. FP (%)**	**Moy. Précision (%)**	**Moy. Rappel (%)**	**Moy. F-Mesure (%)**
15×15	64.99	3.82	65.33	64.99	64.76
20×15	92.18	0.95	92.50	92.18	92.32
20×20	88.09	1.27	88.27	88.09	88.04
30×15	94.62	0.56	94.93	94.62	94.76
30×20	87.77	1.40	88.16	87.77	87.86
50×15	89.26	1.19	89.91	89.26	89.46
50×20	96.33	0.41	96.59	96.33	96.44
100×20	96.34	0.42	96.41	96.34	96.37
Moyenne	**88.70**	**1.25**	**89.01**	**88.70**	**88.75**

De même que lors des expérimentations concernant 70% des données, les arbres de décisions doivent être évalués et améliorés si possible en changeant les paramètres de l'algorithme J48 afin d'optimiser les mesures de performance. Les meilleurs arbres de décision sont obtenus en utilisant les problèmes 50×20 et 100×20.

Tableau 5.15. Résolution des problèmes / instances 15×15 pour l'apprentissage

Catégorie de problèmes résolus	**Moy. C_{max} ADD**	**Moy. C_{max} AG**	**Ecart C_{max} (%)**
20×15	1972.3	1672.5	17.90
20×20	2265.9	1953.5	16.01
30×15	2586.6	2292.1	12.89
30×20	3014.2	2589.2	16.43
50×15	3662.7	3351.4	9.32
50×20	3956.2	3613.3	9.42
100×20	6883.3	6293.2	9.37

Tableau 5.16. Résolution des problèmes / instances 20×15 pour l'apprentissage

Catégorie de problèmes résolus	Moy. C_{max} ADD	Moy. C_{max} AG	Ecart C_{max} (%)
15×15	1718.4	1442.8	19.22
20×20	2272.1	1953.5	16.29
30×15	2717.4	2292.1	18.68
30×20	2885	2589.2	11.46
50×15	3707.1	3351.4	10.66
50×20	3990.2	3613.3	10.48
100×20	6760.4	6293.2	7.45

Tableau 5.17. Résolution des problèmes / instances 20×20 pour l'apprentissage

Catégorie de problèmes résolus	Moy. C_{max} ADD	Moy. C_{max} AG	Ecart C_{max} (%)
15×15	1730.5	1442.8	20.13
20×15	1902.8	1672.5	13.74
30×15	2584.9	2292.1	12.81
30×20	2926.6	2589.2	13.02
50×15	3822.4	3351.4	14.12
50×20	3962.2	3613.3	9.68
100×20	6814.8	6293.2	8.27

Tableau 5.18. Résolution des problèmes / instances 30×15 pour l'apprentissage

Catégorie de problèmes résolus	Moy. C_{max} ADD	Moy. C_{max} AG	Ecart C_{max} (%)
15×15	1695.7	1442.8	17.60
20×15	2052.5	1672.5	22.66
20×20	2258.1	1953.5	15.75
30×20	2929.7	2589.2	13.17
50×15	3813	3351.4	13.84
50×20	3980	3613.3	10.21
100×20	6756.6	6293.2	7.36

Tableau 5.19. Résolution des problèmes / instances 30×20 pour l'apprentissage

Catégorie de problèmes résolus	Moy. C_{max} ADD	Moy. C_{max} AG	Ecart C_{max} (%)
15×15	1702.6	1442.8	18.10
20×15	1964.9	1672.5	17.49
20×20	2300.3	1953.5	17.89
30×15	2619.6	2292.1	14.39
50×15	3732	3351.4	11.38
50×20	4033.7	3613.3	11.68
100×20	6762.6	6293.2	7.44

Tableau 5.20. Résolution des problèmes / instances 50×15 pour l'apprentissage

Catégorie de problèmes résolus	Moy. C_{max} ADD	Moy. C_{max} AG	Ecart C_{max} (%)
15×15	1758.1	1442.8	22.01
20×15	1965.1	1672.5	17.52
20×20	2299.9	1953.5	17.80
30×15	2601.5	2292.1	13.54
30×20	2909.7	2589.2	12.43
50×20	4005.6	3613.3	10.94
100×20	6818	6293.2	8.36

Tableau 5.21. Résolution des problèmes / instances 50×20 pour l'apprentissage

Catégorie de problèmes résolus	Moy. C_{max} ADD	Moy. C_{max} AG	Ecart C_{max} (%)
15×15	1708.5	1442.8	18.63
20×15	2016.2	1672.5	20.56
20×20	2336.4	1953.5	19.69
30×15	2699.6	2292.1	17.95
30×20	2874.3	2589.2	11.05
50×15	3747.3	3351.4	11.80
100×20	6870	6293.2	9.16

Tableau 5.22. Résolution des problèmes / instances 100×20 pour l'apprentissage

Catégorie de problèmes résolus	Moy. C_{max} ADD	Moy. C_{max} AG	Ecart C_{max} (%)
15×15	1740	1442.8	20.81
20×15	1930.9	1672.5	15.42
20×20	2334.8	1953.5	19.55
30×15	2595.3	2292.1	13.27
30×20	2931.7	2589.2	13.28
50×15	3874	3351.4	15.59
50×20	4103.2	3613.3	13.56

En se basant sur les résultats obtenus (du Tableau 5.15 au Tableau 5.23), on peut conclure que le processus d'apprentissage peut se faire efficacement en utilisant n'importe quel problème shop en dépit du nombre de tâches et/ou de machines. Par exemple, en utilisant les instances 15×15, l'écart moyen est de 13.05% alors qu'il est de 15.92 en utilisant les problèmes 100×20 pour l'apprentissage, permettant ainsi à l'approche proposée de s'adapter à d'autres problèmes Job Shop. Il est aussi utile de note que l'écart moyen le plus faible en termes de makespan entre l'arbre de décision et l'algorithme génétique est de 7.36% et est obtenu en utilisant les instances 30×15 pour l'apprentissage en tentant de résoudre les problèmes 100×20. Alors que l'écart le plus élevé est égal à 22.66% impliquant le même ensemble d'apprentissage lors de la résolution de problèmes 20×15.

Tableau 5.23. Résumé des résultats

Catégorie de problèmes utilisés pour l'apprentissage	Taux d'apprentissage (%)	Moy. C_{max} ADD	Moy. C_{max} AG	Moy. Ecart C_{max} (%)
15×15	65.58	3477.31	3109.31	13.05
20×15	91.60	3435.80	3076.50	13.46
20×20	88.58	3392.03	3036.36	13.11
30×15	94.99	3355.09	2987.99	14.37
30×20	87.54	3302.24	2945.54	14.05
50×15	89.39	3193.99	2836.66	14.66
50×20	96.34	3178.90	2799.24	15.55
100×20	96.23	2787.13	2416.40	15.92
Moyenne	**88.78**	**3265.31**	**2901.00**	**14.27**

Le meilleur ensemble d'apprentissage comme présenté dans le Tableau 5.23 est obtenue par les instances 15×15 avec un écart moyen de 13.05% alors que le pire ensemble d'apprentissage est constitué de problèmes 100×20 avec un écart moyen de 15.92%, soit une différence de 2.87%. A base de ces remarques, on peut conclure que le choix des données pour l'apprentissage est très important et qui peut avoir un impact positif ou négatif significatif lors de la résolution de nouveaux problèmes en utilisant l'arbre de décision. Enfin, l'arbre de décision a un écart moyen en termes de makespan de 34.22 et 36.78% comparé aux meilleures bornes supérieures et inférieures de l'OR-Library alors que celui de l'AG est de 25.07 et 26.65%.

Il est tout aussi intéressant d'observer que plus le nombre de tâche augmente dans un problème à résoudre par ADD, moindre est l'écart entre ce dernier et l'AG. Par exemple, lors de la résolution des problèmes 15×15, 20×15, 20×20, 30×15, 30×20, 50×15, 50×20, et 100×20 en utilisant les différents ensembles d'apprentissage (différents arbres de décision), l'écart moyen est de 19.50, 17.90, 15.37, 14.79, 12.97, 12.39, 10.85, et 8.20% respectivement. Ceci est dû au nombre de tâches sur les files d'attente des machines. Par exemple, avec un petit nombre de tâches sur une file d'attente, i.e. 2, 3, ou 4 tâches, la sélection de la meilleure règle de priorité en utilisant l'arbre de décision n'est pas efficace car on n'arrive pas déterminer la meilleure règle par vote (pas de règle populaire). Cependant, avec un plus grand nombre de tâches sur les files d'attentes, le vote s'améliore et permet d'identifier de meilleures règles de priorité, rendant l'approche plus efficace et lui permettant de résoudre des problèmes avec un grand nombre de tâches.

Au final, on peut conclure que l'utilisation des arbres de décision est une solution viable pour la sélection des règles de priorité en comparaison avec l'algorithme génétique avec un écart moyen de 14.27%. En plus, l'arbre de décision est capable d'identifier et d'allouer les règles de priorité en temps-réel aux différentes machines de l'atelier contrairement à l'algorithme génétique qui nécessite beaucoup plus de temps de calcul. Par conséquent, pour la résolution de problèmes d'ordonnancement dynamique et afin de mieux réagir aux évènements inattendus, il est intéressant d'opter pour la fouille de données pour la mise en place d'un système d'aide à la décision qui résout les problèmes Job Shop presque aussi efficacement que d'autres méthodes de résolutions tout en étant capable de le faire en temps-réel.

Enfin, comme démontré par Habib Zahmani et al. [201], l'utilisation de plusieurs règles de priorité à la fois permet d'améliorer le makespan contrairement à l'utilisation d'une règle pour toutes les machines (règle unique). Dans le Tableau 5.24 ci-dessous, nous présentons l'écart moyen entre l'algorithme génétique proposé ainsi que l'utilisation des arbres de décisions pour la sélection des règles de priorité comparés à l'utilisation d'une seule règle pour toutes les machines. Par exemple, dans la première ligne du tableau est montré l'écart entre AG et ADD comparé à l'utilisation de la règle $PT \times PS$ dans toutes les machines. L'utilisation de plusieurs règles de priorité améliore le makespan de 20.93% en faveur de l'AG et de 13.13% en faveur des ADD.

Tableau 5.24. AG et ADD vs règle unique

Règle de priorité	**Moy. Ecart C_{max} vs AG (%)**	**Moy. Ecart C_{max} vs ADD (%)**
$PT \times PS$	14.23	6.55
$PT \times TPT$	13.80	6.18
$PT \times WT$	15.64	7.99
$PT \times TWR$	17.95	10.25
$PS \times TPT$	17.50	9.88
$PS \times WT$	22.41	14.52
$PS \times TWR$	25.30	17.34
$TPT \times WT$	25.80	17.86
$TPT \times TWR$	27.74	19.79
$WT \times TWR$	28.93	20.96
Moyenne	**20.93**	**13.13**

5.7. Conclusion et perspectives

Dans cette approche, nous avons développé un système capable d'allouer automatiquement des règles de priorité aux machines de l'atelier en temps-réel pour la résolution du problème Job Shop. Cette approche inclut un algorithme génétique, les arbres de décision, et un simulateur. Les expérimentations montrent que l'algorithme génétique proposé connait un écart moyen de 25.07 et 26.65% comparé aux meilleures bornes inférieures et supérieures. Tandis que l'arbre de décision a un gap de 14.27% en comparaison avec l'AG et un écart moyen de 34.22 et 36.78% comparé aux meilleures bornes inférieures et supérieures fournies dans l'OR-Library.

Cette approche basée sur la fouille de données analyse et extrait de la connaissance à partir de précédents problèmes résolus en utilisant un algorithme génétique couplé avec un module de simulation. La connaissance extraite sous forme d'arbres de décision est par la suite utilisée pour résoudre de nouveaux problèmes Job Shop donnant, en temps-réel, de bons résultats en termes de makespan. Cette approche, d'après les résultats des expérimentations, est intéressante et peut être considérée comme une alternative sérieuse aux méta-heuristiques ou à la simulation pour l'allocation des règles de priorité. En plus, cette approche est capable de fonctionner en ligne et peut être ainsi implémentée dans un système de production réel.

Comme perspectives, de futures recherches devront se focaliser sur la qualité des solutions initiales à la base de l'arbre de décision. Par exemple, l'algorithme proposé dans cette approche

peut être amélioré afin de trouver de meilleures solutions. Il est aussi possible de considérer d'autres méthodes de résolution comme la recherche tabou ou les essaims particulaires.

Une autre possibilité d'amélioration consisterait en l'utilisation d'autres méthodes de la fouille de données, autres que les arbres de décision, pour perfectionner les modèles de décision.

Enfin, et comme démontré par les expérimentations, l'accroissement du nombre de tâches, dans notre modèle basé sur les arbres de décision, permet de réduire significativement le temps d'exécution total ; il serait intéressant, lors de futures expérimentations, de considérer des problèmes avec un plus grand nombre de tâches.

CHAPITRE

6. Extraction de Règles de Priorité pour le Single Machine à l'Aide d'un Algorithme Génétique Modifié et de Fouille de Données

Résumé : Dans ce chapitre nous proposons une nouvelle approche pour la résolution du problème Machine Unique avec somme des retards pondérés. Pour cela il est mis en avant de nouvelles heuristiques, la fouille de données et des algorithmes génétiques. Les techniques de fouille de données sont utilisées pour explorer, analyser et extraire de la connaissance à partir de solutions du problème à Machine Unique. Un algorithme génétique hybride combiné avec des règles de priorité est proposé pour trouver des solutions proches de l'optimal. A base de ces solutions, la fouille de données extrait de la connaissance qui est combinée à son tour avec trois heuristiques proposées pour résoudre de nouveaux problèmes. Les expérimentations montrent l'efficacité et la supériorité de cette approche en comparaison avec des règles de priorité réputées pour leurs bonnes performances. Elle imite le comportement de l'algorithme génétique tout en retenant les points forts des règles de priorité, à savoir, le faible temps de calcul, la réactivité dans l'ordonnancement dynamique.

Mots-clés : Machine Unique · Somme des Retard Pondérés · Algorithmes Génétiques · Fouille de Données · Arbres de Décision · Heuristiques · Règles de Priorité

6.1. Introduction

L'ordonnancement est une des tâches cruciales durant la planification, où le but et d'allouer d'une manière efficace les machines disponibles aux tâches ou aux opérations qui composent ces tâches, tout en essayant de trouver un compromis entre plusieurs objectifs exprimés en termes de coût, temps, responsivité, etc. Lors de cette opération, le responsable de production se repose souvent sur son intuition et son expérience, basée sur les connaissances acquises lors de la résolution de ce type de problèmes par le passé. Cependant, lorsque cette connaissance ne peut être explicitement capturée et mise à profit, n'importe quelle approche proposée se révèlera inefficace car ne pouvant prendre en considération les différents aspects du problème, dû en partie à sa complexité [180]. Cet état de fait pousse ainsi l'expert à s'orienter vers l'intelligence artificielle et la fouille de données afin de mettre à profit les informations issues du système de production lui-même.

Cependant Li & Olafsson [180] affirment que la fouille de données n'est pas très utilisée dans l'ordonnancement, où elle peut notamment apporter des améliorations aux solutions [22], automatiser la sélection des règles de priorité [201], ou générer de nouvelles règles de priorité [24], [35]. Or, en raison de la forte complexité des problèmes d'ordonnancement, de la difficulté de capturer et d'analyser leurs différents aspects ainsi que les gros volumes de données générées par les tâches, machines, capteurs, ou autres, l'intégration de la fouille de données devient une nécessité. Parmi les différentes sources de données dans l'ordonnancement, il convient de mentionner les solutions, c'est-à-dire, les séquences. Ces solutions sont trouvées à l'aide de méthodes de résolution tel que les algorithmes génétiques ou les règles de priorité.

Les algorithmes génétiques ont été adaptées avec succès pour différents problèmes d'optimisation et pour la résolution de problèmes d'ordonnancement, y compris le problème à Machine Unique avec somme des retard pondérés, en anglais « *Single Machine with Total Weighted Tardiness* » (*SMTWT*), comme démontré par Armentano & Mazzini [39], Sevaux & Sörensen [45], ou Ferrolho & Crisóstomo [50]. Dans cette approche [205], nous introduisons un « *Algorithme Génétique Modifié* » (*AGM*), qui inclut plusieurs solutions issues de règles de priorité lors de génération de la population initiale. Ceci, contrairement à l'utilisation de solutions générées aléatoirement, permet à l'algorithme génétique d'atteindre de meilleures solutions et de parvenir dans certains cas à trouver les solutions optimales. Par la suite, la fouille de données est utilisée pour générer de la connaissance issue des solutions de l'AGM. Cette connaissance est ensuite combinée avec trois heuristiques proposées offrant la possibilité de résoudre de nouveaux problèmes d'ordonnancement de la même manière que les règles de priorité. Cette combinaison impliquant connaissances et heuristiques donne de meilleurs résultats en comparaison avec des règles de priorité reconnues pour leurs bonnes performances pour la résolution de problèmes SMTWT.

6.2. Single Machine total weighted tardiness

Le problème à Machine Unique avec somme des retards pondérés noté $1//\sum w_i T_i$ [53], se compose de n tâches (jobs) $J = \{J_0, J_1, \dots, J_{n-1}\}$ qui nécessitent d'être exécutées sur une seule machine. Chaque tâche J_i, aussi notée i, est constituée d'une seule opération avec un temps d'exécution $pt_i > 0$ où $i = 0, \dots, n-1$ et est disponible au temps zéro. L'importance d'une tâche est exprimée en fonction d'un poids positif (*weight*) $w_i > 0$, $i = 0, \dots, n-1$. Une machine ne peut opérer qu'une seule tâche à la fois, et l'exécution d'une tâche ne peut être interrompue et poursuivie plus tard. Chaque tâche i est supposée être achevée avant une date d'échéance (*due date*) d_i et est considérée comme « *hâtive* » (*early*) si sa date de fin d'exécution effective est plus petite que sa date d'échéance. D'un autre côté, on parle de tâche « *tardive* » (*tardy*), si la date de fin d'exécution est supérieure à la date d'échéance, une pénalité est alors assignée à la tâche donnée par $T_i = \max\{C_i - d_i; 0\}$, où C_i est la date de fin d'exécution (*completion time*) de la tâche dans la séquence actuelle. Le but du SMTWT est de trouver une séquence faisable S qui minimise la somme des retards pondérés, en anglais « *Total Weighted Tardiness* » (*TWT*) donnée par l'Équation 6.1 :

$$\min TWT = \sum_{i=0}^{n-1} w_i \times T_i \tag{6.1}$$

Il a été démontré dans la littérature que le problème SMTWT est NP-Difficile [50], [51], [65], [206], par conséquence, l'utilisation de méthodes exactes n'est pas possible en raison du temps de calcul nécessaire qui augmente exponentiellement en fonction de la taille du problème. Ce problème a attiré et intéressé de nombreux chercheurs et a abouti à la mise en place de divers algorithmes et approche pour sa résolution, par exemple, le Branch & Bound [12] pour les problème à petite taille, la recherche tabou, les colonies de fourmis, ou les algorithmes génétiques [3], [43], [50], [53], [55]. Les règles de priorité se sont aussi avérées une bonne alternative pour la résolution de ce problème comme la règle « *Earliest Due Date* » (*EDD*) ou « *Weighted Modified Due Date* » (*WMDD*) [163] qui sont connues pour leurs bons résultats pour les critères d'optimisation basés sur le retard (cf. Section 1.2.4). D'autres méthodes de résolution basées sur l'hybridation sont aussi proposées dans la littérature, comme le travail de Maheswaran et al. [51] qui combinent la règle de priorité « *Modified Due Date* » (*MDD*) avec un mécanisme de recherche locale.

6.3. Algorithmes génétiques et Single Machine : état de l'art

L'algorithme génétique est composé de plusieurs opérations qui sont : la création de la population initiale, l'évaluation des chromosomes, la sélection, le croisement, et la mutation. Durant la génération de la population initiale, les chromosomes peuvent être générés aléatoirement [45], ou en se reposant sur d'autres heuristiques ou méta-heuristiques qui construisent de meilleures solutions, réduisent les chances d'atteindre un optimum local, et permette d'atteindre, idéalement, une solution optimale. C'est le cas notamment de Liu et al. [46] qui utilise cinq règles de priorité lors de la génération de la population initiale. L'algorithme génétique évalue tous les chromosomes de la population et leur affecte une valeur

(fitness) qui permet de déterminer leurs chances de survie dans la prochaine génération via une procédure de sélection. La sélection tente de maintenir les chromosomes avec le plus grand potentiel en faisant appel au croisement et à la mutation. Dans littérature plusieurs techniques de sélection sont adoptées, par exemple, Armentano & Mazzini [39] considèrent une sélection par roulette afin de déterminer l'ensemble de chromosomes à faire évoluer. Sevaux & Sörensen [45] utilisent une sélection par classement (*ranking selection*) qui favorise les chromosomes les mieux classés de la population comme premier parent, alors que le second est choisi aléatoirement. Chou [53] propose une procédure de sélection appelée « *Lowerbound-Base Bias Roulette* » qui donne une plus grande probabilité de sélection au chromosome avec la borne inférieure la plus petite donnée par le Branch & Bound. Kapanoglu & Alikalfa [88] utilisent une sélection par tournoi avec élitisme, où un groupe de chromosomes avec les meilleures valeurs de critère d'optimisation est sélectionné pour la reproduction. Quant à l'élitisme, il consiste à maintenir le meilleur individu de la population et d'assurer sa survie dans la prochaine population en le protégeant contre les opérateurs d'évolution.

Une fois les chromosomes sélectionnés pour le processus d'évolution, l'algorithme génétique utilise les opérateurs de croisement ou de mutation. Dans la littérature on dénombre plusieurs procédures de croisement comme « *Partially Mapped Crossover* », « *Cycle Crossover* », « *Order Crossover* », ou « *Edge Recombination* » [13]. Par exemple, Sevaux & Sörensen [45] utilisent un croisement à un point (*one-point crossover*) qui construit le premier fils en copiant la première partie du premier parent jusqu'à atteindre un point de coupure choisi aléatoirement et complète par les gènes manquants en suivant la séquence donnée par le second parent. La même procédure est répétée pour construire le second fils en inversant les rôles des deux parents. En ce qui concerne la mutation, il est possible d'opter pour un échange de valeur d'un gène, un échange de gènes, ou d'autres techniques afin de diversifier la population.

Pour résoudre le SMTWT de nombreuses approches de la littérature s'orientent vers les algorithmes génétiques. C'est le cas de Armentano & Mazzini [39] qui utilisent l'AG pour résoudre le problème à Machine Unique avec temps de préparations, dates d'échéances, et la somme des retards comme critère d'optimisation. L'algorithme génétique proposé repose sur méthode de croisement modifiée qui se base sur la techniques « *edge recombination crossover* ». Les expérimentations sont menées sur plusieurs ensembles de problèmes avec 30, 32, et 90 instances. Liu et al. [46] utilisent des règles de priorité lors de la génération de la population initiale afin de réduire l'espace de recherche et introduisent une nouvelle procédure de croisement. Lors de expérimentations, l'algorithme génétique proposé est comparé avec « *Descent Method with Zero Interchanges* », « *Descent Method* », *EDD*, et « *Weighted Shortest Processing Time* » (*WSPT*) en utilisant des problèmes à Machine Unique avec minimisation de la somme des retards pondérés composés de 50, 100, 200, et 500 tâches. Les règles de priorité considérées lors de la génération de la population initiale sont : *EDD*, *WSPT*, « *Shortest Processing Time* » (*SPT*), « *Biggest Weight First* », et « *Apparent Urgency* ».

Dans une autre contribution, Ferrolho & Crisóstomo [50] ont développé un outil appelé « *HybFlexGA* » dans lequel ils implémentent et testent plusieurs procédures de croisement et de mutation pour la résolution de benchmarks SMTWT constitués de 40, 50, et 100 tâches. Ces benchmarks sont issues de la fameuse OR-Library [141]. Chou [53] a développé ELGA

« *Experienced Learning Genetic Algorithm* » pour la résolution des problèmes SMTWT. La particularité d'ELGA réside dans le fait qu'il reprend les informations de l'ensemble des chromosomes de la population précédente lors de l'évolution contrairement aux autres AG de la littérature où uniquement les informations des parents sont prises en considération. Pour se faire, l'auteur utilise deux matrices tâche-tâche (*job-job*) et position-tâche (*position-job*) pour créer les nouveaux chromosomes à prendre en considération lors de la génération suivante. Les expérimentations sont menées sur des problèmes à Machine Unique issues de l'OR-Library, où ELGA arrive à atteindre quasiment l'ensemble des solutions optimales.

Azadeh et al. [57] proposent un algorithme génétique pour la résolution d'un problème à Machine Unique avec « *m-familles* » (*m* ensembles de tâches similaires) et trois objectifs à optimiser à la fois (optimisation multi-objectifs). Les auteurs proposent une procédure de croisement appelée « *parameterised uniform crossover* » qui repose sur le jet d'une pièce (biased coin) afin d'identifier le ou les parents à considérer pour créer de nouveaux chromosomes. Les auteurs considèrent une mutation basée sur l'échange de gènes (*swap*). Les objectifs du problème d'ordonnancement sont : la minimisation du retard (*tardiness*), les temps de cycle (*cycle time*), et la maximisation de l'utilisation des machines. Lors des expérimentations, plusieurs ensembles de problèmes sont générés et sont constitués de 10, 20, 50, et 100 tâches, des temps d'exécution variants entre 1 et 5 et un nombre de familles fixé à 3, 5, 10, 20, ou 50. Une étude comparative est effectuée avec les règles de priorité SPT et EDD.

6.4. Problématique de l'extraction de règles de priorité

Comme discuté dans la Section 2.4, les chercheurs ont exprimés un vif intérêt envers les règles de priorité, démontré par le grand nombre de contributions en ce sens. Cet intérêt est très souvent motivé par l'efficacité et le faible temps de calcul des règles de priorité qui a un impact positif sur la qualité des solutions des problèmes d'ordonnancement en termes de rapidité et d'adaptabilité. Ces atouts des règles de priorité ont poussé les chercheurs à aller encore plus loin et à s'intéresser aux principes qui guident les autres de méthodes de résolution, qui sont connues pour leurs meilleures performances en comparaison avec les règles de priorité, et tenter d'analyser et d'exploiter ces principes (cf. Section 3.4.2). En ce sens, plusieurs contributions ont été proposées qui reposent sur la fouille de données afin d'analyser, d'explorer, et d'exploiter les solutions obtenues par des méthodes de résolution comme la recherche tabou [24], [91], les algorithmes génétiques [35], ou autres heuristiques et méta-heuristiques [56], [177]. Le but de cette contribution est de reprendre la même idée proposée dans la littérature afin de mettre à profit la fouille de données et l'utiliser pour reproduire le comportement des méthodes de résolution en transformant leurs solutions en nouvelles règles de priorité. Contrairement aux autres travaux présentés dans l'état de l'art, cette approche s'intéresse plus à la manière d'utilisation de ces nouvelles règles de priorité et à leur adaptation à des problèmes complexes comme le SMTWT.

En détails, dans cette approche [205] on s'intéresse à deux points qui ne sont pas traités dans la littérature. Le premier est lié à la taille et la complexité des problèmes résolus par les travaux proposés dans la littérature. Par exemple, Koonce & Tsai [73] et Shahzad & Mebarki [24], [82], [91] s'intéressent à des problèmes Job Shop composés de 6 tâches et 6 machines. Bien que le

Job Shop soit considéré NP-Difficile, les problèmes 6×6 sont de très petite taille et peuvent être résolus manuellement et très facilement en utilisant les nouvelles règles de priorité extraites par fouille de données. En contrepartie, dans cette approche nous nous intéressons à des problèmes à plus forte complexité, ce qui est le cas du SMTWT. Lors des expérimentations les problèmes considérés sont constitués de 40, 50, ou 100 tâches. Contrairement aux problèmes Job Shop 6×6 ou le pire des cas serait d'avoir les 6 tâches sur la file d'attente d'une machine, avec les problèmes SMTWT considérés le pire des cas serait d'avoir les 40, 50, ou 100 tâches toutes sur la file d'attente. Ainsi il devient difficile de mettre en place et d'utiliser les nouvelles règles de priorité.

Le deuxième point clé traité dans cette approche est lié à la manière d'utiliser les nouvelles règles de priorité. En effet, en tentant de reproduire les approches proposés par Shahzad & Mebarki [82] ou Li & Olafsson [180] on remarque que les règles de priorité extraites sous forme « *si-alors* » ne peuvent construire de bonnes séquences pour différents types de scénarios. Ceci est notamment dû à des erreurs de classification de l'arbre de décision comme expliqué avec l'exemple dans la Section 6.5.4. Ce qui nous a motivé à proposer trois différentes heuristiques afin d'utiliser la connaissance produite (arbre de décision) par la fouille de données et de résoudre des problèmes d'ordonnancement SMTWT en temps-réel tout en essayant de remédier aux problèmes liés à son utilisation.

6.5. Approche proposée pour l'extraction de règles de priorité

L'approche proposée [205] (voir Figure 6.1) est composée de trois principaux modules. Le premier « *module 1* » est un module de résolution et d'optimisation, le second « *module 2* » s'intéresse à l'apprentissage par fouille de données, et le troisième « *module 3* » combine la connaissance obtenue par le second module avec trois heuristiques proposées pour la résolution de nouveaux problèmes d'ordonnancement.

En détails, l'objectif du premier module (cf. Section 6.5.1 et Section 6.5.2) est de résoudre les problèmes SMTWT en utilisant l'algorithme génétique modifié et de sauvegarder la meilleure solution trouvée dans une base de données. L'AGM s'appuie sur plusieurs règles de priorité afin d'améliorer la qualité de la population initiale qui peut avoir un impact positif sur la qualité de la solution finale mais aussi permet de réduire le temps de calcul nécessaire. Par la suite, l'AGM explore l'espace de recherche en faisant évoluer les chromosomes des différentes générations dans le but d'atteindre une solution faisable et de bonne qualité, idéalement optimale, afin de minimiser la somme des retards pondérés.

Quant au second module (cf. Section 6.5.3), les solutions obtenues par l'AGM sont stockées dans la base de données et sont transformées en un fichier contenant une comparaison de toutes les tâches (comparaison tâche-tâche) qui se base sur leurs attributs suivant la procédure décrite par Li & Olafsson [180]. Les données obtenues (fichier) sont ensuite analysées et prétraitées (suppression des données doubles) et un arbre de décision est créé en utilisant l'algorithme C4.5 qui peut aussi être présenté sous la forme de règles « *si-alors* ».

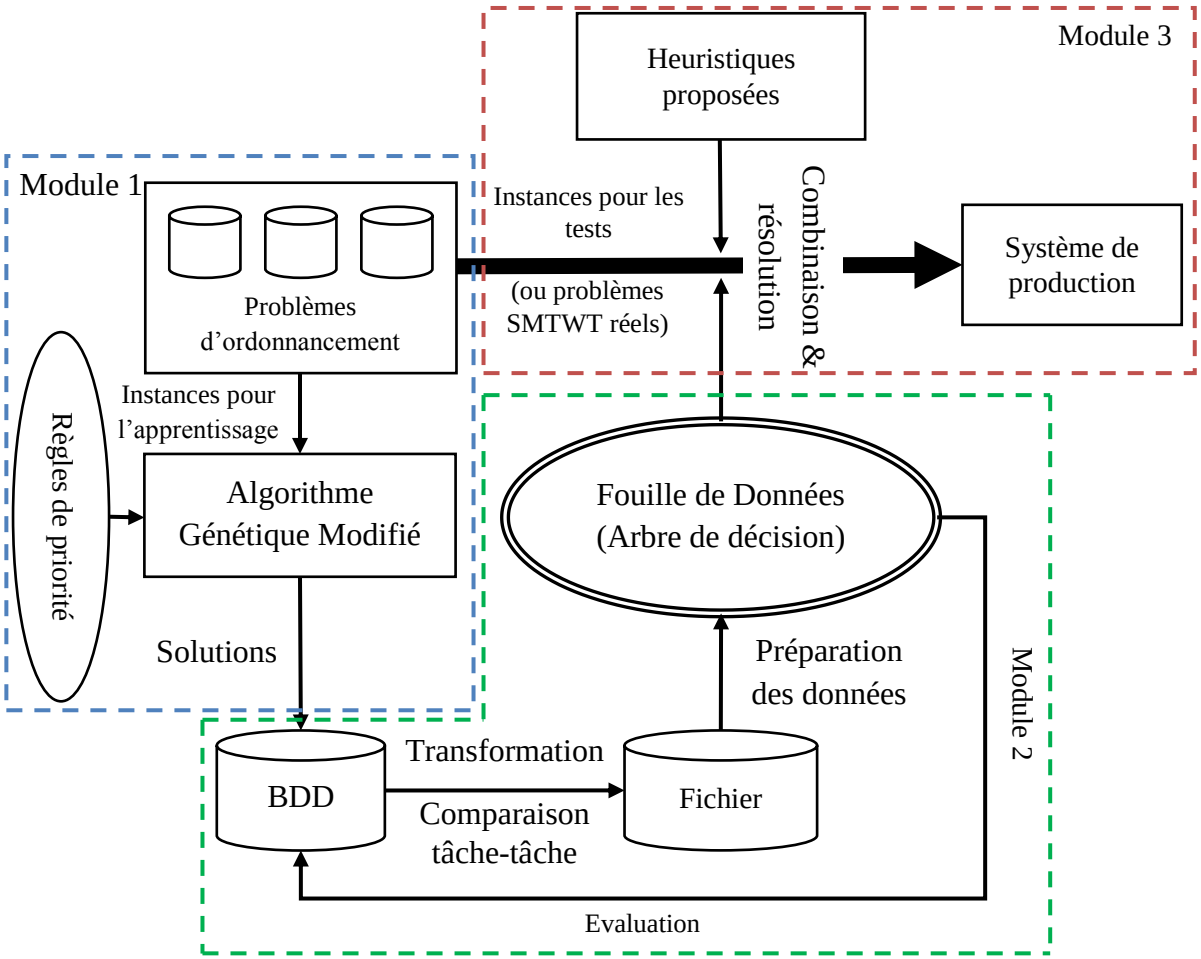

Figure 6.1. Approche proposée

Enfin, durant le dernier module (cf. Section 6.5.4) les nouveaux problèmes SMTWT sont résolus en combinant l'arbre de décision obtenu (connaissances) et les trois heuristiques proposées. Cette fusion permet de résoudre le problème d'ordonnancement d'une manière similaire aux règles de priorité et qui a pour but de minimiser la somme des retards pondérés. Contrairement aux autres règles de priorité où un seul attribut de tâche et pris en considération (EDD par exemple) ou deux attributs (WSPT par exemple) afin de déterminer le séquencement des tâches, l'arbre de décision permet de considérer plus d'attributs à la fois afin d'obtenir de meilleures solutions.

6.5.1. Single Machine total weighted tardiness

Dans cette contribution et à fin d'extraire de la connaissance à partir de données issues de l'algorithme génétique, les problème SMTWT avec leurs meilleures solutions connues de l'OR-Library [141] sont examinés. Beasley [141] donne la procédure utilisée pour générer ces problèmes benchmark de taille $n = 40$, $n = 50$ et $n = 100$, où n est le nombre de tâches et où chacune des catégories (taille du problème) est constituée de 125 instances. Chaque tâche arrive au temps zéro et possède un temps d'exécution pt (*processing time*) oscillant entre 1 et 100, une date d'échéance d (*due date*), et un poids w (*weight*) variant entre 1 et 10. Ces deux attributs sont générés suivant une distribution uniforme. En ce qui concerne la date d'échéance,

elle est aussi générée aléatoirement suivant une distribution uniforme $[P(1-TF-RDD/2), P(1-TF+RDD/2)]$ où RDD désigne « *Range of Due Dates* » $RDD = \{0.2, 0.4, 0.6, 0.8, 1.0\}$ et TF est « *Tardiness Factor* » $TF = \{0.2, 0.4, 0.6, 0.8, 1.0\}$. Lors de la génération de ces problèmes par Beasley [141] pour chaque combinaison de valeurs *RDD*/*TF*, c'est-à-dire 25 combinaisons possibles, cinq instances sont générées donnant ainsi un ensemble de 125 instances pour chaque taille de problèmes. Lors des expérimentations, les solutions de l'approche proposée sont comparées avec les meilleures TWT connues données dans l'OR-Library, ainsi qu'avec l'AGM, et quelques règles de priorité de la littérature.

6.5.2. Algorithme génétique modifié

Afin de mettre en place un algorithme génétique pour résoudre le problème SMTWT, les solutions doivent être encodées d'une manière adéquate, permettant la représentativité de toutes les combinaisons possibles et la possibilité d'atteindre l'ensemble des solutions du problème. Cette représentation doit aussi permettre l'utilisation des procédures de croisement et de mutation. Dans cette approche, chaque population P_j est composée de P chromosomes et où j est le numéro de la génération actuelle. Pour le problème d'ordonnancement à Machine Unique, une simple représentation peut être utilisée, comme expliqué par Armentano & Mazzini [39] ou Sevaux & Sörensen [45]. Chaque solution du problème est présentée sous la forme d'un vecteur à n éléments. Chaque élément de ce vecteur donné par un entier représente une tâche, et l'ensemble du tableau représente la séquence (l'ordre d'exécution des tâches). Par exemple, la séquence d'un problème à Machine Unique composé de cinq tâches peut être codé comme suit : $\{4, 2, 0, 1, 3\}$.

Afin d'améliorer l'efficacité de l'algorithme génétique et parce que les solutions obtenues doivent être d'une bonne qualité, minimisant au mieux le TWT, nous proposons d'inclure quelques règles de priorité lors de la génération de la population initiale. Selon Liu et al. [46], l'intégration de règles à l'algorithme génétique, au lieu de le faire reposer uniquement sur des solutions générées aléatoirement, permet d'atteindre de meilleures solutions. Les règles de priorité utilisées dans cette approche sont connues dans la littérature pour leur efficacité pour des problèmes avec des objectifs liés aux retards des tâches comme « *Earliest Due Date* », « *Weighted Shortest Processing Time* », ou « *Mixed Dispatching Rule* ». Les règles considérées sont :

- Shortest Processing Time (*SPT*),
- Longest Processing Time (*LPT*),
- Weighted Shortest Processing Time (*WSPT*),
- Earliest Due Date (*EDD*),
- Critical Ratio (*CR*),
- Weighted Modified Due Date (*WMDD*) [163],
- Mixed Dispatching Rule (*MDR*) [61].

Nous proposons aussi d'autres règles de priorité malgré leurs faibles performances. Le but de ces règles et de diversifier la population initiale et d'éviter la convergence rapide de l'algorithme génétique. Les règles proposées sont :

- Date d'échéance moins temps d'exécution « *Due Date Minus Processing Time* » $(d_i - pt_i)$, avec une priorité donnée à la tâche avec la plus petite valeur,
- Poids le plus faible « *Lowest Weight* » : la tâche possédant le poids w_i le plus petit est lancée en premier,
- Poids le plus grand « *Highest Weight* » : la tâche possédant le poids w_i le plus grand est lancée en premier.

Quant aux autres chromosomes de la population initiale, ils sont générés aléatoirement. Il est utile de noter que les chromosomes redondants ne sont pas éliminés. Une fois la génération terminée, une sélection par tournoi est utilisée afin d'identifier les chromosomes pour le croisement et/ou mutation. La procédure de croisement considérée et présentée dans la Figure 6.2, suggérée par Sevaux & Sörensen [45], est une procédure à un point. Elle consiste à définir aléatoirement un point de coupure, et les gènes se trouvant à gauche du point de coupure du premier parent sont copiés vers le premier fils. Ensuite, les gènes manquants de ce fils sont complétés en utilisant le second parent, où chaque gène figurant sur le parent 2 et ne figurant pas sur fils 1 est copié en respectant l'ordre d'apparition sur le parent 2. La même procédure est répétée pour créer le second chromosome fils en inversant les rôles des deux parents.

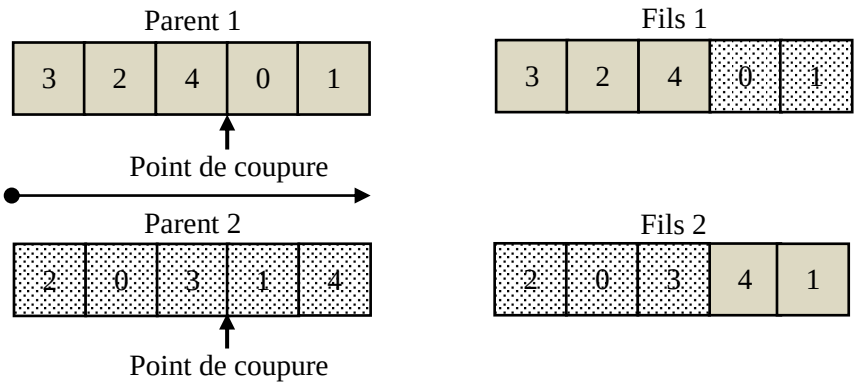

Figure 6.2. Procédure de croisement

La mutation considérée (voir Figure 6.3) consiste à échanger les positions de deux gènes (tâches) sélectionnés aléatoirement, connue sous le terme de mutation par échange. Son rôle est de maintenir la diversité de la population à chaque génération et par conséquent réduire les chances d'atteindre un local optimal.

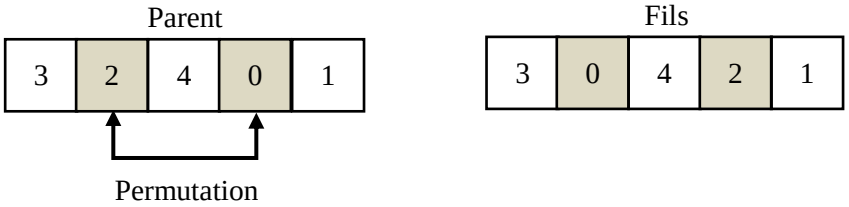

Figure 6.3. Procédure de mutation

Une fois qu'un nouveau chromosome est créé, que ça soit lors de la génération de la population initiale, en utilisant le croisement, ou la mutation, il est évalué et une valeur lui est affectée

correspondante à la somme des retards pondérés des tâches donnée par l'Équation 6.2 ci-dessous. Lors de l'évolution (croisement et mutation), si les nouveaux chromosomes fils sont meilleurs que leurs parents, alors ils sont insérés dans la population et les parents sont éliminés.

$$\sum_{i=0}^{n-1} w_i \times \max\{C_i - d_i; 0\} \tag{6.2}$$

L'exécution d'un algorithme génétique est suspendue si un certain nombre de générations est atteint ou en fonction de la meilleure solution trouvée. L'AGM proposé repose sur ces deux critères d'arrêt. Le premier est le nombre maximal d'itérations appelé « $MaxIter$ », si l'AGM atteint cette limite, il est automatiquement arrêté. Quant au second critère, l'AGM est stoppé si une solution optimale est trouvée. Les solutions optimales, meilleures valeurs TWT, sont connues pour chaque problème et sont disponibles dans l'OR-Library [141].

6.5.3. Préparation des données et extraction de connaissances

Pour passer à l'étape d'extraction de connaissances, il est d'abord nécessaire de préparer et formater les données. Cette procédure est d'une importance capitale si l'on veut obtenir des connaissances utiles. Dans cette contribution et en s'inspirant de la littérature [24], [56], [73], [82], [177], les solutions du SMTWT obtenues par l'AGM sont transformées en un fichier de données (tableau) qui peut être utilisé par la suite comme entrée pour la méthode de fouille de données, à savoir, les arbres de décision (voir module 2 dans la Figure 6.1). Ce fichier sous forme de tableau est composé de lignes et de colonnes. Chacune des colonnes représente l'un des attributs des tâche et une ligne contient une information partielle indépendantes des autres linges [180]. La création de ce fichier est illustrée dans l'exemple suivant. On suppose un problème SMTWT composé de quatre tâches (voir Tableau 6.1), où chaque tâche possède un temps d'exécution, un poids, et une date d'échéance.

Tableau 6.1. Problème SMTWT à quatre tâches

ID tâche	Temps d'exécution pt	Poids w	Date d'échéance d
0	5	2	10
1	4	1	12
2	7	3	15
3	8	4	20

La génération du fichier de données et directement dépendante d'une séquence faisable. Cette séquence peut être obtenu en utilisant n'importe quelle méthode de résolution y compris avec les règles de priorité et les algorithmes génétiques. Pour cet exemple illustratif, on suppose que la séquence obtenue par l'AGM est |3|0|2|1|, où la tâche 3 doit être lancée en premier, suivie par les tâches 0, 2, et 1, dans cet ordre. Une fois cette solution acquise par l'AGM, un tableau (voir Tableau 6.2) de comparaison tâche-tâche est construit. Chaque tâche représentée par ses attributs est comparée aux autres tâches en se basant sur la séquence de l'AGM et un attribut classe est défini, appelé « *tâche1première* » (en anglais « *job1first* »). Cette classe peut prendre deux valeurs « *oui* » si la tâche 1 doit être lancée avec la tâche 2, « *non* » sinon.

Tableau 6.2. Tableau de comparaison tâche-tâche

Tâche1	pt_1	w_1	d_1	Tâche2	pt_2	w_2	d_2	Tâche1 première
0	5	2	10	1	4	1	12	oui
0	5	2	10	2	7	3	15	oui
0	5	2	10	3	8	4	20	non
1	4	1	12	2	7	3	15	non
1	4	1	12	3	8	4	20	non
2	7	3	15	3	8	4	20	non

Une fois que le fichier est construit, une opération de prétraitement est alors conduite. Le prétraitement peut inclure la construction et la sélection d'attributs, l'agrégation, ou la suppression de données redondantes. Cette étape, comme soulevé plusieurs fois auparavant (cf. Section 3.2.2), est cruciale durant le processus d'extraction de données. En général, d'énormes efforts doivent être déployés en vue de préparer les données avant de pouvoir utiliser la fouille de données et d'extraire de la connaissance utile. Durant ces expérimentations, seule une suppression des doublons est menée car les données contenues dans ce fichier sont correctes et complètes et ne nécessitent aucune autre opération de prétraitement.

Les attributs pris en considérations dans ces expérimentations, et comme illustré dans le Tableau 6.1 et le Tableau 6.2, sont le temps d'exécution, le poids, et la date d'échéance. Ces attributs sont indépendants et ne peuvent être agrégés. En plus, vu que ces attributs sont disponibles pour chaque tâche et pour chaque problème d'ordonnancement, alors le fichier obtenu ne contient aucune valeur manquante ou incorrecte. Cependant, comme plusieurs instances de problèmes SMTWT sont groupées en un seul fichier pour l'apprentissage, il est possible d'avoir des données redondantes qu'il faut élaguer. Les identifiants des tâches (colonnes « *Tâche* 1 » et « *Tâche* 2 ») affichés dans le Tableau 6.2 ne sont pas pris en considération lors de l'apprentissage et sont affichés uniquement pour faciliter la compréhension de cet exemple.

Lorsque le prétraitement est accompli, un algorithme d'apprentissage machine est utilisé pour générer un modèle prédictif. Plusieurs algorithmes pour l'apprentissage peuvent être considérés comme les réseaux de neurones, les machines à vecteurs de support, ou les arbres de décision. Ces derniers, selon Li & Olafsson [180], sont facilement compréhensibles et peuvent être traités, lors de l'ordonnancement, par un opérateur humain contrairement aux réseaux de neurones considérés comme une boite noire. Dans cette approche, des arbres de décision sont utilisées, car suggérées par Li & Olafsson [180], mais aussi en raison de leur simplicité et leur efficacité. De plus, ces algorithmes ont été largement sollicité dans la littérature pour l'extraction de connaissance notamment dans les travaux de recherche [24], [56], [69], [180] (cf. Section 3.4). Le module d'apprentissage repose sur l'algorithme J48 pour l'extraction de connaissance à partir du fichier de données sous la forme « *si-alors* » (ou sous une forme visuelle arborescente). J48 est une implémentation Java de l'algorithme C4.5 disponible dans le logiciel de fouille de données Weka développé par Hall et al. [178]. L'arbre de décision obtenu est comme suit (voir aussi Figure 6.4) :

si $d_1 \leq 10$ **et** $pt_2 \leq 7$ **alors** la Tâche 1 est lancée en premier (**oui**)
si $d_1 \leq 10$ **et** $pt_2 > 7$ **alors** la Tâche 2 est lancée en premier (**non**)
si $d_1 > 10$ **alors** la Tâche 2 est lancée en premier (**non**)

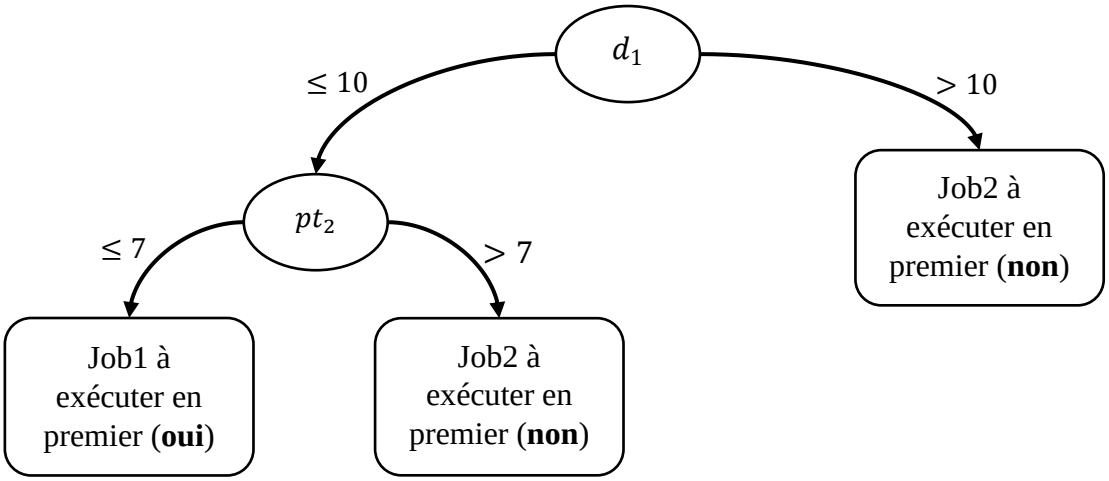

Figure 6.4. Arbre de décision

L'arbre de décision obtenu est ensuite testé et évalué en vue d'améliorer son fonctionnement et son efficacité. En ce sens, plusieurs mesures peuvent être calculées, comme le taux d'apprentissage, la précision, ou le rappel. Si les résultats ne sont pas jugés satisfaisants, alors la procédure d'apprentissage et répétée en changeant les paramètres de l'algorithme jusqu'à obtenir les performances désirées. Enfin, ces règles « *si-alors* » peuvent être utilisées afin de résoudre de nouveaux problèmes d'ordonnancement [82].

6.5.4. Connaissances et heuristiques pour l'ordonnancement

Notre contribution en comparaison à d'autres approches basées sur la fouille de données pour l'extraction de connaissances peut se résumer en deux points. Le premier est lié à la plus grande taille des problèmes résolus et qui, de ce fait, sont plus difficiles que ceux considérés dans la littérature. Par exemple Koonce & Tsai [73] et Shahzad & Mebarki [24], [82], [91] s'intéressent à des problèmes Job Shop composés de 6 tâches et 6 machines. Alors que Aissani et al. [177] testent leur approche sur un problème Job Shop 4×2, et Balasundaram et al. [69] un Flow Shop composé de 5 tâches et de deux 2 machines. Par ailleurs, dans cette approche les problèmes à Machine Unique considérés sont composés de 40, 50, et 100 tâches. Ces problèmes sont plus complexes que le Job Shop 6×6 ou le Flow Shop 5×2 en raison du grand nombre de tâches et de la nature NP-Difficile du problème. Par exemple, pour le problème Job Shop 6×6 le pire des scénarios serait d'avoir les 6 tâches en même temps, en concurrence sur une des machines. Mais dans le cas du SMTWT, une tâche à lancer en premier doit être choisi par un ensemble de 40, 50, ou 100 tâches, toutes nécessitant d'être exécutées sur cette machine.

En ce qui concerne le second point, il est en relation avec la manière d'utilisation des règles sous forme « *si-alors* ». Shahzad & Mebarki [24] explique qu'au moment où une machine est disponible et qu'il existe un certain nombre de tâches en attente et qui nécessitent cette ressource, une comparaison doit être effectuée afin de déterminer celle à faire passer en premier. Par exemple, trois tâches J_1, J_2, et J_3 sont en concurrence pour obtenir la machine, donc il est

nécessaire de comparer ces tâches en utilisant l'arbre de décision (règles « *si-alors* »). L'arbre de décision donne la réponse « *oui* » si la tâche J_i doit être lancée avant la tâche J_j ou « *non* » sinon. Ceci, selon Shahzad & Mebarki [24] permet de construire une séquence faisable. Par exemple, si les couples J_1 vs J_2, J_1 vs J_3, et J_2 vs J_3 renvoient, à l'aide de l'arbre de décision, oui, oui, et non, respectivement, alors la séquence obtenue serait $|J_1|J_3|J_2|$. Cependant, avec plus de tâches la comparaison pourrait éventuellement mener vers des boucles. Par exemple, si après classification on obtient les réponses oui, non, et oui. Ceci veut dire que la tâche J_1 doit être exécutée avec la tâche J_2, J_3 avant J_1, et J_2 avant J_3. Si on prend en considération uniquement la première et la dernière comparaison, alors la séquence obtenue par déduction est $|J_1|J_2|J_3|$ et qui est faisable. Mais si la seconde comparaison est aussi prise en considération, alors la séquence devient non-faisable, en un mot, la séquence devient une boucle fermée, parce que la tâche J_3 doit être lancée avec J_1. Il est aussi possible de rencontrer d'autres erreurs de classification, comme par exemple lors de la comparaison de J_i vs J_j et J_j vs J_i, l'arbre de décision peut renvoyer oui, oui ou non, non.

Afin de résoudre ce type de situations causées par des erreurs de classification de l'arbre de décision, trois heuristiques sont proposées. Leur but est d'interpréter les résultats de l'arbre de décision et de corriger si nécessaire les erreurs survenues tout en tentant de minimiser la somme des retards pondérés. La première heuristique compare les tâches un à un en utilisant l'arbre de décision et échange leurs positions si l'arbre de décision renvoie la réponse « *non* » comme résultat de classification. Cette procédure est répétée continuellement jusqu'à ce qu'un nombre maximal d'itérations soit atteint, tout en gardant en mémoire la meilleure solution trouvée.

Heuristique Proposée 1 : Tri Rapide

Entrée : MaxIterations
Sortie : Séquence

```
    /*** Initialiser la séquence S avec les tâches triées de 0 to n ***/
 1: S ← ListeDesTâches();
    /*** Sauvegarder la solution initiale en tant que meilleure séquence S* ***/
 2: S* ← S
 3: TWT* ← TWT(S)
    /*** Comparer et trier les tâches en utilisant l'arbre de décision ***/
 4: tant que S change et MaxIterations not encore atteint faire
 5:         pour chaque tâche i ∈ S faire
 6:                 pour chaque tâche j ∈ S − {i} faire
 7:                         si !ClasserAvecADD(i, j) alors
 8:                                 Echanger(i, j)
 9:                         fin si
10:                 fin pour
11:         fin pour
            /*** Sauvegarder la meilleure solution ***/
12:         si TWT(S) < TWT* alors
13:                 S* ← S
14:                 TWT* ← TWT(S)
15:         fin si
16: fin tant que
17: renvoyer S
```

La deuxième heuristique compare aussi les tâche un à un et met à jour les valeurs d'un vecteur. La valeur du vecteur à une position i correspond au nombre de « *oui* » obtenus par cette la tâche i. A chaque fois qu'une comparaison est effectuée entre la tâche i et les autres tâche j où $i,\ j = 0, \dots, n-1$ et $j \neq i$ et la réponse obtenue est « *oui* » alors la valeur du vecteur à cette position est incrémentée de 1. Cette opération est répétée pour l'ensemble des tâches, et après tri, la tâche qui obtient le meilleur score (le plus grand nombre de « *oui* ») est lancée en premier.

Heuristique Proposée 2 : Tri par nombre de « oui » obtenus

Sortie : Séquence

```
      /*** Initialiser la séquence S avec les tâches triées de 0 to n ***/
 1 :  S ← ListeDesTâches ();
      /*** Comparer les tâches et mettre à jour les valeurs du Vecteur ***/
 2 :  pour chaque tâche i ∈ S faire
 3 :          pour chaque tâche j ∈ S faire
 4 :                  si i ≠ j alors
 5 :                          if ClasserAvecADD (i, j) then
 6 :                                  Vecteur (i) ← Vecteur (i) + 1
 7 :                          fin si
 8 :                  fin si
 9 :          fin pour
10 :  fin pour
      /** Tri des tâches de S en utilisant les valeurs du Vecteur dans un ordre décroissant **/
11 :  Trier(S, Vecteur, ↓)
12 :  renvoyer S
```

Lors de la résolution d'un problème d'ordonnancement en utilisant les règles de priorité, il est nécessaire d'intégrer un moyen de départager les tâches en cas de parité entre deux ou plusieurs d'entre-elles (en anglais « *tie-breaking* »). Par exemple, deux tâches avec le même temps d'exécution auront la même priorité d'exécution en utilisant la règle de priorité SPT. Comme solution, il est possible de sélectionner l'une des deux aléatoirement ou alors de considérer une seconde règle de priorité qui se base sur un autre attribut des tâches et qui pourrait les départager comme la règle EDD. Dans cette optique, dans la troisième heuristique lorsque l'arbre de décision génère une mauvaise décision, c'est-à-dire, renvoyer « *oui* » pour les couples de tâches (i, j) et (j, i), ou « *non* » pour le deux, alors la règle de priorité WSPT et considérée pour les départager et la tâches sont triés en conséquence.

Durant les expérimentations, les trois heuristiques proposées sont combinées avec l'arbre de décision afin d'interpréter et corriger les décisions de l'arbre, et de construire une séquence faisable. La valeur la plus petite de la somme des retards pondérés obtenues par l'une des heuristiques est utilisée par la suite lors de l'étude comparative.

Heuristique Proposée 3 : Trier avec WSPT pour le tie-breaking

Sortie : Séquence

/*** Initialiser la séquence S avec les tâches triées de 0 to n ***/

1 : $S \leftarrow$ ListeDesTâches ();

/*** Comparer et trier les tâches en utilisant l'ADD et WSPT pour le tie-breaking ***/

2 : **pour** chaque tâche $i \in S$ **faire**

3 : **pour** chaque tâche $j \in S$ où $j \leftarrow i + 1$ **faire**

/* En cas d'erreur de classification utiliser WSPT pour départager les tâche i et j */

4 : **si** ClasserAvecADD (i, j) **et** ClasserAvecADD (j, i) **alors**

5 : **si** $WSPT_i > WSPT_j$ **alors**

6 : Echanger(i, j)

7 : **fin si**

8 : **alors**

9 : **si !**ClasserAvecADD (i, j) **alors**

10 : Echanger (i, j)

11 : **fin si**

12 : **fin si**

13 : **fin pour**

14 : **fin pour**

15 : **renvoyer** S

Tableau 6.3. Notations utilisées dans les heuristiques proposées

Notation	Description
S	Séquence des tâches triées initialement de 0 à $n - 1$
ListeDesTâches ()	Procédure utilisée pour initialiser la séquence S
S^*	Meilleure séquence trouvée
TWT^*	Meilleure valeur somme des retards pondérés (total weighted tardiness) trouvée (S^*)
$TWT(S)$	Procédure pour calculer la somme des retards pondérés (total weighted tardiness) pour une séquence S
MaxIterations	Nombre maximum d'itération défini par l'utilisateur
ClasserAvecADD (i, j)	Procédure qui utilise l'arbre de décision pour classer l'instance composée des tâches i et j. Renvoie *vrai* (oui) si i doit être lancée avant j. *Faux* (non) sinon
Echanger(i, j)	Echange les positions des tâches i et j dans la séquence S
Vecteur (i)	Vecteur contant le nombre de réponses *vrai* (oui) renvoyées lors des comparaisons de tâches de la tâche i
Trier(S, Array, ↓)	Trie la séquence S en prendant en consideration le nombre de of *vrai* (oui) dans Vecteur pour chaque tâche et dans un ordre décroissant
$WSPT_i$	Le temps d'exécution pondérés (weighted shortest processing time) de la tâche i donné par w_i/p_{ij}

Nous illustrons dans l'exemple suivant l'utilisation des connaissances extraites et des trois heuristiques proposées pour la résolution du problème SMTWT présenté dans le Tableau 6.1. On suppose qu'après la résolution d'un certain nombre de problèmes en utilisant l'AGM et

après la création d'un arbre de décision, les tâches sont comparées un à un (classification) et le Tableau 6.4 est obtenu.

Tableau 6.4. Résultats de la classification

Tâche	0	1	2	3
0	-	Non	Oui	Oui
1	Non	-	Non	Oui
2	Non	Non	-	Oui
3	Non	Non	Non	-

Pour avoir une idée plus claire de l'incidence des trois heuristiques proposées, on les utilise, tour à tour, sur le Tableau 6.4, et on observe les séquences obtenues. D'abord, en utilisant le tri rapide (heuristiques proposée 1), et on fixe à 5 le nombre maximum d'itérations (*MaxIterations*). Cette procédure répète le tri des tâches à chaque fois que la séquence change et que « *MaxIterations* » n'est pas encore atteint. A chaque fois qu'une nouvelle séquence est obtenue, elle est évaluée et considérée meilleure (S^*) si elle obtient le TWT le plus petit (voir Tableau 6.5).

Tableau 6.5. Séquence obtenue par l'heuristiques proposée 1

Itération	Séquence	TWT
0 (initiale)	|0|1|2|3|	19*
1	|1|3|2|0|	40
2	|0|1|3|2|	27
3	|1|2|0|3|	28
4	|0|1|3|2|	27
5	|1|2|0|3|	28
Meilleure séquence	**|0|1|2|3|**	**19**

La seconde heuristique proposée (tri en fonction du nombre de réponses « *oui* » obtenues), donne la séquence |1|0|2|3| et a un score (somme des retards pondérés) de 19. Quant à la troisième heuristique, elle donne la séquence |3|2|0|1| avec un score de 32. Enfin, l'approche proposée sélectionne la ou les meilleures solutions obtenues à savoir |1|0|2|3| ou |1|0|2|3| les deux avec un $TWT = 19$.

6.6. Résultats

Les expérimentations sont menées sur des ateliers SMTWT composés de 40, 50, et 100 tâches. Les résultats de l'AGM sont montrés et discutés en premier, suivis de l'utilisation de l'arbre de décision J48 avec les heuristiques proposées. Les expérimentations sont codées en Java est menées sur un ordinateur avec un processeur Intel Core i7 2.4 GHz et 8 Go de mémoire vive avec un seul thread.

Tableau 6.6. Paramètres utilisés

Paramètre	Valeur
Taille population (P)	200
Chromosomes sélectionnés ($P/2$)	100
Taux de croisement	90%
Taux de mutation	10%
Nombre maximal d'itérations de l'AGM ($MaxIter$)	10.000 pour les problèmes à 40 et 50 tâches 20.000 pour les problèmes à 100 tâches
Nombre maximum d'itération de l'heuristique proposées 1 (MaxIterations)	100

6.6.1. Algorithme génétique modifié

Pour chaque problème d'ordonnancement à Machine Unique, la meilleure valeur de la somme des retards pondérés est connue ; par conséquent, l'écart entre l'AGM et les meilleurs résultats est calculé en utilisant l'Équation 6.3, où $AGM(i)$ est le plus petit TWT obtenu par l'AGM pour un problème i est $Meilleur(i)$ est la meilleure valeur TWT connue (disponible dans l'OR-Library) pour ce même problème i.

$$\left(MGA(i) - Meilleur(i)\right) \Big/ Meilleur(i) \tag{6.3}$$

Il est important de noter que pour certains problèmes la meilleure valeur du TWT connue est égale à zéro. En conséquence, si l'AGM trouve une solution avec un objectif égal à zéro, alors l'écart est automatiquement fixé à zéro. Sinon, si l'AGM donne un TWT supérieur à zéro alors que la meilleure valeur est égale à zéro (division par zéro), le problème est ignoré.

Dans cette section, les résultats de tous les problèmes SMTWT sont exposés, c'est-à-dire, 375 problèmes : 125 problèmes à 40 tâches, 125 problèmes à 50 tâches, et 125 problèmes à 100 tâches. Le but est de montrer l'efficacité de l'intégration de règles de priorité lors de la génération de la population initiale comme discuté auparavant. Pour rappel, le nombre maximal d'itérations de l'AGM pour les problèmes à 40 et 50 tâches est fixé à 10.000 alors que pour les problèmes à 100 tâches ce nombre est fixé à 20.000 itérations. Si l'AGM ne retrouve pas la même valeur TWT que celle disponibles dans l'OR-Library (meilleure valeur) avant d'atteindre le nombre maximal d'itérations, alors son exécution est arrêtée et la meilleure solution obtenue jusque-là est renvoyée puis sauvegardée pour l'apprentissage.

Parmi les 125 problèmes à 40 tâches, la meilleure TWT (OR-Library) est trouvée lors de la première itération (génération de la population initiale) pour 39 problèmes. Pour 40 autres problèmes, la meilleure solution n'est pas trouvée même après 10.000 itérations, alors que pour les 46 problèmes restants la meilleure solution est identifiée avant que l'AGM ne soit stoppé (voir Figure 6.5).

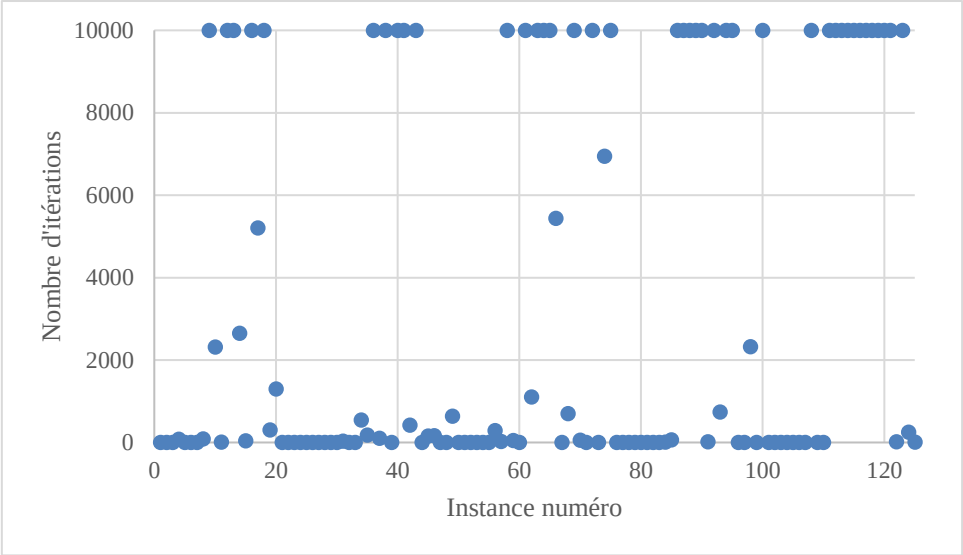

Figure 6.5. Nombre d'itérations de l'AGM pour les problèmes à 40 tâches

L'AGM retrouve les meilleures solutions pour 85 sur 125 problèmes soit un taux de 68,00% avec un écart moyen avec les meilleures valeurs de l'OR-Library de 0,17%. En raison du faible écart entre les résultats de l'AGM et ceux disponibles dans l'OR-Library, l'écart est remplacé par la différence entre ces deux valeurs en utilisant l'Équation 6.4 et cette différence est montrée dans la Figure 6.6.

$$AGM(i) - Meilleur(i) \tag{6.4}$$

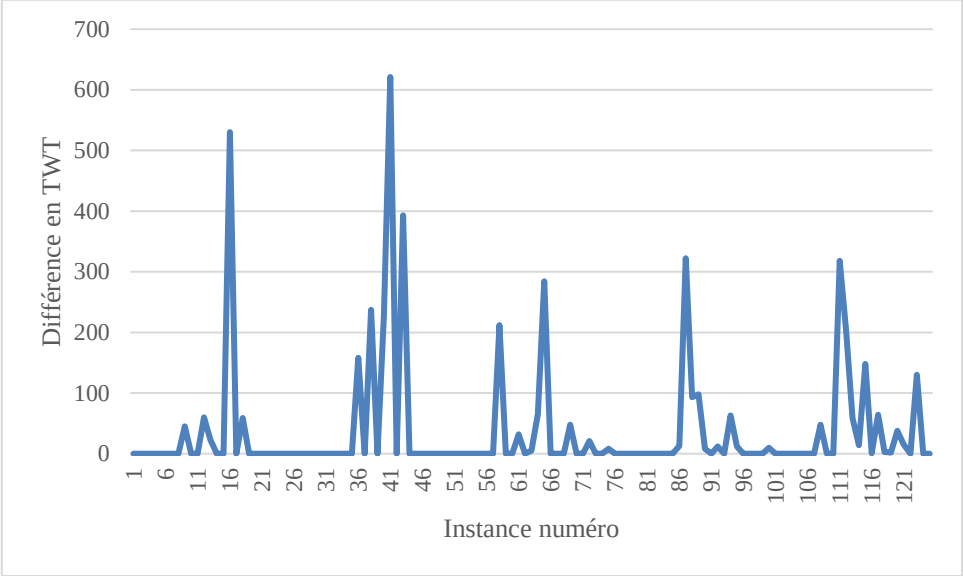

Figure 6.6. Différence en termes de TWT pour les problèmes à 40 tâches

En ce qui concerne les problèmes à 50 tâches où le nombre maximal d'itérations est aussi fixé à 10.000, l'AGM retrouve 44,80% des solutions optimales, soit 56 problèmes sur les 125. En détails, pour 29 problèmes la solution optimale est trouvée lors de la génération de la population initiale, alors pour 69 problèmes elle n'est pas trouvée même après avoir atteint les 10.000 itérations. Cependant, pour les 27 problèmes restants, une solution optimale est trouvée avant

d'atteindre le plafond d'itérations (voir Figure 6.7). Pour ces problèmes, l'AGM a un écart moyen de 0,63% en comparaisons avec les valeurs de l'OR-Library. Dans la Figure 6.8 sont présentés les écarts en TWT entre l'AGM et les meilleurs résultats connus pour chaque problème.

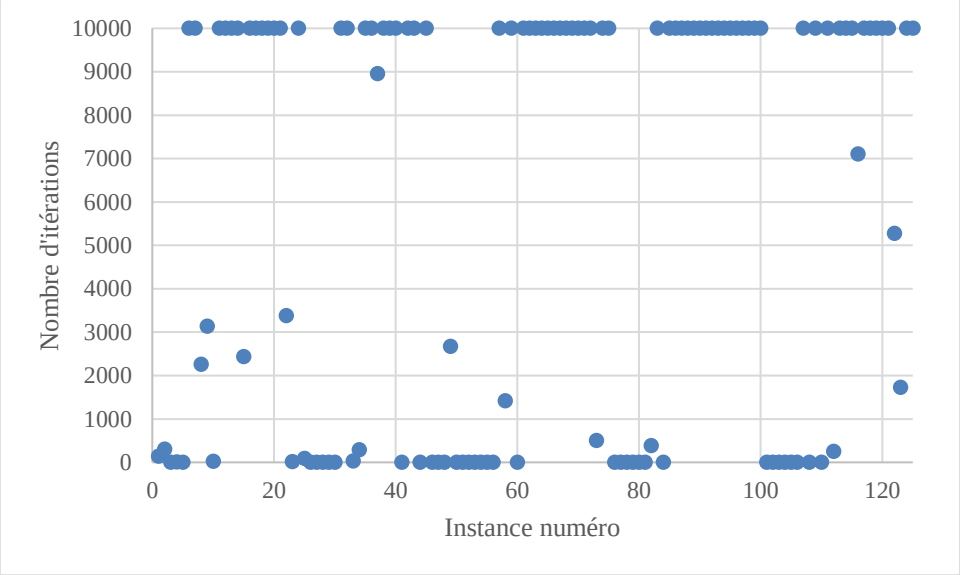

Figure 6.7. Nombre d'itérations de l'AGM pour les problèmes à 50 tâches

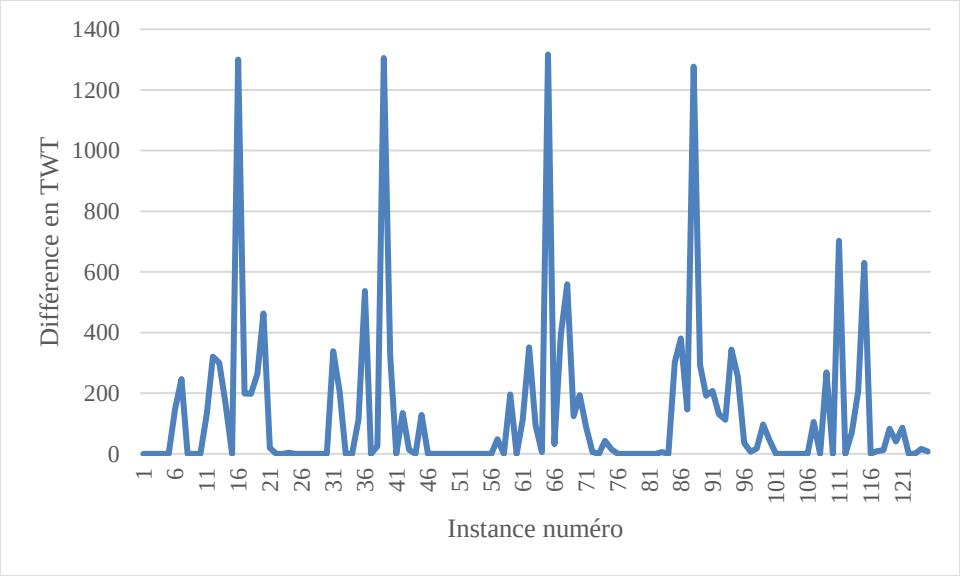

Figure 6.8. Différence en termes de TWT pour les problèmes à 50 tâches

Quant aux problèmes composés de 100 tâches, les solutions optimales sont trouvées lors de la première génération pour 25 problèmes alors qu'elles ne sont pas trouvées pour 90 autres problèmes. Pour les 10 problèmes restants, une solution optimale est trouvée avant d'atteindre le nombre maximal d'itérations qui est de 20.000 (voir Figure 6.9). L'AGM a un taux de précision (problèmes pour lesquels une solution optimale est trouvée sur le nombre de problèmes) de l'ordre de 28,00% soit 35 problèmes sur 125 avec un écart moyen de 2,00%.

Dans la Figure 6.10, nous exposons l'écart en termes de TWT entre l'AGM et les résultats de l'OR-Library.

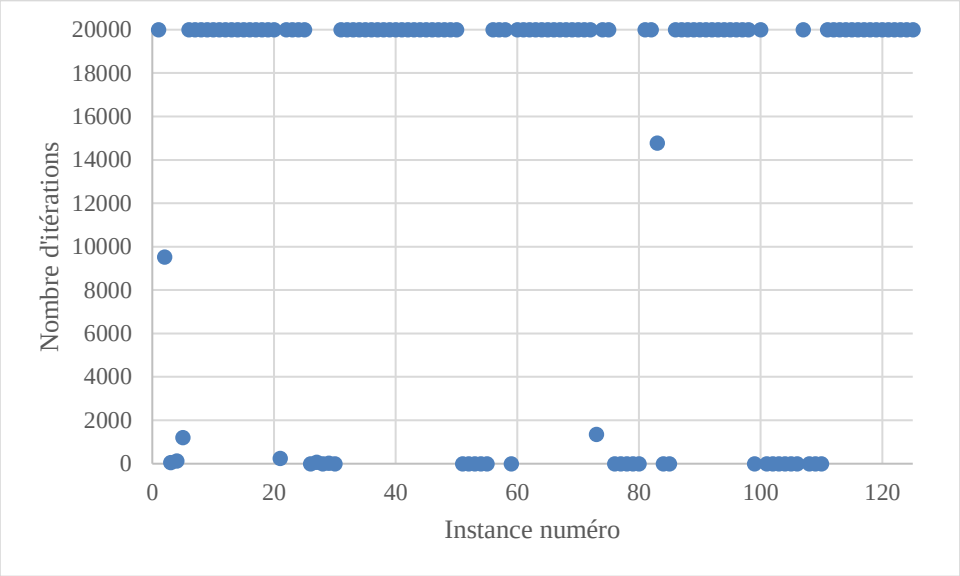

Figure 6.9. Nombre d'itérations de l'AGM pour les problèmes à 100 tâches

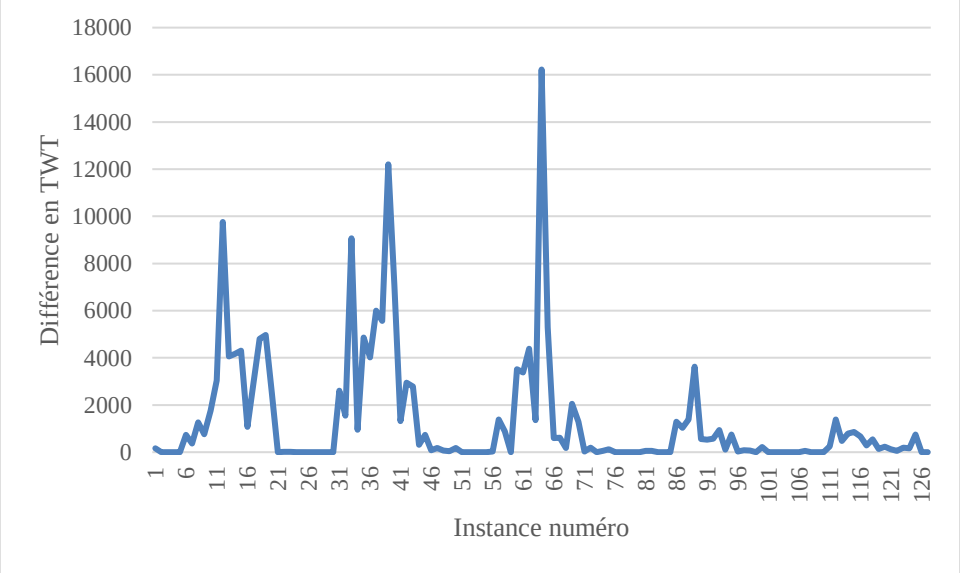

Figure 6.10. Différence en termes de TWT pour les problèmes à 100 tâches

6.6.2. Fouille de données

Une fois que les solutions pour chaque problème SMTWT aient été trouvées en utilisant l'AGM, elles sont transformées en fichier comme expliqué dans la Section 6.5.3 (tableau de comparaison tâche-tâche). Trois ensembles de données sont ainsi créés, un pour chaque taille de problèmes (40, 50, et 100). Chacun de ces ensembles est composé de deux sous-ensembles de données, l'un pour l'apprentissage et l'autre pour les tests. La génération des problèmes SMTWT (cf. Section 6.5.1) se fait en fonction des pairs de valeurs RDD et TF afin de faire varier la complexité des problèmes. Pour chacune des 25 pairs, cinq instances sont générées. Durant ces expérimentations impliquant la fouille de données, quatre des cinq problèmes sont

réservés pour l'apprentissage et la création de l'arbre de décision et le dernier pour les tests et la validation. Ainsi, pour chaque type de problèmes SMTWT, 100 problèmes sont dédiés à l'apprentissage alors les 25 restants seront résolus à l'aide de l'arbre de décision et des heuristiques proposées (cf. Section 6.5.4). Les solutions des problèmes réservés pour l'apprentissage sont transformées en un seul fichier de données, sur lequel l'algorithme J48 va se baser pour construire l'arbre de décision, après suppression des données redondantes. Quant aux 25 problèmes restants, ils sont résolus en utilisant la combinaison arbres de décision et heuristiques proposées. La valeur minimale de la fonction objectif donnée par l'une des trois heuristiques est utilisée pour la comparaison avec l'AGM et les meilleurs résultats disponibles dans l'OR-Library. Après plusieurs expérimentations, les taux de classification des arbres de décision obtenus sont de 98,70% pour les problèmes à 40 et 50 tâches et de 99,32% pour les problèmes à 100 tâches.

Les expérimentations montrent que les solutions trouvées à l'aide de l'arbre de décision et des heuristiques proposées sont meilleures que celles obtenues par quelques règles de priorité. Une étude comparative est menée sur les résultats obtenus d'un côté en appliquant les règles de priorité lors de la génération de la population initiale, d'un autre côté par l'AGM, et les meilleurs résultats de l'OR-Library. Les règles de priorité considérées sont connues pour leurs bonnes performances pour les problèmes SMTWT et sont aussi utilisées par Kanet & Li [163], Liu et al. [46], ou Yin & Wang [61]. Dans le Tableau 6.7 sont exposés les écarts entre l'ADD combiné avec les heuristiques proposées et les règles de priorité considérées pour l'étude comparative. L'approche proposée (AP), comme démontré par ces résultats, permet de résoudre plus efficacement les problèmes SMTWT que les règles de priorité. Quant à l'AGM il donne de meilleurs résultats que l'ADD mais nécessite beaucoup plus de temps de calcul. Il est utile de noter que les problèmes avec un $TWT = 0$ ne sont pas pris en considération lors de cette phase comparative.

Tableau 6.7. AP vs règles de priorité, AGM et meilleures valeurs connues

Taille prob.	**Ecart moyen (%)**								
	SPT	**LPT**	**WSPT**	**EDD**	**CR**	**MDR**	**WMDD**	**AGM**	**Meilleur**
40 tâches	940.15	4579.0.2	512.43	87.68	1326.37	87.00	461.54	-46.59	-46.88
50 tâches	8406.21	19784.22	4843.09	95.72	10729.08	95.01	49.37	-51.03	-52.18
100 tâches	8297.89	34702.51	4170.01	26.19	15355.50	26.00	4422.22	-115.45	-121.73

Pour les problèmes à 40 tâches, l'approche proposée retrouve une solution optimale pour 17,00% des problèmes. Alors que pour les problèmes à 50 et 100 tâche, ce taux est de 15,00%. Dans le Tableau 6.8 sont affichés le nombre de problèmes où l'approche proposée (*AP*) ou toute autre règle de priorité (SPT, LPT, WSPT, EDD, CR, MDR, ou WMDD) retrouve la meilleure solution. Cette comparaison est menée en changeant les ensembles d'apprentissage et de test 50 fois (itérations), soit 100 problèmes SMTWT testés. L'astérisque désigne le plus grand nombre de meilleures solutions trouvées pour le même type de problèmes (même ligne).

Tableau 6.8. Approche proposée vs règles de priorité

Taille prob.	Nombre de meilleures solutions retrouvées							
	AP	**SPT**	**LPT**	**WSPT**	**EDD**	**CR**	**MDR**	**WMDD**
40 tâches	**85***	0	0	7	23	0	22	1
50 tâches	**85***	0	0	9	19	0	19	1
100 tâches	**80***	0	0	13	22	0	23	0

D'autres expérimentations supplémentaires démontrent aussi la supériorité de l'approche proposée lors de la résolution de problèmes SMTWT contre les heuristiques H1, H2, et H3 proposées par Yoon & Lee [132] avec un écart moyen oscillant entre 1,08 et 8,44%. D'un autre point de vue, l'approche proposée combinant arbre de décision et heuristique peut être utilisée en temps réel pour résoudre des problèmes dynamiques ou faire face aux perturbations, contrairement aux algorithmes génétiques ou autres méta-heuristiques qui nécessitent un temps de calcul beaucoup plus important. Ceci est notamment dû au fait que la partie d'apprentissage qui est assez couteuse en temps de calcul est faite en hors-ligne et par conséquence, n'affecte pas les performances du système de production, alors que l'ordonnancement des tâches est fait en temps-réel. Dans le Tableau 6.9 sont affiché les temps de calculs requis pour la création de l'arbre de décision, ainsi que la résolution des problèmes SMTWT en utilisant les heuristiques proposées et l'ADD. Ces expérimentations sont répétées 50 fois et les moyennes de temps de calcul sont affichées dans le tableau. Notez que la création de l'arbre de décision (colonne ADD) est faite en hors-ligne pour chaque type de problèmes (taille problème).

Tableau 6.9. Temps de calcul

Taille prob.	Temps de calcul en millisecondes										
	ADD	**PH1**	**PH2**	**PH3**	**SPT**	**LPT**	**WSPT**	**EDD**	**CR**	**MDR**	**WMDD**
40 tâches	11221	100.91	1.17	2.63	0.02	0.01	0.01	0.01	0.01	0.04	0.06
50 tâches	25985	163.98	1.87	4.24	0.02	0.01	0.01	0.01	0.01	0.05	0.08
100 tâches	181267	643.44	7.44	15.82	0.03	0.02	0.01	0.02	0.02	0.15	0.26

L'efficacité de l'approche présentée dans cette contribution est liée à la capacité de l'arbre de décision de considérer plusieurs attributs de tâches en même temps même si ces attributs ne sont pas directement liés ou directement comparables, comme le cas du temps d'exécution et des dates d'échéance. L'arbre de décision comme n'importe quelle règle de priorité peut comparer les tâches deux à deux et décider de la meilleure façon de les ordonnancer afin de minimiser la somme des retards pondérés. Cependant, il est incapable de considérer plusieurs tâches à la fois sans pour autant créer de solutions infaisables (revoir exemple dans la Section 6.5.4) et c'est pour cette raison que les trois heuristiques sont proposées afin d'aider l'arbre de décision de à résoudre les problèmes SMTWT comme toute autre règle de priorité. D'un autre côté, l'arbre de décision nécessite d'être implémenté en utilisant des problèmes similaires afin de prendre en considération les caractéristiques de ces problèmes, tel que les fourchettes de valeurs, lors de la résolution de nouveaux problèmes.

6.7. Conclusion

Dans cette contribution, une approche innovante est développée montrant que la fouille de données peut être considérée comme une bonne alternative pour la résolution du problème à Machine Unique avec minimisation de la somme des retards pondérés. La fouille de données peut imiter jusqu'à un certain point les solutions retournées par n'importe quelle méthode de résolution y compris les algorithmes génétiques.

Dans cette approche, un algorithme génétique modifié est proposé pour la résolution du SMTWT. L'AGM combiné avec un certain nombre de règles de priorité lors de la génération de la population initiale est efficace pour résoudre les problèmes SMTWT. Il donne de bons résultats tout en trouvant des solutions optimales pour plusieurs problèmes avec un taux de 46,93% pour les 375 problèmes considérés avec un écart moyen de 0,93% en comparaison avec les meilleurs résultats connus. Ensuite, un arbre de décision est appliqué avec succès pour extraire de la connaissance ; cette connaissance combinée avec trois heuristiques proposées est utilisée pour la résolution du SMTWT et surpasse quelques règles de priorité bien connues pour leurs bonnes performances pour la résolution de ce type de problèmes avec un écart de 2819,15% en termes de TWT, en excluant H1, H2, et H3. Les expérimentations sont menées sur des problèmes SMTWT constitués de 40, 50, et 100 tâches.

Des pistes de recherches futures doivent se focaliser sur l'amélioration de la qualité des solutions en améliorant l'algorithme génétique ou en utilisant d'autres méthodes comme la recherche tabou ou l'optimisation par essaim particulaire entre autres. Il serait aussi intéressant de considérer d'autres techniques de fouille de données comme les réseaux de neurones, ou d'autres variantes d'arbres de décision et d'évaluer leur impact lors de la résolution de problèmes SMTWT. Il est aussi possible de s'attaquer à des problèmes d'ordonnancement plus complexes comme le Flow Shop ou le Job Shop avec un grand nombre de tâches et de machines.

Conclusion Générale

Le contexte de notre travail concerne l'ordonnancement dans les ateliers de production Job Shop et Single Machine. Nous avons pour cela élaboré plusieurs approches comprenant la fouille de données pour la résolution de ces problèmes. La première contribution est faite sur un problème classique de jobs shop noté $J//C_{max}$ et comporte deux volets :

- Le premier volet s'appuie sur l'utilisation de règles de priorité pour la résolution du problème Job Shop. Lors de la résolution par simulation, plusieurs règles de priorité sont considérées et chaque machine de l'atelier peut se avoir attribuée une règle différente des autres machines dans le but de minimiser le makespan (C_{max}). L'identification et l'attribution des meilleures règles de priorité sont réalisées en utilisant la simulation et une technique de force brute qui teste toutes les combinaisons possibles et renvoie la meilleure. Les données issues de cette opération sont continuellement enregistrées et stockées dans une base de données en vue de leur exploitation durant le second volet,
- Lors du deuxième volet les données obtenues par simulation sont analysées, prétraitées, et un modèle de décision est créé à l'aide de techniques de fouille de données. Ce modèle est alors capable d'attribuer une règle de priorité à chaque machine en se basant sur les attributs des tâches sur la file d'attente de cette dernière, ce qui permet de minimiser le temps d'exécution global (makespan). Cette opération d'affectation est faite en temps-réel permettant de surpasser les autres méthodes de résolution qui nécessitent un temps de calcul assez important.

Dans la deuxième contribution, nous apportons une amélioration au processus d'identification des règles de priorité (premier volet de la première contribution). Au lieu d'utiliser une technique de force brute pour identifier les meilleurs ensembles de règles de priorité, nous avons introduit un algorithme génétique qui permet de retrouver ces ensembles rapidement en explorant efficacement l'espace de recherche. L'algorithme génétique ne souffre pas de problèmes de changement d'échelle comme c'est le cas pour la méthode de force brute, lorsque le nombre de machines et/ou de règles de priorité considérés augmente. Durant cette deuxième approche le module de fouille de données est aussi amélioré en introduisant une comparaison des tâches une-à-une basée sur différents attributs. Cette comparaison permet ainsi de créer un modèle de décision plus efficace et qui par la suite a été utilisé pour résoudre de nouveaux problèmes d'affectation de règles de priorité dans un atelier Job Shop prouvant ainsi l'intérêt d'une telle approche à la fois pour le problème $J//C_{max}$ mais aussi pour l'ordonnancement dynamique, le cas échéant.

Enfin, lors de la troisième et dernière contribution, la fouille de données est utilisée afin d'analyser et reproduire le comportement d'un algorithme génétique utilisé pour la résolution du problème Single Machine $1//\sum w_i T_i$. Dans cette approche un ensemble de problèmes Single Machine sont résolus à l'aide d'un algorithme génétique modifié et les solutions trouvées sont stockées. Par la suite, les tâches sont comparées une-à-une et à l'aide de solutions obtenues préalablement des fichiers de données sont construits. Ces fichiers sont utilisés par la fouille de données afin créer un arbre de décision. Ce dernier combiné avec trois heuristiques proposées

est utilisé pour résoudre de nouveaux problèmes Single Machine surpassant des règles de priorité issues de la littérature connues pour leurs bonnes performances pour ce type de problèmes.

A base des résultats obtenus par les différentes approches, nous pouvons conclure que l'utilisation de la fouille de données permet d'aider lors de la résolution de problèmes d'ordonnancement, à la fois en générant de nouvelles règles de priorité plus efficaces, mais aussi en automatisant la sélection des règles de priorité en temps-réel. Il est ainsi possible d'en apercevoir toute l'utilité pour l'ordonnancement dynamique pour lequel l'utilisation des méthodes de résolutions classiques s'avérait improductive.

Ainsi, lors de futures contributions il serait utile de reprendre les différentes approches proposées et de les orienter vers l'ordonnancement dynamique où l'apport des systèmes d'aide à la décision et les systèmes capables de résoudre les problèmes en temps-réel est beaucoup plus considérable. Par exemple en cas de panne de machine, renvoyer en temps-réel la meilleure règle de priorité à utiliser permet de minimiser l'impact de cette perturbation contrairement aux autres méthodes de résolution classiques qui nécessitent de suspendre l'exécution de toutes les tâches, de trouver une solution, et de reprendre ensuite leur exécution. Dans d'autres contributions, il serait intéressant de mener de nouvelles expérimentations à bases de techniques de fouille de données autres que les arbres de décision, comme les réseaux de neurones ou les réseaux bayésiens.

Bibliographie

[1] J. Jimenez, A. Bekrar, D. Trentesaux, G. Z. Rey, and P. Leitao, "Governance mechanism in control architectures for flexible manufacturing systems," *IFAC-PapersOnLine*, vol. 48, no. 3, pp. 1093–1098, 2015.

[2] C. Pach, "ORCA : Architecture hybride pour le contrôle de la myopie dans le cadre du pilotage des Systèmes Flexibles de Production," Université de Valenciennes et du Hainaut-Cambrésis, 2013.

[3] T. Sen, J. M. Sulek, and P. Dileepan, "Static scheduling research to minimize weighted and unweighted tardiness: A state-of-the-art survey," *Int. J. Prod. Econ.*, vol. 83, no. 1, pp. 1–12, 2003.

[4] D. Ouelhadj and S. Petrovic, "A survey of dynamic scheduling in manufacturing systems," *J. Sched.*, vol. 12, no. 4, pp. 417–431, 2009.

[5] M. O. Adamu and a O. Adewumi, "A survey of single machine scheduling to minimize weighted number of tardy jobs," *J. Ind. Manag. Optim.*, vol. 10, no. 1, pp. 219–241, 2014.

[6] T. Chaari, S. Chaabane, N. Aissani, and D. Trentesaux, "Scheduling under uncertainty : survey and research directions," in *Advanced Logistics and Transport (ICALT), 2014 International Conference on*, 2014, pp. 229–234.

[7] B. Çaliş and S. Bulkan, "A research survey: review of AI solution strategies of job shop scheduling problem," *J. Intell. Manuf.*, vol. 26, no. 5, pp. 961–973, 2015.

[8] I. A. Chaudhry and A. A. Khan, "A research survey: Review of flexible job shop scheduling techniques," *Int. Trans. Oper. Res.*, vol. 23, no. 3, pp. 551–591, 2016.

[9] M. Yousefi, M. Yousefi, D. Hooshyar, and J. Ataide De Souza Oliveira, "An evolutionary approach for solving the job shop scheduling problem in a service industry," *Int. J. Adv. Intell. Informatics*, vol. 1, no. 1, pp. 2442–6571, 2015.

[10] S. Nguyen, "Automatic Design of Dispatching Rules for Job Shop Scheduling with Genetic Programming," Victoria University ofWellington, 2013.

[11] D. Applegate and W. Cook, "A Computational Study of the Job-Shop Scheduling Problem," *ORSA J. Comput.*, vol. 3, no. 2, pp. 149–156, May 1991.

[12] C. N. Potts and L. N. Van Wassenhove, "A Branch and Bound Algorithm for the Total Weighted Tardiness Problem," *Oper. Res.*, vol. 33, no. 2, pp. 363–377, Apr. 1985.

[13] M. Gen, Y. Tsujimura, and E. Kubota, "Solving job-shop scheduling problems by genetic algorithm," *Proc. IEEE Int. Conf. Syst. Man Cybern.*, vol. 2, pp. 1577–1582, 1994.

[14] M. S. Jayamohan and C. Rajendran, "New dispatching rules for shop scheduling: A step forward," *Int. J. Prod. Res.*, vol. 38, no. 3, pp. 563–586, 2000.

[15] K. Natarajan, K. M. Mohanasundaram, B. S. Babu, S. Suresh, K. A. A. D. Raj, and C. Rajendran, "Performance evaluation of priority dispatching rules in multi-level assembly job shops with jobs having weights for flowtime and tardiness," *Int. J. Adv. Manuf. Technol.*, vol. 31, no. 7–8, pp. 751–761, 2007.

[16] G. Ochoa, J. A. Vazquez-Rodriguez, S. Petrovic, and E. Burke, "Dispatching rules for production scheduling: A hyper-heuristic landscape analysis," *2009 IEEE Congr. Evol. Comput.*, pp. 1873–1880, May 2009.

[17] A. K. Kaban, Z. Othman, and D. S. Rohmah, "Comparison of dispatching rules in job-shop scheduling problem using simulation: a case study," *Int. J. Simul. Model.*, vol. 11, no. 3, pp. 129–140, Sep. 2012.

[18] B. Chen and T. I. Matis, "A flexible dispatching rule for minimizing tardiness in job shop scheduling," *Int. J. Prod. Econ.*, vol. 141, no. 1, pp. 360–365, Jan. 2013.

[19] P. Korytkowski, T. Wiśniewski, and S. Rymaszewski, "An evolutionary simulation-based optimization approach for dispatching scheduling," *Simul. Model. Pract. Theory*, vol. 35, pp. 69–85, 2013.

[20] S. Nguyen, M. Zhang, M. Johnston, and K. C. Tan, "Learning iterative dispatching rules for job shop scheduling with genetic programming," *Int. J. Adv. Manuf. Technol.*, vol. 67, no. 1–4, pp. 85–100, 2013.

[21] H.-L. Fan, H.-G. Xiong, G.-Z. Jiang, and G.-F. Li, "Survey of the selection and evaluation for dispatching rules in dynamic job shop scheduling problem," in *2015 Chinese Automation Congress (CAC)*, 2015, pp. 1926–1931.

[22] G. Metan, I. Sabuncuoglu, and H. Pierreval, "Real time selection of scheduling rules and knowledge extraction via dynamically controlled data mining," *Int. J. Prod. Res.*, vol. 48, no. 23, pp. 6909–6938, Dec. 2010.

[23] S. Nguyen, M. Zhang, M. Johnston, and K. C. Tan, "A Computational Study of Representations in Genetic Programming to Evolve Dispatching Rules for the Job Shop Scheduling Problem," *IEEE Trans. Evol. Comput.*, vol. 17, no. 5, pp. 621–639, Oct. 2013.

[24] A. Shahzad and N. Mebarki, "Learning Dispatching Rules for Scheduling: A Synergistic View Comprising Decision Trees, Tabu Search and Simulation," *Computers*, vol. 5, no. 1, p. 3, 2016.

[25] C. Song, H. Luo, T. Qu, H. L. Lv, and G. Q. Huang, "A Simulation Approach to Hybrid Flowshop Scheduling Using Dispatching Rules," in *Proceedings of the 6th CIRP-Sponsored International Conference on Digital Enterprise Technology*, vol. 66, G. Q. Huang, K. L. Mak, and P. G. Maropoulos, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 877–884.

[26] E. B. da Silva, M. G. Costa, M. F. de S. da Silva, and F. H. Pereira, "Simulation study of dispatching rules in stochastic job shop dynamic," *World J. Model. Simul.*, vol. 10, no. 3, pp. 231–240, 2014.

[27] A. Subramanian, M. Battarra, and C. N. Potts, "An Iterated Local Search heuristic for the single machine total weighted tardiness scheduling problem with sequence-dependent setup times," *Int. J. Prod. Res.*, vol. 52, no. 9, pp. 2729–2742, May 2014.

[28] H. Boukef Ben Othman, "Sur l'ordonnancement d'ateliers job-shop flexibles et flow-shop en industries pharmaceutiques Optimisation par algorithmes génétiques et essaims particulaires," École Nationale d'Ingénieurs de Tunis, 2009.

[29] Y. Harrath, "Contribution à l'ordonnancement conjoint de la production et de la

maintenance : Application au cas d'un job shop," Université de Franche-Comté, 2003.

[30] F. Tangour Toumi, "Ordonnancement Dynamique dans les Industries Agroalimentaires," Université des Sciences et Technologies de Lille, 2007.

[31] M. A. Shahzad, "Une Approche Hybride de Simulation-Optimisation Basée sur la Fouille de Données pour les Problèmes d'ordonnancement," École polytechnique de l'Université de Nantes, 2011.

[32] F. Benbouzid Sitayeb, "Contribution à l'étude de la performance et de la robustesse des ordonnancements conjoints production/maintenance – cas du flow shop," Université de Franche-Comté, 2005.

[33] R. Djeridi, "Contribution à la maîtrise de la disponibilité de systèmes complexes : Proposition d'une méthode de réordonnancement proactif de la maintenance," École Nationale Supérieure d'Arts et Métiers ParisTech, 2010.

[34] I. Kacem, "Genetic algorithm for the flexible job-shop scheduling problem," in *International Conference on Systems, Man and Cybernetics SMC'03*, 2003, vol. 4, pp. 3464–3469.

[35] M. Habib Zahmani, B. Atmani, and A. Bekrar, "Efficient Dispatching Rules Based on Data Mining for the Single Machine Scheduling Problem," in *Computer Science & Information Technology (CS & IT)*, 2015, pp. 199–208.

[36] A. Caumond, "Le problème de jobshop avec contraintes : modélisation et optimisation," Université Blaise Pascal Clermont Ferrand II, 2006.

[37] Y.-G. Huang, L. V. N. Kanal, and S. K. Tripathi, "Reactive scheduling for a single machine: problem definition, analysis, and heuristic solution," *Int. J. Comput. Integr. Manuf.*, vol. 3, no. 1, pp. 6–12, 1990.

[38] A. T. Unal, R. Uzsoy, and A. S. Kiran, "Rescheduling on a single machine with part-type dependent setup times and deadlines," *Ann. Oper. Res.*, vol. 70, pp. 93–113, 1997.

[39] V. a. Armentano and R. Mazzini, "A genetic algorithm for scheduling on a single machine with set-up times and due dates," *Prod. Plan. Control*, vol. 11, no. 7, pp. 713–720, Jan. 2000.

[40] D. Biskup and M. Feldmann, "Benchmarks for scheduling on a single machine against restrictive and unrestrictive common due dates," *Comput. Oper. Res.*, vol. 28, pp. 787–801, 2001.

[41] M. A. Aloulou, M. Portmann, and A. Vignier, "Predictive-reactive scheduling for the single machine problem," in *8th Workshop on Project Management and Scheduling*, 2002, pp. 3–5.

[42] W. Jang, "Dynamic scheduling of stochastic jobs on a single machine," *Eur. J. Oper. Res.*, vol. 138, no. 3, pp. 518–530, 2002.

[43] M. Feldmann and D. Biskup, "Single-machine scheduling for minimizing earliness and tardiness penalties by meta-heuristic approaches," *Comput. Ind. Eng.*, vol. 44, pp. 307–323, 2003.

[44] E. J. Anderson and C. N. Potts, "Online scheduling of a single machine to minimize total weighted completion time," *Math. Oper. Res.*, vol. 29, no. 3, pp. 686–697, 2004.

[45] M. Sevaux and K. Sörensen, "A genetic algorithm for robust schedules in a one-machine environment with ready times and due dates," *4OR*, vol. 2, no. 2, pp. 129–147, Jul. 2004.

[46] N. Liu, M. A. Abdelrahman, and S. Ramaswamy, "A genetic algorithm for the single machine total weighted tardiness problem," *Int. J. Intell. Control Syst.*, vol. 10, no. 3, pp. 218–225, 2005.

[47] N. Kasap, H. Aytug, and A. Paul, "Minimizing makespan on a single machine subject to random breakdowns," *Oper. Res. Lett.*, vol. 34, no. 1, pp. 29–36, 2006.

[48] F. Der Chou, P. C. Chang, and H. M. Wang, "A hybrid genetic algorithm to minimize makespan for the single batch machine dynamic scheduling problem," *Int. J. Adv. Manuf. Technol.*, vol. 31, no. 3–4, pp. 350–359, 2006.

[49] L. Liu, H. Y. Gu, and Y. G. Xi, "Robust and stable scheduling of a single machine with random machine breakdowns," *Int. J. Adv. Manuf. Technol.*, vol. 31, no. 7–8, pp. 645–654, 2007.

[50] A. Ferrolho and M. Crisóstomo, "Single machine total weighted tardiness problem with genetic algorithms," *2007 IEEE/ACS Int. Conf. Comput. Syst. Appl. AICCSA 2007*, pp. 1–8, 2007.

[51] R. Maheswaran, S. G. Ponnambalam, and N. Jawahar, "Hybrid heuristic algorithms for single machine total weighted tardiness scheduling problems," *Int. J. Intell. Syst. Technol. Appl.*, vol. 4, no. 1–2, pp. 34–56, 2008.

[52] J. Tian, R. Fu, and J. Yuan, "A best on-line algorithm for single machine scheduling with small delivery times," *Theor. Comput. Sci.*, vol. 393, no. 1–3, pp. 287–293, 2008.

[53] F. D. Chou, "An experienced learning genetic algorithm to solve the single machine total weighted tardiness scheduling problem," *Expert Syst. Appl.*, vol. 36, no. 2 PART 2, pp. 3857–3865, 2009.

[54] Q. Li and B. Wang, "A stable scheduling for single machine under uncertainty," *Proc. 2009 IEEE Int. Conf. Autom. Logist. ICAL 2009*, no. August, pp. 526–531, 2009.

[55] M. J. Geiger, "On heuristic search for the single machine total weighted tardiness problem - Some theoretical insights and their empirical verification," *Eur. J. Oper. Res.*, vol. 207, no. 3, pp. 1235–1243, 2010.

[56] S. Olafsson and X. Li, "Learning effective new single machine dispatching rules from optimal scheduling data," *Int. J. Prod. Econ.*, vol. 128, no. 1, pp. 118–126, 2010.

[57] A. Azadeh, A. Keramati, A. Karimi, and M. Moghaddam, "A multi-objective genetic algorithm for scheduling optimisation of m job families on a single machine," *Int. J. Ind. Syst. Eng.*, vol. 6, no. 4, p. 417, 2010.

[58] Q. Li, L. Liang, and W. Qiao, "A hybrid robust scheduling for single machine subject to random machine breakdown," *Proc. 4th Int. Work. Adv. Comput. Intell. IWACI 2011*, pp. 700–705, 2011.

[59] X. Cai, X. Wu, and X. Zhou, “Scheduling deteriorating jobs on a single machine subject to breakdowns,” *J. Sched.*, vol. 14, no. 2, pp. 173–186, 2011.

[60] C. C. Lu, S. W. Lin, and K. C. Ying, “Robust scheduling on a single machine to minimize total flow time,” *Comput. Oper. Res.*, vol. 39, no. 7, pp. 1682–1691, 2012.

[61] A. Yin and J. Wang, “Mixed Dispatch Rule for Single Machine Total Weighted Tardiness Problem,” *J. Appl. Sci.*, vol. 13, no. 21, pp. 4616–4619, 2013.

[62] I. Kacem, A. Nagih, and M. Seifaddini, “Maximum lateness minimization with positive tails on a single machine with an unexpected non-availability interval,” in *2014 World Congress on Computer Applications and Information Systems (WCCAIS)*, 2014, pp. 1–5.

[63] R. M’Hallah and A. Alhajraf, “Ant colony systems for the single-machine total weighted earliness tardiness scheduling problem,” *J. Sched.*, vol. 19, no. 2, pp. 191–205, Apr. 2016.

[64] O. Herr and A. Goel, “Minimising total tardiness for a single machine scheduling problem with family setups and resource constraints,” *Eur. J. Oper. Res.*, vol. 248, no. 1, pp. 123–135, 2016.

[65] C. Koulamas, “The single-machine total tardiness scheduling problem: Review and extensions,” *Eur. J. Oper. Res.*, vol. 202, no. 1, pp. 1–7, 2010.

[66] C. Rajendran and O. Holthaus, “A comparative study of dispatching rules in dynamic flowshops and jobshops,” *Eur. J. Oper. Res.*, vol. 116, no. 1, pp. 156–170, Jul. 1999.

[67] N. Grangeon, A. Tanguy, and N. Tchernev, “Generic Simulation Model for Hybrid Flow-Shop,” *Comput. Ind. Eng.*, vol. 37, pp. 207–210, 1999.

[68] H. Khademi Zare and M. B. Fakhrzad, “Solving flexible flow-shop problem with a hybrid genetic algorithm and data mining: A fuzzy approach,” *Expert Syst. Appl.*, vol. 38, no. 6, pp. 7609–7615, Jun. 2011.

[69] R. Balasundaram, N. Baskar, and R. S. Sankar, “A New Approach to Generate Dispatching Rules for Two Machine Flow Shop Scheduling Using Data Mining,” *Procedia Eng.*, vol. 38, pp. 238–245, 2012.

[70] F. Della Croce, R. Tadei, and G. Volta, “A genetic algorithm for the job shop problem,” *Computers & Operations Research*, vol. 22, no. 1. pp. 15–24, 1995.

[71] O. Holthaus and C. Rajendran, “Efficient dispatching rules for scheduling in a job shop,” *Int. J. Prod. Econ.*, vol. 48, no. 1, pp. 87–105, Jan. 1997.

[72] I. Sabuncuoglu and M. Bayız, “Analysis of reactive scheduling problems in a job shop environment,” *Eur. J. Oper. Res.*, vol. 126, no. 3, pp. 567–586, Nov. 2000.

[73] D. Koonce and S. Tsai, “Using data mining to find patterns in genetic algorithm solutions to a job shop schedule,” *Comput. Ind. Eng.*, vol. 38, no. 3, pp. 361–374, Oct. 2000.

[74] L. Wang and D.-Z. Zheng, “An effective hybrid optimization strategy for job-shop scheduling problems,” *Comput. Oper. Res.*, vol. 28, no. 6, pp. 585–596, 2001.

[75] H. Ren and M. X. Weng, “An efficient priority rule for scheduling job shops to minimize mean tardiness,” *IIE Trans.*, vol. 38, no. 9, pp. 789–795, 2006.

[76] T. Chiang, “Using dispatching rules for job shop scheduling with due date-based objectives,” in *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.*, 2006, no. May, pp. 1426–1431.

[77] V. P. Eswaramurthy and A. Tamilarasi, “Tabu Search Strategies for Solving Job Shop Scheduling Problems,” *J. Adv. Manuf. Syst.*, vol. 6, no. 1, pp. 59–75, Jun. 2007.

[78] H. Suwa and H. Sandoh, “Capability of cumulative delay based reactive scheduling for job shops with machine breakdowns,” *Comput. Ind. Eng.*, vol. 53, pp. 63–78, 2007.

[79] A. Baykasoğlu, M. Göçken, L. Özbakır, and S. Kulluk, “Modelling, Computation and Optimization in Information Systems and Management Sciences,” in *Modelling, Computation and Optimization in Information Systems and Management Sciences*, vol. 14, H. A. Le Thi, P. Bouvry, and T. Pham Dinh, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 389–398.

[80] F. Pezzella, G. Morganti, and G. Ciaschetti, “A genetic algorithm for the Flexible Job-shop Scheduling Problem,” *Comput. Oper. Res.*, vol. 35, no. 10, pp. 3202–3212, Oct. 2008.

[81] C. Özgüven, L. Özbakır, and Y. Yavuz, “Mathematical models for job-shop scheduling problems with routing and process plan flexibility,” *Appl. Math. Model.*, vol. 34, no. 6, pp. 1539–1548, Jun. 2010.

[82] A. Shahzad and N. Mebarki, “Discovering dispatching rules for job shop scheduling problem through data mining,” in *8th International Conference of Modeling and Simulation - MOSIM’10*, 2010.

[83] L.-N. Xing, Y.-W. Chen, P. Wang, Q.-S. Zhao, and J. Xiong, “A Knowledge-Based Ant Colony Optimization for Flexible Job Shop Scheduling Problems,” *Appl. Soft Comput.*, vol. 10, no. 3, pp. 888–896, Jun. 2010.

[84] A. M. Kuczapski, M. V. Micea, L. A. Maniu, and V. I. Cretu, “Efficient generation of near optimal initial populations to enhance genetic algorithms for Job-Shop Scheduling,” *Inf. Technol. Control*, vol. 39, no. 1, pp. 32–37, 2010.

[85] S. M. K. Hasan, R. Sarker, and D. Essam, “Genetic algorithm for job-shop scheduling with machine unavailability and breakdowns,” *Int. J. Prod. Res.*, vol. 49, no. 16, pp. 4999–5015, Aug. 2011.

[86] K. Ritwik and S. Deb, “A Genetic Algorithm-Based Approach for Optimization of Scheduling in Job Shop Environment,” *J. Adv. Manuf. Syst.*, vol. 10, no. 2, pp. 223–240, Dec. 2011.

[87] H. L. Lu, G. Q. Huang, and H. D. Yang, “Integrating order review/release and dispatching rules for assembly job shop scheduling using a simulation approach,” *Int. J. Prod. Res.*, vol. 49, no. 3, pp. 647–669, Feb. 2011.

[88] M. Kapanoglu and M. Alikalfa, “Learning IF-THEN priority rules for dynamic job shops using genetic algorithms,” *Robot. Comput. Integr. Manuf.*, vol. 27, no. 1, pp. 47–55, 2011.

[89] J. F. Wang, B. Q. Du, and H. M. Ding, “A Genetic Algorithm for the Flexible Job-Shop Scheduling Problem,” in *Advanced Research on Computer Science and Information Engineering*, vol. 152, H. X. (eds) Shen G., Ed. Berlin, Heidelberg: Springer Berlin

Heidelberg, 2011, pp. 332–339.

[90] K. Bülbül and P. Kaminsky, “A linear programming-based method for job shop scheduling,” *J. Sched.*, vol. 16, no. 2, pp. 161–183, Feb. 2012.

[91] A. Shahzad and N. Mebarki, “Data mining based job dispatching using hybrid simulation-optimization approach for shop scheduling problem,” *Eng. Appl. Artif. Intell.*, vol. 25, no. 6, pp. 1173–1181, 2012.

[92] D. Trentesaux, C. Pach, A. Bekrar, Y. Sallez, T. Berger, T. Bonte, P. Leitão, and J. Barbosa, “Benchmarking flexible job-shop scheduling and control systems,” *Control Eng. Pract.*, vol. 21, no. 9, pp. 1204–1225, Sep. 2013.

[93] P. Sharma and A. Jain, “Performance analysis of dispatching rules in a stochastic dynamic job shop manufacturing system with sequence-dependent setup times: Simulation approach,” *CIRP J. Manuf. Sci. Technol.*, vol. 10, pp. 110–119, Aug. 2015.

[94] Y. Xu, L. Wang, S. yao Wang, and M. Liu, “An effective teaching-learning-based optimization algorithm for the flexible job-shop scheduling problem with fuzzy processing time,” *Neurocomputing*, vol. 148, pp. 260–268, 2015.

[95] N. Said, W. Mouelhi, and K. Ghedira, “Classification Rules for the Job Shop Scheduling Problem with Machine Breakdowns,” *Int. J. Inf. Electron. Eng.*, vol. 5, no. 4, pp. 300–304, 2015.

[96] A. K. Gupta and A. I. Sivakumar, “Job shop scheduling techniques in semiconductor manufacturing,” *Int. J. Adv. Manuf. Technol.*, vol. 27, pp. 1163–1169, 2006.

[97] D. Bai, Z. H. Zhang, and Q. Zhang, “Flexible open shop scheduling problem to minimize makespan,” *Comput. Oper. Res.*, vol. 67, pp. 207–215, 2016.

[98] L. Zhang, L. Lu, and J. Yuan, “Two-machine open-shop scheduling with rejection to minimize the makespan,” *OR Spectr.*, vol. 38, no. 2, pp. 519–529, 2016.

[99] M. Habib Zahmani, B. Atmani, A. Bekrar, and N. Aissani, “Multiple priority dispatching rules for the job shop scheduling problem,” in *2015 3rd International Conference on Control, Engineering & Information Technology (CEIT)*, 2015, pp. 1–6.

[100] J. Carlier, “The one-machine sequencing problem,” *Eur. J. Oper. Res.*, vol. 11, no. 1, pp. 42–47, 1982.

[101] M. R. Garey and D. S. Johnson, “Computer and intractability,” *A Guid. to Theory NP-Completeness*, 1979.

[102] J. K. Lenstra, A. H. G. Rinnooy Kan, and P. Brucker, “Complexity of Machine Scheduling Problems,” *Ann. Discret. Math.*, vol. 1, pp. 343–362, 1977.

[103] I. Mhedhbi Brinis, “Ordonnancement d’ateliers de traitements de surfaces pour une production mono-robot/multi-produits : Résolution et étude de la robustesse,” Ecole Centrale de Lille, 2011.

[104] A. Morvan, “Utilisation du modèle polyédrique pour la synthèse d’architectures pipelinées,” Ecole normale sup_erieure de Cachan, 2013.

[105] R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. H. G. R. Kan, “Optimization and

Approximation in Deterministic Sequencing and Scheduling: a Survey," *Ann. Discret. Math.*, vol. 5, pp. 287–326, 1979.

[106] J. Schauer and C. Schwarz, "Job-shop scheduling in a body shop," *J. Sched.*, vol. 16, no. 2, pp. 215–229, Nov. 2012.

[107] H. Metta, "Adaptive , Multi-Objective Job Shop Scheduling Using Genetic," University of Kentucky, 2008.

[108] J. Blazewicz, K. Ecker, E. Pesch, G. Schmidt, and J. Weglarz, *Handbook on Scheduling From Theory to Applications*. 2007.

[109] R. E. Bellman and R. S. Roth, *The Bellman Continuum: A Collection of the Works of Richard E. Bellman*. World Scientific, 1986.

[110] Z. Liu and Y. K. Ro, "Rescheduling for machine disruption to minimize makespan and maximum lateness," *J. Sched.*, vol. 17, no. 4, pp. 339–352, 2014.

[111] T. Hildebrandt, J. Heger, and B. Scholz-Reiter, "Towards improved dispatching rules for complex shop floor scenarios: a genetic programming approach," in *GECCO '10: Proceedings of the 12th annual conference on Genetic and evolutionary computation*, 2010, pp. 257–264.

[112] C. Hicks and P. Pongcharoen, "Dispatching rules for production scheduling in the capital goods industry," *Int. J. Prod. Econ.*, vol. 104, no. 1, pp. 154–163, Nov. 2006.

[113] N. M. Sadeh and Y. Nakakuki, "Focused simulated annealing search: An application to job shop scheduling," *Ann. Oper. Res.*, vol. 63, no. 1, pp. 77–103, Feb. 1996.

[114] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simmulated annealing," *Science (80-.).*, vol. 220, pp. 671–680, 1983.

[115] F. Glover, "Future paths for integer programming and links to artificial intelligence," *Comput. Oper. Res.*, vol. 13, no. 5, 1986.

[116] S. Kumar and C. S. P. Rao, "Application of ant colony, genetic algorithm and data mining-based techniques for scheduling," *Robot. Comput. Integr. Manuf.*, vol. 25, no. 6, pp. 901–908, Dec. 2009.

[117] J. H. Holland, *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. University of Michigan Press, 1975.

[118] A. Colorni, M. Dorigo, and V. Maniezzo, "Distributed Optimization by Ant Colonies," in *Proceedings of the first European conference on artificial life*, 1991, no. 142, pp. 134–142.

[119] R. Eberhart and J. Kennedy, "A new optimizer using particle swarm theory," in *MHS'95. Proceedings of the Sixth International Symposium on Micro Machine and Human Science*, 1995, pp. 39–43.

[120] M. Souier, "Métaheuristiques pour la manipulation de routages alternatifs dans un job shop," Université Abou Bakr Belkaid de Tlemcen, 2009.

[121] K. R. Baker, *Introduction to sequencing and scheduling*. John Wiley & Sons, 1974.

[122] T. Hildebrandt, D. Goswami, and M. Freitag, “Large-scale simulation-based optimization of semiconductor dispatching rules,” in *WSC ’14 Proceedings of the 2014 Winter Simulation Conference*, 2014, pp. 2580–2590.

[123] C. D. Geiger, R. Uzsoy, and H. Aytuğ, “Rapid Modeling and Discovery of Priority Dispatching Rules: An Autonomous Learning Approach,” *J. Sched.*, vol. 9, no. 1, pp. 7–34, Feb. 2006.

[124] F. T. S. Chan, H. K. Chan, H. C. W. Lau, and R. W. L. Ip, “Analysis of dynamic dispatching rules for a flexible manufacturing system,” *J. Mater. Process. Technol.*, vol. 138, no. 1–3, pp. 325–331, 2003.

[125] S. Panwalker and W. Iskander, “A survey of dispatching rules,” *Oper. Res.*, vol. 25, pp. 45–61, 1977.

[126] J. H. Blackstone, D. T. Phillips, and G. L. Hogg, “A state-of-the-art survey of dispatching rules for manufacturing job shop operations,” *Int. J. Prod. Res.*, vol. 20, no. 1, pp. 27–45, Jan. 1982.

[127] K. Kemppainen, “Priority scheduling revisited – dominant rules, open protocols, and integrated order management,” UNIVERSITATIS OECONOMICAE HELSINGIENSIS, 2005.

[128] T. Lejmi and I. Sabuncuoglu, “Effect of load, processing time and due date variation on the effectiveness of scheduling rules,” *Int. J. Prod. Res.*, vol. 40, no. 4, pp. 945–974, 2002.

[129] E. Taillard, “Benchmarks for basic scheduling problems,” *European Journal of Operational Research*, vol. 64. pp. 278–285, 1993.

[130] E. Demirkol, M. Sanjay, and U. Reha, “Benchmarks for shop scheduling problems,” *Eur. J. Oper. Res.*, vol. 109, no. 1, pp. 137–141, 1998.

[131] F. Qiao, L. Li, Y. Ma, and B. Shi, “Single Machine Oriented Match-Up Rescheduling Method for Semiconductor Manufacturing System,” in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 7507 LNAI, no. PART 2, L. H. (eds) Su CY., Rakheja S., Ed. Springer, 2012, pp. 217–226.

[132] S. H. Yoon and I. S. Lee, “New constructive heuristics for the total weighted tardiness problem,” *J. Oper. Res. Soc.*, vol. 62, no. 1, pp. 232–237, 2011.

[133] H. P. Huang and T. Y. Chen, “On-line rescheduling for semiconductor manufacturing,” *2006 IEEE Int. Conf. Autom. Sci. Eng. CASE*, pp. 106–111, 2007.

[134] M. Cheng, M. Sugi, J. Ota, M. Yamamoto, H. Ito, and K. Inoue, “Online rescheduling in semiconductor manufacturing,” in *IEEE International Conference on Intelligent Robots and Systems*, 2006, pp. 2245–2250.

[135] R. W. Conway, “Priority dispatching and job lateness in a job shop,” *J. Ind. Eng.*, vol. 16, no. 4, p. 228, 1965.

[136] M. H. Bulkin, J. L. Colley, and H. W. Steinhoff Jr, “Load forecasting, priority sequencing, and simulation in a job shop control system,” *Manage. Sci.*, vol. 13, no. 2, p. B--29, 1966.

[137] S. EILON and D. J. COTTERJLL, “A modified SI rule in job shop scheduling,” *Int. J. Prod. Res.*, vol. 7, no. 2, pp. 135–145, 1968.

[138] C. T. Baker and B. P. Dzielinski, “Simulation of a simplified job shop,” *Manage. Sci.*, vol. 6, no. 3, pp. 311–323, 1960.

[139] D. C. Carroll, “Heuristic sequencing of jobs with single and multiple components,” *Cambridge, MA Sloan Sch. Mgmt., Massachusetts Inst. Technol.*, 1965.

[140] B. Giffler and G. L. Thompson, “Algorithms for solving production-scheduling problems,” *Oper. Res.*, vol. 8, no. 4, pp. 487–503, 1960.

[141] J. E. Beasley, “OR-Library: Distributing Test Problems by Electronic Mail,” *J. Oper. Res. Soc.*, vol. 41, no. 11, p. 1069, Nov. 1990.

[142] S. Albers and G. Schmidt, “Scheduling with unexpected machine breakdowns,” *Discret. Appl. Math.*, vol. 110, no. 2–3, pp. 85–99, 2001.

[143] A. Hassam, “Développement et analyse de méthodes d’ordonnancement temps réel pour les systèmes flexibles de production,” Université Abou Bakr Belkaid de Tlemcen, 2012.

[144] H. Pierreval and N. Mebarki, “Dynamic scheduling selection of dispatching rules for manufacturing system,” *Int. J. Prod. Res.*, vol. 35, no. 6, pp. 1575–1591, Jun. 1997.

[145] K.-C. Jeong and Y.-D. Kim, “A real-time scheduling mechanism for a flexible manufacturing system: Using simulation and dispatching rules,” *Int. J. Prod. Res.*, vol. 36, no. 9, pp. 2609–2626, Sep. 1998.

[146] A. P. J. Vepsalainen and T. E. Morton, “Priority rules for job shops with weighted tardiness costs,” *Manage. Sci.*, vol. 33, no. 8, pp. 1035–1047, 1987.

[147] W. H. Hausman and G. D. Scudder, “Priority scheduling rules for repairable inventory systems,” *Manage. Sci.*, vol. 28, no. 11, pp. 1215–1232, 1982.

[148] T. E. Morton and D. W. Pentico, “Heuristic scheduling systems,” *J. Oper. Res. Soc.*, vol. 46, no. 4, pp. 543–544, 1995.

[149] C. R. SCHULTZ, “An expediting heuristic for the shortest processing time dispatching rule,” *Int. J. Prod. Res.*, vol. 27, no. 1, pp. 31–41, 1989.

[150] S. C. Aggarwal, F. PAUL WYMAN, and B. A. McCARL, “An investigation of a cost-based rule for job-shop scheduling,” *Int. J. Prod. Res.*, vol. 11, no. 3, pp. 247–261, 1973.

[151] W. L. Berry and V. Rao, “Critical ratio scheduling: an experimental analysis,” *Manage. Sci.*, vol. 22, no. 2, pp. 192–201, 1975.

[152] E. J. Anderson and J. C. Nyirenda, “Two new rules to minimize tardiness in a job shop,” *Int. J. Prod. Res.*, vol. 28, no. 12, pp. 2277–2292, 1990.

[153] A. A. H. Brown, “Dispatching Work : Finding the best dispatching method for real job-shops by,” University of Waterloo, 2014.

[154] A. S. Spachis and J. R. King, “Job-shop scheduling heuristics with local neighbourhood search,” *Int. J. Prod. Res.*, vol. 17, no. 6, pp. 507–526, 1979.

[155] K. R. Baker and J. W. M. Bertrand, "A dynamic priority rule for scheduling against due-dates," *J. Oper. Manag.*, vol. 3, no. 1, pp. 37–42, 1982.

[156] I. Kurtulus and E. W. Davis, "Multi-project scheduling: Categorization of heuristic rules performance," *Manage. Sci.*, vol. 28, no. 2, pp. 161–172, 1982.

[157] S.-J. Chen and L. I. Lin, "Reducing total tardiness cost in manufacturing cell scheduling by a multi-factor priority rule," *Int. J. Prod. Res.*, vol. 37, no. 13, pp. 2939–2956, 1999.

[158] T. R. HOFFMANN and G. D. SCUDDER, "Priority scheduling with cost considerations," *Int. J. Prod. Res.*, vol. 21, no. 6, pp. 881–889, 1983.

[159] T. S. Raghu and C. Rajendran, "An efficient dynamic dispatching rule for scheduling in a job shop," *Int. J. Prod. Econ.*, vol. 32, no. 3, pp. 301–313, 1993.

[160] S. Miyazaki, "Combined scheduling system for reducing job tardiness in a job shop," *Int. J. Prod. Res.*, vol. 19, no. 2, pp. 201–211, 1981.

[161] M. Oral and J.-L. Malouin, "Evaluation of the shortest processing time scheduling rule with truncation process," *AIIE Trans.*, vol. 5, no. 4, pp. 357–365, 1973.

[162] M. Moser and S. Engell, "Comprehensive Evaluation Of Paority Rules For On-line Scheduling: The Single Machine Case," in *Computer Integrated Manufacturing, 1992., Proceedings of the Third International Conference on*, 1992, pp. 403–412.

[163] J. J. Kanet and X. Li, "A weighted modified due date rule for sequencing to minimize weighted tardiness," *J. Sched.*, vol. 7, no. 4, pp. 261–276, 2004.

[164] J. Fritz and R. H. Dolores, "Using Data Mining to Explore the Regularity of Genetic Algorithms in Job Shop Schedule Problems," Ohio University, 1997.

[165] J. Solarte, "A proposed data mining methodology and its application to industrial procedures," University of Tennessee, 2002.

[166] X. Li, "Application of data mining in scheduling of single machine system," Iowa State University, 2006.

[167] B. Brahimi, "Extraction de connaissances à partir de données incomplètes et imprécises," Université de M'Sila, 2011.

[168] U. Fayyad, G. Piatetsky-Shapiro, and P. Smyth, "From Data Mining to Knowledge Discovery in Databases," *AI Mag.*, vol. 17, no. 3, p. 37, 1996.

[169] F. Pinto, "Database marketing intelligence methodology supported by ontologies and knowlegde discovery in databases," University of Minho, 2009.

[170] S. A. Mokeddem, "Fouille de Données pour L'analyse des Traces Patients," Université d'Oran 1 Ahmed Benbella, 2016.

[171] J. A. Harding, M. Shahbaz, Srinivas, and A. Kusiak, "Data Mining in Manufacturing: A Review," *Journal of Manufacturing Science and Engineering*, vol. 128, no. 4. pp. 969–976, 2006.

[172] H. Aytug, S. Bhattacharyya, G. J. Koehler, and J. L. Snowdon, "A review of machine learning in scheduling," *IEEE Trans. Eng. Manag.*, vol. 41, no. 2, pp. 165–171, May

1994.

[173] G. Anderson, T. Marwala, and F. Nelwamondo, “Use of Data Mining in Scheduler Optimization,” *arXiv Prepr. arXiv1011.1735*, 2010.

[174] A. D. Innani, “Applying data mining to job shop scheduling using regression analysis,” College of Engineering and Technology of Ohio University, 2004.

[175] H. Benamar, “Etude explorative d’outils pour le data mining,” Université du Québec à Trois-Rivières, 2007.

[176] R. Ismail, Z. Othman, and A. A. Bakar, “Data mining in production planning and scheduling: A review,” in *2009 2nd Conference on Data Mining and Optimization*, 2009, no. October, pp. 154–159.

[177] N. Aissani, B. Atmani, D. Trentesaux, and B. Beldjilali, “Extraction of Priority Rules for Boolean Induction in Distributed Manufacturing Control,” *Serv. Orientat. Holonic Multi-Agent Manuf. Robot. Stud. Comput. Intell.*, vol. 544, pp. 127–143, 2014.

[178] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, “The WEKA data mining software: an update,” *ACM SIGKDD Explor. Newsl.*, vol. 11, no. 1, pp. 10–18, 2009.

[179] B. Chebel Morello, D. Michaut, and P. Baptiste, “A knowledge discovery process for a flexible manufacturing system,” in *ETFA 2001. 8th International Conference on Emerging Technologies and Factory Automation. Proceedings (Cat. No.01TH8597)*, 2001, vol. 1, pp. 651–658.

[180] X. Li and S. Olafsson, “Discovering Dispatching Rules Using Data Mining,” *J. Sched.*, vol. 8, no. 6, pp. 515–527, Dec. 2005.

[181] C. D. Geiger and R. Uzsoy, “Learning effective dispatching rules for batch processor scheduling,” *Int. J. Prod. Res.*, vol. 46, no. 6, pp. 1431–1454, 2008.

[182] H. Ingimundardottir and T. P. Runarsson, *Supervised Learning Linear Priority Dispatch Rules for Job-Shop Scheduling*, vol. 6683. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011.

[183] Y. Wang, Y. Zhang, Y. Yu, and C. Zhang, “Data Mining Based Approach for Jobshop Scheduling,” in *Proceedings of 2013 4th International Asia Conference on Industrial Engineering and Management Innovation (IEMI2013)*, Taiwan: Springer Berlin Heidelberg, 2014, pp. 761–771.

[184] P. Priore, D. De La Fuente, A. Gomez, and J. Puente, “A review of machine learning in dynamic scheduling of flexible manufacturing systems,” *Ai Edam*, vol. 15, no. 3, pp. 251–263, 2001.

[185] D. J. Fonseca and D. Navaresse, “Artificial neural networks for job shop simulation,” *Adv. Eng. Informatics*, vol. 16, pp. 241–246, 2002.

[186] A. K. Choudhary, J. A. Harding, and M. K. Tiwari, “Data mining in manufacturing: a review based on the kind of knowledge,” *J. Intell. Manuf.*, vol. 20, no. 5, pp. 501–521, Jul. 2008.

[187] G. Köksal, İ. Batmaz, and M. C. Testik, "A review of data mining applications for quality improvement in manufacturing industry," *Expert Syst. Appl.*, vol. 38, no. 10, pp. 13448–13467, Sep. 2011.

[188] J. R. Quinlan, *C4.5: Program for Machine Learning*. 1993.

[189] L. Hawarah, "Une approche probabiliste pour le classement d'objets incomplètement connus dans un arbre de décision," Université Joseph Fourier - Grenoble I, 2008.

[190] M. Kurdi, "A new hybrid island model genetic algorithm for job shop scheduling problem," *Comput. Ind. Eng.*, vol. 88, no. 10, pp. 273–283, Oct. 2015.

[191] M. Habib Zahmani and B. Atmani, "A Data Mining Based Dispatching Rules Selection System for the Job Shop Scheduling Problem," *J. Adv. Manuf. Syst.*, to be published.

[192] M. G. Kharat, S. S. Khadke, R. D. Raut, S. S. Kamble, S. J. Kamble, and M. G. Kharat, "Application of Hybrid Firefly Algorithm-Tabu Search Technique to Minimize the Makespan in Job Shop Scheduling problem," *Int. J. Appl. Ind. Eng.*, vol. 3, no. 2, pp. 1–21, Jun. 2016.

[193] R. Pérez-Rodríguez, S. Jöns, A. Hernández-Aguirre, and C. Alberto-Ochoa, "Simulation optimization for a flexible jobshop scheduling problem using an estimation of distribution algorithm," *Int. J. Adv. Manuf. Technol.*, vol. 73, no. 1–4, pp. 3–21, 2014.

[194] V. Carrera-trejo, G. Sidorov, S. Miranda-jiménez, M. M. Ibarra, and R. C. Martínez, "Latent Dirichlet Allocation complement in the vector space model for Multi-Label Text Classification," *Int. J. Comb. Optim. Probl. Informatics*, vol. 6, no. 1, pp. 7–19, 2015.

[195] G. Tsoumakas and I. Katakis, "Multi-Label Classification," *Int. J. Data Warehous. Min.*, vol. 3, no. 3, pp. 1–13, Jan. 2007.

[196] A. C. P. L. F. de Carvalho and A. A. Freitas, "A Tutorial on Multi-label Classification Techniques," in *Studies in Computational Intelligence*, vol. 203, no. January, 2009, pp. 177–195.

[197] H. Modi and M. Panchal, "Experimental Comparison of Different Problem Transformation Methods for Multi-Label Classification using MEKA," *Int. J. Comput. Appl.*, vol. 59, no. 15, pp. 10–15, 2012.

[198] D. Gjorgjevikj, G. Madjarov, and S. Džeroski, "Hybrid Decision Tree Architecture Utilizing Local SVMs for Efficient Multi-Label Learning," *Int. J. Pattern Recognit. Artif. Intell.*, vol. 27, no. 7, p. 1351004, Nov. 2013.

[199] N. Mebarki and A. Shahzad, "Correlation among tardiness-based measures for scheduling using priority dispatching rules," *Int. J. Prod. Res.*, vol. 51, no. 12, pp. 3688–3697, Jun. 2013.

[200] J. Read and P. Reutemann, "MEKA: a multi-label extension to WEKA," *URL http//meka. sourceforge. net*, 2012.

[201] M. Habib Zahmani, B. Atmani, A. Bekrar, and N. Aissani, "A Real Time Data Mining Rules Selection Model for The Job Shop Scheduling Problem," in *45th International Conference on Computers & Industrial Engineering 2015 (CIE45)*, 2015, pp. 465–472.

[202] S. M. K. Hasan, R. Sarker, and D. Essam, “Evolutionary scheduling with rescheduling option for sudden machine breakdowns,” in *2010 IEEE World Congress on Computational Intelligence, WCCI 2010 - 2010 IEEE Congress on Evolutionary Computation, CEC 2010*, 2010, pp. 1–8.

[203] T. C. E. Cheng, B. Peng, and Z. Lü, “A hybrid evolutionary algorithm to solve the job shop scheduling problem,” *Ann. Oper. Res.*, vol. 242, no. 2, pp. 223–237, Jul. 2016.

[204] U. Wilensky, “NetLogo,” *Cent. Connect. Learn. Comput. Model.*, 1999.

[205] M. Habib Zahmani and B. Atmani, “Extraction of Dispatching Rules for Single Machine Total Weighted Tardiness using a Modified Genetic Algorithm and Data Mining,” *Int. J. Manuf. Res.*, vol. 13, no. 1, pp. 1–25, 2018.

[206] H. M. Soroush, “Stochastic bicriteria single machine scheduling with sequence–dependent job attributes and job–dependent learning effects,” *Eur. J. Ind. Eng.*, vol. 8, no. 4, pp. 421–456, 2014.