الجمهوريــة الجزائريــة الديمقراطيــة الشعبيــة
République Algérienne Démocratique et Populaire
وزارة التعلـيـم العـالي و البـحـث العـلمـي
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université MUSTAPHA Stambouli
Mascara
Faculté des Sciences Exactes
Département d'Informatique

جامعة مصطفى أسطمبولي
معسكر

THESE de DOCTORAT de 3ème cycle

Spécialité : Informatique

Option : Technologie de l'information et de la communication

Intitulée

Optimisation de la gestion des ressources dans les systèmes informatiques

Présentée par : AMRANE Abdelkader

Devant le jury :

Président	Dr. REBBAH Mohamed	MCA	Université M. S. de Mascara
Examinateur	Pr. BENSLIMANE Sidi Mohamed	Professeur	Ecole Sup en informatique SBA
Examinateur	Dr. BOUFERA Fatma	MCA	Université M. S. de Mascara
Encadreur	Pr. DEBBAT Fatima	Professeur	Université M. S. de Mascara
Co-encadreur	Dr. YAHYAOUI Khadidja	MCA	Université M. S. de Mascara

Année Universitaire : 2020-2021

Dédicaces

A mes parents et ma petite famille

A vous chers lecteurs

Amrane abdelkader

Remerciements

Tout d'abord, je voudrais remercier le bon Dieu de m'avoir donné le courage et les raisons du succès de ce travail.

Ma plus grande gratitude va à mes superviseurs, Mme DEBBAT Fatima et Mme YAHYAOUI Khadidja, pour leur disponibilité et le savoir-faire dont j'ai pu bénéficier lors de nombreuses discussions.

Mes grands remerciements à Mr REBBAH Mohamed, Maître de conférences à l'Université de Mascara pour l'honneur qu'il m'a fait en acceptant de présider le jury de ma thèse, à Mr BENSLIMANE Sidi Mohammed, Professeur à L'école supérieur en informatique de Sidi Bel Abbes et à Madame BOUFERA Fatma, maître de conférences à l'Université de Mascara pour avoir accepté d'examiner ce travail et consacré leur temps précieux.

Mes sincères remerciements à ma famille et à tous ceux qui m'ont aidé dans ce modeste travail.

Résumé

L'industrie manufacturière est l'un des éléments importants de la durabilité de la société moderne. La complexité des produits augmente tandis que leur cycle de vie se raccourcit. Les systèmes de fabrication traditionnels et leurs approches de gestion associées doivent être améliorés pour s'adapter aux changements susmentionnés.

L'efficacité des systèmes est devenue un paramètre essentiel des processus de fabrication industriel et cela avec le fait de prendre en compte les problèmes de planification dans le monde réel avec une gestion optimisée des tâches et des ressources. Nous proposons un modèle d'ordonnancement d'atelier de Type Job-Shop en tenant compte les contraintes adaptées aux nouvelles exigences de l'industrie moderne. Comme le problème est fortement NP-difficile, un algorithme génétique hybride parallèle est développé afin d'obtenir un plan d'ordonnancement optimisé avec un meilleur temps de réponse adapté à l'environnement dynamique et aux différentes contraintes, la méthode proposée est conçue pour être hautement cohérente avec la plateforme NVIDIA CUDA. Des expériences numériques sont menées et montrent que l'approche permet non seulement de résoudre le problème de manière flexible, mais également d'obtenir des résultats compétitifs et de réduire considérablement les délais.

Mots clés : Systèmes manufacturiers, Ordonnancement, Planification, Job-Shop, Algorithme génétique cellulaire, Parallélisme, GPGPU, CUDA

Abstract

Manufacturing systems are one of the most important elements in the sustainability of modern society. The complexity of products increases while their life cycle is shortened. Traditional manufacturing systems and their associated management approaches need to be improved to accommodate the aforementioned changes.

The efficiency of systems has become an essential parameter of industrial manufacturing processes along with taking into account planning problems in the real world with an optimized management of tasks and resources. We propose a Job-Shop type-scheduling model, taking into account the constraints adapted to the new requirements of modern industry. As the problem is strongly NP-hard, a parallel hybrid genetic algorithm is developed in order to obtain an optimized scheduling with a better response time adapted to the dynamic environment and to different constraints; the proposed method is designed to be highly consistent with the NVIDIA CUDA platform. Numerical experiments that are carried out show that the approach not only allows to solve the problem in a flexible way, but also to achieve competitive results and significantly reduce the overall duration.

Keywords:

Manufacturing systems, Scheduling, Planning, Job-Shop, Cellular genetic algorithm, Parallelism, GPGPU, CUDA.

ملخص

التصنيع هو أحد العناصر المهمة في استدامة المجتمع الحديث. بينما يزداد تعقيد المنتجات تتقلص دورة حياتها. تحتاج أنظمة التصنيع التقليدية ومناهج ادارتها إلى تحسين دوري لاستيعاب التغييرات والتحولات في هذا المجال.

أصبحت كفاءة الأنظمة حجر زاوية لعمليات التصنيع وهذا مع مراعاة مشاكل التخطيط في العالم الحقيقي من خلال إدارة محسّنة للمهام والموارد. نقترح نموذج جدولة من نوع –Job-Shop-مع الأخذ بعين الاعتبار التكيف مع المتطلبات الجديدة للصناعة الحديثة. نظرًا لأن المشكلة صعبة للغاية، فقد تم تطوير خوارزمية جينية هجينة متوازية من أجل الحصول على جدول زمني محسن مع وقت استجابة أفضل يتكيف مع البيئة الديناميكية والقيود المختلفة، تم تصميم الطريقة المقترحة لتكون متوافقة للغاية مع منصة -NVIDIA CUDA-. بعد إجراء تجارب عديدة تبين أن التصميم المنتهج لا يسمح فقط بحل المشكلة بطريقة مرنة، ولكن أيضًا لتحقيق نتائج تنافسية وتقليل المدة الزمنية بشكل كبير.

الكلمات الدالة:

أنظمة التصنيع، الجدولة، التخطيط، Job-Shop، الخوارزمية الجينية، التوازي، GPGPU، CUDA

Table des matières

Liste des figures

Liste des tableaux

I. Chapitre I Introduction générale

I.1 Avant-propos

Les concepts des systèmes de fabrication à travers l'histoire moderne étaient en étroite relation avec les principes et les découvertes de la science. Les futurs concepts de fabrication devront être adaptés aux besoins de la société moderne et, plus particulièrement, à l'écosystème plus que jamais. Le processus de production des entreprises de fabrication a toujours été un facteur clé pour la réussite commerciale globale. Les problèmes de planification de la production sont confrontés à des milliers d'entreprises dans le monde entier qui sont engagés dans la production de biens tangibles. Les conditions préalables à une participation réussie aux réseaux de production dynamiques et globaux exigent que les processus de production, les ressources, les structures des usines, les schémas des systèmes de fabrication ainsi que leurs concepts logistiques et organisationnels soient adaptables rapidement et avec peu d'effort. Cette capacité est nécessaire pour les entreprises de production pour résister aux changements continus et de l'environnement de fabrication turbulente en face d'eux, et peuvent être décrits comme «mutabilité».

Ainsi, il n'est pas sans raison que la résolution efficiente et efficace des problèmes de planification de la production ait attiré l'intérêt de nombreux praticiens et chercheurs des deux domaines de la gestion de la production et de l'optimisation combinatoire. Cet intérêt est encore renforcé par la similitude des problèmes d'ordonnancement de la production avec les problèmes qui se posent dans d'autres domaines scientifiques et donc l'applicabilité des méthodes développées. Une grande variété de problèmes d'ordonnancement a été identifiée et une gamme encore plus large de méthodologies de solutions a été proposée. Au tout début, la recherche portait sur des méthodes exactes, c'est-à-dire des méthodes qui garantissaient la solution optimale d'un problème donné.

En raison du manque de ressources informatiques et de la nécessité de résoudre des problèmes d'ordonnancement à grande échelle, on a rapidement constaté que les méthodes exactes étaient peu pratiques et que, par conséquent, la recherche portait sur des heuristiques spécifiques aux problèmes. D'autre part, le besoin de solutions plus robustes a conduit au développement et à l'application des premières méta-heuristiques dont la performance s'améliore constamment. Les méthodologies contemporaines combinent généralement plusieurs algorithmes heuristiques et méta-heuristiques (algorithmes hybrides) dans le but de surmonter les limitations inhérentes aux composants méta-heuristiques simples.

Pour cette raison, dans les systèmes de fabrication actuels, en particulier les approches déterministes sont utilisées pour la synchronisation des flux, d'énergie et d'informations. Cela signifie que les méthodes basées sur les résultats mathématiques exacts et les règles de logique sont utilisées pour la modélisation et le fonctionnement des systèmes. La production est un processus très dynamique avec de nombreux événements inattendus et de nouvelles exigences en constante évolution ou la description exacte d'un système n'est souvent pas possible par les méthodes conventionnelles. Cependant, dans de nombreux domaines différents de la science et de la technologie, il a été récemment possible de remarquer le passage à la conception de

systèmes intégrés capables d'apprendre et de répondre efficacement à la complexité croissante, à l'imprévisibilité et à la variabilité de l'environnement.

Les problèmes d'aujourd'hui sont plus intéressants et complexes. La complexité de l'optimisation, le nombre d'axes de recherche et le niveau d'optimalité sont des dimensions dans lesquelles les solutions font des pas de géant. Les progrès technologiques logiciels et matériels sont en partie responsables de ces sauts, les processeurs, la mémoire et les supports d'interconnexion sont tous devenus plus rapides, mais en particulier les processeurs graphiques, ou GPU (Graphics Processing Unit), également appelés coprocesseurs graphiques sur certains systèmes qui subissent des changements importants et deviennent plus puissants à un rythme plus rapide que la loi de Moore. Ces changements incluent l'incorporation d'un traitement à fonction fixe sur les pixels, puis généralisant ceux-ci pour être entièrement programmables. De plus, l'exploitation d'infrastructures légères et polyvalentes telles que les GPU en raison de leur faible coût et de leur disponibilité omniprésente, ramène au premier plan des architectures parallèles complexes et offre l'opportunité de développer des axes de recherche d'algorithmes d'optimisation dédiés.

I.2 Contexte et Problématique

Les systèmes de fabrication ont évolué au fil des années en réponse à de nombreux facteurs externes, notamment l'introduction de nouvelles technologies et de nouveaux matériaux de fabrication, l'évolution constante de nouveaux produits et l'accent mis sur la qualité ainsi que la concurrence mondiale croissante et le besoin pressant de réactivité, adaptabilité. En interne, le désir de réduire les déchets et d'accroître l'efficacité et la productivité tout en créant / préservant / augmentant des emplois à haute valeur ajoutée avec une implication humaine significative est un défi identique. Plusieurs paradigmes de systèmes de fabrication sont apparus au fil des années en raison de ces facteurs et contre une baisse de volatilité dans les demandes du marché, en changeant les préférences des clients et le besoin de plus de différenciation et de personnalisation des produits. En outre, les chaînes d'approvisionnement mondiales et distribuées et les pressions accrues sur les questions de main-d'œuvre, la concurrence des pays en développement et les fluctuations des devises augmentent la pression. Les paradigmes des systèmes de fabrication récemment introduits, tels que la fabrication souple et reconfigurable, répondent à ces besoins de différentes façons. Ils peuvent être considérés comme des catalyseurs du changement et de la transformation à différents niveaux. Fabrication flexible permet de modifier les horaires individuels des opérations, des processus, des pièces de routage et de production. Cela correspond à des variations de produits dans un périmètre prédéfini d'une famille de pièces. Il permet également d'ajuster la capacité de production dans les limites du système existant. Par conséquent, les systèmes de fabrication flexibles (FMS) offre une flexibilité généralisée qui permet des changements et l'adaptation des processus et des volumes de production, dans les limites prédéfinies, sans modifier physiquement le système de fabrication lui-même. Fabrication reconfigurable (RMS) (Figure I.1) permet la fonctionnalité changeante et une capacité évolutive en modifiant physiquement les composants du système par l'ajout, la suppression ou la modification de modules de machines, des machines, des cellules, des unités de matière de manipulation et / ou des lignes complètes. Ainsi, RMS répond

aux changements en offrant flexibilité ciblée sur la demande en reconfigurant physiquement le système de fabrication. La reconfiguration matérielle nécessite également des changements majeurs dans le logiciel utilisé pour contrôler des machines individuelles, des cellules et des systèmes complets, ainsi que pour planifier et contrôler les processus individuels et la production globale.

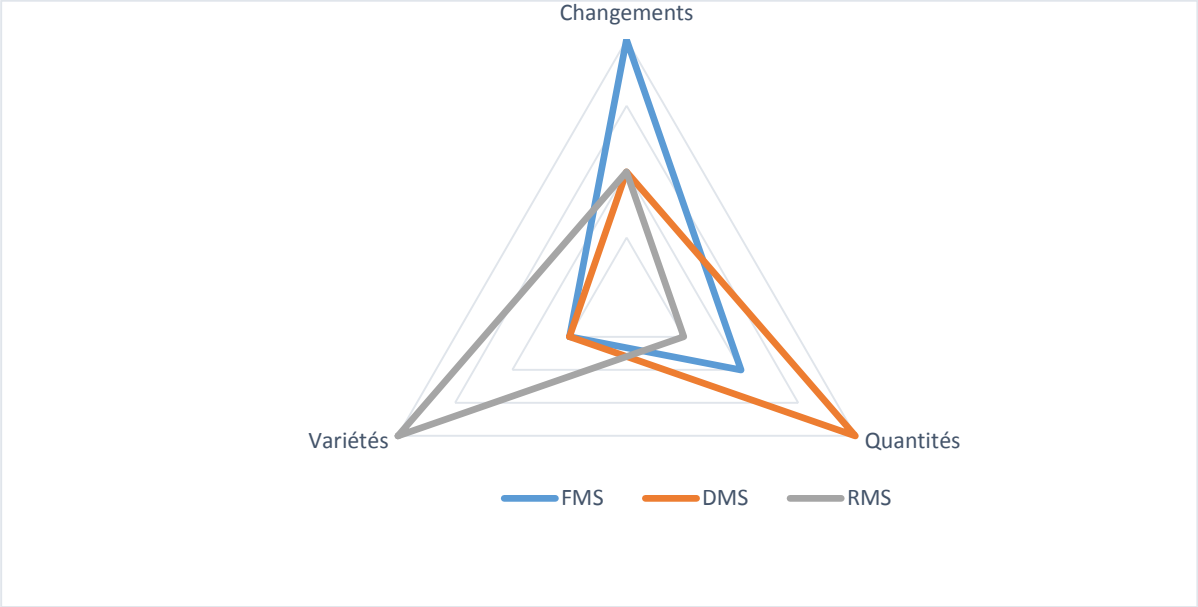


Figure I.1 Comparaison entre les différents systèmes de fabrication

La réduction significative du temps de développement des produits et l'introduction plus rapide de modèles et de variations de nouveaux produits n'ont pas été parallèles dans le domaine de la conception et du développement des systèmes de fabrication. Ces systèmes de fabrication doivent être conçus pour satisfaire certaines exigences et contraintes de produits qui varient au fil du temps et le rythme de ces changements s'est accéléré récemment qui commence avec la conception et la synthèse initiales du système en fonction des objectifs et des contraintes spécifiés suivis par la modélisation, l'analyse et la simulation, puis la conception finale est réalisée, mise en œuvre et utilisée dans la production. Le système de fabrication subit une refonte et une reconfiguration, tout au long de son exploitation et à mesure que de nouveaux besoins émergent et des changements sont nécessaires, visant à répondre aux exigences de l'environnement changé. Par conséquent, la reconfiguration logicielle et matérielle (ElMaraghy, 2005) et la flexibilité sont en fait des facteurs de changement et peuvent étendre l'utilité, la facilité d'utilisation et la durée de vie des systèmes de fabrication.

De plus, on a observé que le cycle de vie d'un produit était approximativement dans la même plage de vie que les machines-outils et les procédés technologiques. Avec la durée de vie toujours croissante, les processus installés et l'équipement devraient produire la ou les prochaines générations de produit. Cependant, à l'heure actuelle, la livraison fiable de produits personnalisés sur des marchés distribués à l'échelle mondiale revêt la plus haute priorité. Cette priorité détermine de plus en plus le développement des produits, des procédés et des installations de production, y compris la logistique, selon les lignes directrices suivantes :

• La conception des processus et des chaînes d'approvisionnement doit être réalisée en tenant compte principalement des besoins du marché mondialement répartis.

• Des unités de production adaptables, et si nécessaire temporaires, sont requises au lieu d'usines centralisées avec une gamme verticale de fabrication élevée.

• La logistique de production est régie par la logistique d'approvisionnement et de distribution.

• Les systèmes de planification et de commande de la même usine doivent maîtriser différents types d'ordres.

• Les structures des produits doivent être adaptées aux exigences changeantes d'une production distribuée à l'échelle internationale.

• Les méthodes de production et d'assemblage doivent tenir compte des conditions locales et globales.

La concurrence dans le domaine de la construction automobile oblige les usines a souligné que tout client peut faire peindre une voiture dans la couleur de son choix ou l'industrie à sa deuxième révolution (également connue sous le nom d'Industrie 2.0) n'a pas été en mesure de prendre en charge les niveaux de personnalisation que recherchaient les clients en raison de son manque de flexibilité. Lors de la troisième révolution industrielle à la fin des années 80, également appelée Industrie 3.0, la demande des clients pour une grande variété de produits a conduit au développement de la personnalisation de masse. Il était basé sur le développement de l'information, de la technologie d'automatisation et de l'ordinateur. Cela a conduit à des équipements avancés tels que des machines à commande numérique par ordinateur, des véhicules à guidage automatique et des systèmes de manutention comme des robots qui ont été intégrés à des ordinateurs et assimilés dans un système de fabrication (Figure I.2). La nouvelle génération des systèmes manufacturiers représentée par l'industrie 4.0 offre des plans de processus complexes et un degré élevé de partage des ressources. Par conséquent, allouer le nombre énorme de tâches aux ressources et les ordonnancer tout en respectant plusieurs contraintes des différents plans de traitement des tâches et de la disponibilité des machines. Selon la taille du problème (représentée par le nombre de tâches et de ressources) la tâche d'ordonnancement peut s'avérer être l'activité la plus longue et la plus difficile au sein d'une entreprise. Une planification inefficace peut entraîner une utilisation réduite des ressources, une capacité surchargée / inactive, des délais de production longs et des engagements de date d'échéance peu fiables, ce qui augmente les coûts de production et réduit la compétitivité sur le marché. En outre, une planification inefficace retarde souvent les commandes et entraîne des clients insatisfaits et peut exposer l'entreprise à des pénalités.

La planification implique l'allocation de ressources, telles que des machines et des transporteurs, à des tâches de production dans une séquence temporelle pour produire des pièces. La plupart des problèmes d'ordonnancement présentés qui doivent prendre en compte ces formulations dans un cadre assez général et sont donc NP-difficiles. Les objectifs globaux du système sont de maximiser l'utilisation des ressources et de minimiser le temps de production tout en respectant une variété de contraintes du système, telles que la précédence entre les tâches de production.

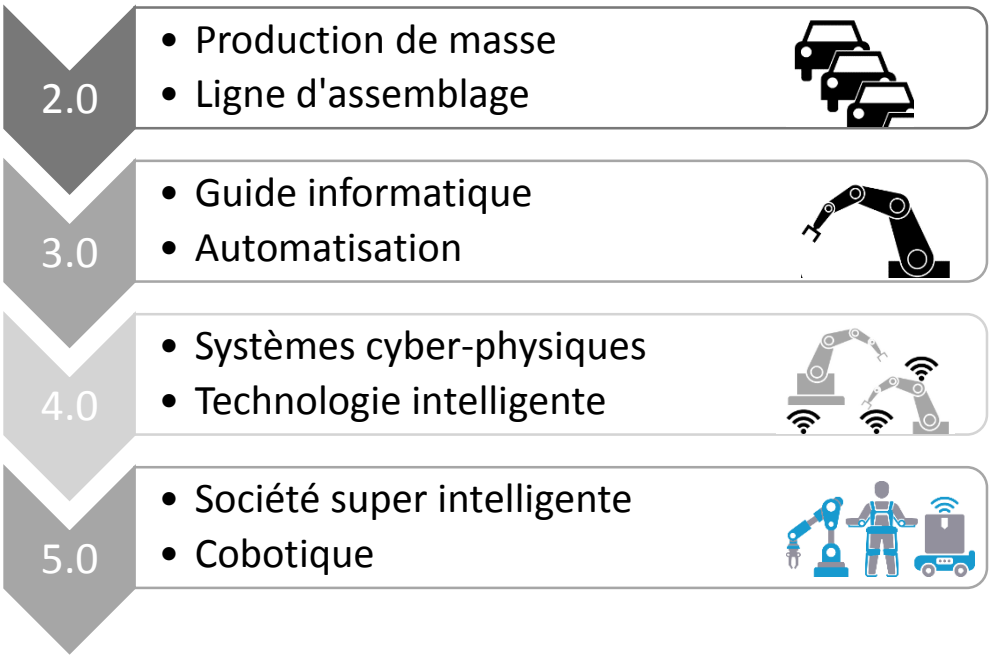


Figure I.2 La révolution industrielle

I.3 Contribution

L'optimisation des systèmes de production est utilisée à tous les niveaux (stratégique, tactique, opérationnel), et pour chaque niveau, l'objectif et les contraintes doivent être déclarés. Au niveau stratégique, les objectifs de production sont définis dans le but de soutenir les décisions telles que la sélection des produits et des procédés, la construction de nouvelles usines et l'acquisition de ressources externes. Au niveau tactique, les objectifs sont définis afin de soutenir les décisions telles que la planification de la production, la gestion des ressources et l'identification des procédures. Les objectifs opérationnels sont strictement liés aux décisions de gestion des flux (Figure I.3), et de contrôle des opérations et de contrôle qualité. De nombreuses entreprises investissent massivement pour déployer un plan d'optimisation qui couvre les trois domaines afin de réduire les coûts et de fonctionner plus efficacement, et pour cela l'application doit se concentrer sur des infrastructures généralement plus spécialisées dans leur production comme les ateliers de type Job-Shop, où le temps passé à trouver des solutions optimisées est un facteur déterminant dans le fonctionnement de l'entreprise

Le problème d'ordonnancement Job-Shop (JSP : Job Shop Problem) consiste à planifier des tâches dans un environnement avec des machines. Chaque tâche est formée de plusieurs opérations avec un ordre de priorité linéaire où chaque opération peut être traitée par une seule machine particulière. Le problème d'ordonnancement Job-Shop est connu pour être fortement NP-difficile .Le temps de traitement de chaque opération sur chaque machine est connu et aucune préemption n'est autorisée. Le problème est de décider dans quel ordre les opérations seront traitées sur chaque machine.

Le but de cette étude est d'améliorer l'efficacité de la production en planifiant un calendrier reconfigurable de production afin qu'ils puissent tirer autant d'avantages que

possible. L'efficacité de la production peut être représentée par une fonction objectif, sous les contraintes du système de fabrication.

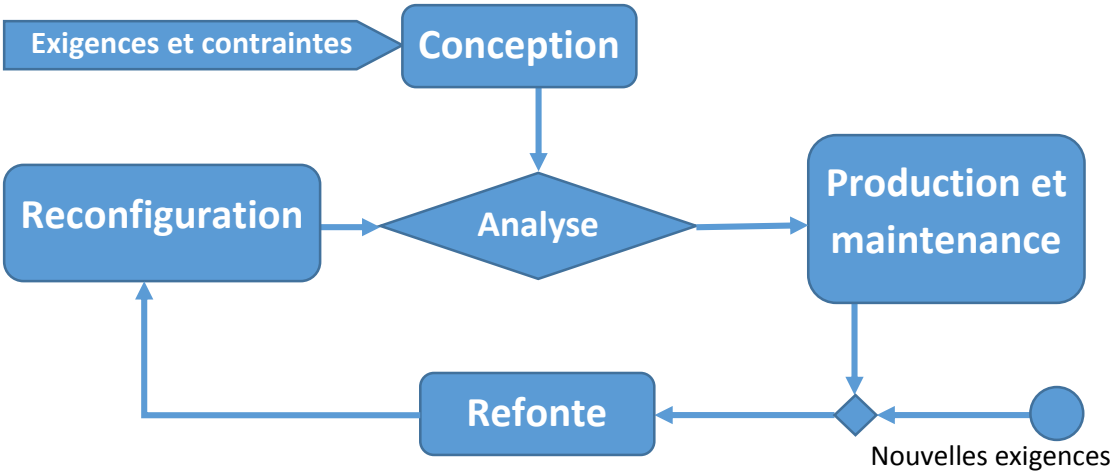


Figure I.3 Cycle de vie des systèmes de fabrication

L'objectif principal est de minimiser la durée totale (c'est-à-dire le temps qui s'écoule du début des travaux à la fin), car de cette manière, les systèmes peuvent réduire le coût, le temps et la consommation des ressources (Cela se traduit par une consommation d'électricité réduite) tout en augmentant la capacité de production et la qualité des produits.

Afin d'atteindre les objectifs, cette étude aborde les questions suivantes :

1. Comment définir et formuler un modèle traitable ?

2. Comment le résoudre avec plusieurs contraintes complexes et des tailles importantes des instances ?

3. Comment formuler le problème pour une infrastructure légère et parallèle ?

5. Quelle sont les limites taille des instances pour lesquelles l'optimalité peut être prouvée, et quelle est la qualité de la solution ?

6. Comment orienter calcul l'encodage/décodage ?

7. Comment construire des solutions réalisables ?

8. Comment exploiter l'infrastructure parallèle des processeurs graphiques pour permettre l'applicabilité des méta-heuristiques parallèles basées sur le voisinage ?

9. Comment se comporte la méthode lorsqu'on limite le nombre de générations contre un temps réponse réduit ?

Pour répondre à ces problématiques, nous proposons un AG parallèle hybride implémenté sur la plateforme NVIDIA CUDA, prenant en compte une codification orientée calcul qui donne toujours des solutions réalisables susceptibles d'être réparées après application d'opérateurs génétiques. De plus, deux niveaux de représentation sont proposés, l'un pour exploiter les performances de l'architecture et l'autre pour contourner la faiblesse en termes de partage de ressources et de phases de synchronisation.

I.4 Plan de la thèse

Le reste de ce manuscrit est structuré comme suit :

Dans le deuxième chapitre, les changements de paradigme de fabrication de produits survenus au cours des dernières années sont expliqués. Ces discussions indiquent également des directions prometteuses pour le développement futur des caractéristiques de fabrication des produits, en particulier les concepts, les méthodologies et les technologies qui permettent une fabrication de produits plus préférable expliquant les principales technologies d'optimisation et les critères pour juger de la pertinence et de la qualité des facteurs considérés. Ensuite un aperçu sur l'état actuel des technologies d'optimisation pour la conception de produits et une présentation des méthodologies et stratégies fondamentales, y compris les optimisations de conception de système et les méthodologies de base pour la prise de décision dans la conception et la fabrication de produits.

Le troisième chapitre traite le problème de planification complexe de type Job-Shop et des méthodes pour le résoudre, commençons par une description des modèles d'ordonnancement de base avec des applications et une introduction à l'optimisation discrète (couvrant la complexité et les méthodes générales d'optimisation).Nous donnons une formulation du problème du Job Shop en termes de représentation graphique. Ensuite, nous passons aux techniques de génération de planification, qui construisent des solutions réalisables. Tout d'abord, différentes manières de représenter un calendrier d'adaptation génétique sont discutées. Sur la base de cette notion, plusieurs perspectives d'adaptation génétique au problème du Job Shop sont évaluées. Des méthodes heuristiques pour résoudre ces problèmes d'ordonnancement complexes sont présentées. Nous achèverons ce chapitre par le passage vers les méta-heuristiques parallèles qui sera l'objet du chapitre suivant.

Le chapitre quatre vise à mieux caractériser les algorithmes génétiques parallèles (PGA) en tant qu'outil puissant d'optimisation, de recherche et d'apprentissage. Nous vous proposons ici deux niveau de présentation. Le premier est ciblé sur les algorithmes eux-mêmes, discutant de leurs composants, du parallélisme physique et des meilleures pratiques pour les utiliser et les évaluer ainsi que le traitement des résultats théoriques pertinents pour la recherche avec les PGA. Une deuxième partie propose une étude des PGA en tant que solutionneur de problème d'ordonnancement, ici, nous soulignons les solutions pour le problème Job-Shop abordant la conception, la codification et la parallélisassions des différentes phases avec une analyse diversifiée pour montrer le grand succès de ces techniques.

Le dernier chapitre présente une description du problème abordé, à savoir l'optimisation sur une plateforme haute performance basée sur des architectures GPU. De plus, les principaux objectifs du travail sont présentés et les concepts fondamentaux et essentiels pour comprendre le travail proposé, tels que les AG parallèles et les GPU. Et la dernière partie traite des résultats expérimentaux obtenus pour chaque solution et les compare à l'implémentation CPU et à d'autres implémentations dans la littérature. Les solutions sont évaluées en fonction de deux métriques, l'accélération et le temps d'écoulement maximum pour des séquences de tâches.

II. Chapitre II : Gestion des ressources : Ordonnancement et optimisation

II.1 Introduction

Dans le domaine de la production industrielle, les tendances actuelles indiquent que les systèmes manufacturiers performants doivent s'adapter rapidement aux fluctuations du marché (demandes aléatoires) et aux perturbations internes (pannes des machines, qualité des produits). Ces exigences industrielles imposées par le marché, la concurrence, la qualité ainsi que la densité et la diversité des produits traités entraînent une complexité sans cesse croissante des systèmes de production. L'objectif associé à ce type de systèmes est alors d'assurer un traitement le plus varié possible avec un maximum de productivité, au moindre coût et dans les délais.

Notre objectif au cours de ce chapitre est de poser le contexte de notre recherche où les systèmes manufacturiers y sont présentés ainsi que leurs caractéristiques principales. Nous commencerons ce chapitre par un survol de la littérature spécifique sur les Systèmes de Production. Nous mettons d'abord l'accent sur les différentes parties d'un Système de production, leurs objectifs ainsi que leurs caractéristiques. Ensuite, nous discutons le problème de l'ordonnancement des tâches avec optimisation de la gestion des ressources dans les systèmes de production.

II.2 Problèmes d'optimisation : définition et formulation

Le domaine de l'optimisation a fait l'objet d'une attention considérable ces dernières années, principalement en raison des progrès rapides de la technologie informatique, notamment le développement et la disponibilité de logiciels conviviaux et de processeurs parallèles et à grande vitesse. Une conception optimale est celle qui donne la meilleure valeur objectif possible, tout en satisfaisant à toutes les contraintes du problème. Problème d'optimisation

II.2.1 Formulation

Dans un problème d'optimisation, une fonction doit être maximisée ou minimisée. La fonction qui est optimisée est appelée fonction objectif ou indice de performance. La fonction est une quantité telle que le coût, le profit, l'efficacité, la taille, la forme, le poids, la production, etc. Il va sans dire que la minimisation des coûts ou la maximisation des profits sont des considérations primordiales pour la plupart des organisations. Les variables de la fonction objectif sont appelées variables de conception ou variables de décision. En général, il peut s'agir des dimensions d'une structure ou de ses attributs matériels, pour un problème d'optimisation de la structure. D'un point de vue pratique, les variables de conception peuvent prendre des valeurs comprises entre une limite inférieure et une limite supérieure uniquement.

Le problème d'optimisation peut être formulé mathématiquement selon (Belegundu & Chandrupatla 2019) comme suit :

m(x) : fonction à minimiser ***II.1***

Avec :

- $e_j(x) = 0 \quad j = 1, 2, \ldots, r < n$: contraintes d'égalité
- $n_i(x) <= 0 \quad i = 1, 2, \ldots, m < n$: contraintes d'inégalité
- (x) : vecteur de variables de conception de n dimension $[x_1 \ldots x_n]$

II.3 Optimisation avec et sans contraintes

Les problèmes liés à la forme générale peuvent être classés selon la nature de la fonction objectif et des contraintes (linéaire, non linéaire, convexe), le nombre de variables (grandes ou petites), la fluidité des fonctions (différentiables ou non différentiables), etc. La distinction la plus importante est sans doute celle entre les problèmes qui ont des contraintes sur les variables et ceux qui n'en ont pas (Kuhn 2013 ; Dobbs & Nelson 1976). Les problèmes d'optimisation sans contraintes se posent directement dans de nombreuses applications pratiques. S'il existe des contraintes naturelles sur les variables, on peut parfois les ignorer et supposer qu'elles n'ont aucun effet sur la solution optimale. Les problèmes sans contraints se présentent également sous la forme de reformulations de problèmes d'optimisation avec contraints, dans lesquelles les contraintes sont remplacées par des restrictions dans la fonction objectif.

Les problèmes d'optimisation avec contraints proviennent de modèles qui incluent des contraintes explicites sur les variables.

La majorité des problèmes d'ingénierie impliquent une minimisation sous contraintes, c'est-à-dire que la tâche consiste à minimiser une fonction soumise à des contraintes.

Le problème de l'optimisation sous contrainte peut être mathématiquement formulé comme suit :

Minimiser $f(x)$ ***II.2***

Soumis à

$$\begin{cases} g_i(\boldsymbol{x}) \leq 0 & i = 1, 2, \ldots, m < n \\ h_j(\boldsymbol{x}) = 0 & j = 1, 2, \ldots, r < n \end{cases}$$

où $x = (x_1, x_2, \ldots, x_n)^T$ est un vecteur n variables réelles.

(f) est la fonction objectif

(g_s) sont des contraintes d'inégalité et

(h_s) sont les contraintes d'égalité.

II.4 L'optimisation des systèmes manufacturiers

Les objectifs principaux de la fabrication de produits avancés sont de développer et de fabriquer des produits essentiels qui répondent aux besoins de style de vie au plus haut degré possible, ainsi que des produits auxiliaires qui rendent notre vie plus confortable, efficace et satisfaisante. La fabrication de tous les produits dépend de différents niveaux de technologies.

Ces exigences impliquent un changement dans le paradigme de fabrication des produits, passant d'une «vente de produits fabriqués» à une «production de produits», en mettant l'accent sur la satisfaction des clients. Une méthode de production qui pourrait faire face à de tels problèmes était la méthode de production dite juste à temps (Just In Time (JIT)) (Yoshimura &

Fujimi 2003) dans laquelle les quantités requises de pièces sont fournies à chaque processus exactement au moment voulu, une méthode de production qui peut être appliquée efficacement à la production en atelier. Dans la seconde moitié des années 90, une méthode de production basée sur le paradigme CALS (Yoshimura & Fujimi 2003) (Continuous Acquisition & Lifecycle Support) a été développée, où toutes les données relatives à l'ensemble du cycle de vie d'un produit sont traitées par des ordinateurs afin que l'échange de données en temps réel et la prise de décision devient possible, facilitée par une technologie de l'information de plus en plus puissante. La signification originale de CALS a été remplacée par la signification actuelle, Commerce at Light Speed. À l'heure actuelle, l'ère de la fabrication en atelier, dans laquelle les clients sélectionnent les produits les plus préférables parmi une variété de produits que les fabricants préparent à l'avance, cède la place à un paradigme de fabrication qui prend en charge la fabrication de produits sur commande, où les besoins spécifiques et détaillés des clients et les désirs peuvent être traités économiquement à un niveau relativement fin. En outre, nous pouvons envisager la possibilité que la fabrication à l'avenir intègre finalement une certaine forme de participation des clients dans le processus de conception des produits, afin de maximiser la satisfaction des clients, permettant des scénarios de fabrication collaborative client-fabricant dans une certaine mesure.

II.4.1 Éléments d'évaluation et critères de production

Pour obtenir des solutions optimales de conception et de fabrication de produits, des éléments d'évaluation et des critères de fabrication des produits doivent d'abord être définis :

A. Qualité, coût et performance

Les performances, les caractéristiques requises et les qualités (peuvent être classées en deux types : les qualités de conception et les qualités de fabrication) sont objectif de la fabrication. Les qualités de conception correspondent aux valeurs que les clients exigent pour le produit et, dans le cas de machines industrielles telles que les machines-outils et les robots industriels, ce sont les précisions, les rendements, les besoins énergétiques opérationnels et les aspects de performance similaires. Dans le cas des automobiles, la conduite, les performances d'accélération et de freinage, l'économie de carburant, la résistance aux chocs, le confort, la polyvalence, l'esthétique, etc., seraient pris en compte. D'autre part, les qualités de fabrication se rapportent aux processus de fabrication utilisés lors de la production de produits qui incorporent les qualités de conception souhaitées. Dans le cas des machines-outils, ces qualités correspondraient à des variations dimensionnelles, à la rugosité de surface dans les zones de contact des joints, à la précision de traitement, etc. Pour assurer un niveau satisfaisant de qualité des produits, les fabricants doivent évaluer si leurs produits répondent ou non aux spécifications de conception désignées. Ici, les variations relatives aux procédés de fabrication sont les principaux facteurs d'évaluation. Les qualités dont les clients ont besoin dans les produits qu'ils achètent sont souvent considérées comme des aspects de la performance des produits. Par exemple, les précisions, lorsqu'elles sont considérées comme une performance du produit, correspondent à certains niveaux de précision lorsque le produit est utilisé pour le travail ou pour atteindre son objectif. Les efficacités sont souvent évaluées par le temps requis pour terminer une tâche ou une séquence d'opérations et un produit qui peut accomplir un travail

plus rapidement est considéré comme ayant une efficacité plus élevée. Le prochain critère important dans la conception d'un produit est le coût total de fabrication, la somme des différents coûts nécessaires à la fabrication effective du produit. Le coût des matériaux des éléments de structure et des composants, les coûts d'usinage, les coûts d'assemblage, etc. Ces coûts peuvent être clairement évalués comme des grandeurs quantitatives et, dans certains cas, les formulations d'optimisation peuvent bénéficier de la transformation de certains autres critères, tels que le temps, en coûts.

B. Éléments relatifs à la méthode de production

Il existe deux principales méthodes de production, la production sur stock et la production sur commande. Dans la production sur stock, une quantité spécifiée du produit final est maintenue dans chaque point de vente en tant qu'inventaire local, et les fabricants reconstituent cet inventaire au besoin. En revanche, dans la production sur commande, la fabrication ne commence qu'après réception des commandes de produits. Pour réduire les délais de production aux scénarios de commande, certains fabricants maintiennent des stocks de pièces à certaines étapes de la fabrication, et ces articles sont préparés en fonction de la production interne aux besoins en stock.

La plupart des produits nécessitent de nombreuses étapes de fabrication et dépendent d'une variété de processus. Cependant, à moins que les lignes de production ne fonctionnent correctement, des pénuries de certaines pièces nécessaires à un processus peuvent se produire, entraînant une accumulation de pièces qui ont été traitées à un stade antérieur mais qui ne peuvent pas être transférées au processus suivant.

La méthode de production JIT est également appelée méthode de fabrication Toyota. Des fiches d'instructions appelées «kanban» sont utilisées pour délimiter explicitement les itinéraires que les pièces doivent suivre, des processus en amont aux étapes suivantes, et pour fournir des instructions de fabrication applicables aux processus en cours. Ces méthodes de production visent à minimiser les pertes de fabrication, tant en termes de matériel que de temps, en évitant les dépassements ou les pénuries de certains produits manufacturés, en évitant les goulots d'étranglement où certaines pièces sont bloquées en attendant l'arrivée d'autres pièces, et minimiser le temps de production global(Makespan) . La minimisation de ces diverses pertes et la maximisation de l'efficacité sont des critères importants dans la fabrication de produits compétitifs. Lorsque les systèmes de gestion de la production sont classés du point de vue de l'information et du flux de produits, il existe deux types principaux : les méthodes de fabrication push et les méthodes de fabrication pull. La fabrication push est un type de système de premier plan, où les fonctions de gestion de la production sont centralisées et des instructions sont distribuées à chaque processus de fabrication. Dans de tels scénarios, les quantités de production sont d'abord déterminées, puis une taille de lot est calculée avant la fabrication. Les quantités de commande acceptables, les procédures de production et les détails de la gestion de la production sont pris en compte sous les contraintes de disponibilité de l'équipement et de capacité de production. Ce système de production est principalement utilisé dans la production de masse. La fabrication pull, en revanche, fonctionne sous un paradigme de pointe de la demande. Les procédures de production et les détails de la gestion de la production sont adaptés aux produits spécifiques pour lesquels les demandes sont connues, puis fabriqués le plus

efficacement possible. Dans tels scénarios, les problèmes tels que les incohérences entre les taux de production des équipements et les stocks en cours de fabrication, et les pertes de productivité dues à des temps de configuration plus longs doivent être évités. Pour éviter ces écueils et réaliser une fabrication efficace, de nouvelles technologies de fabrication sont nécessaires, telles que la fabrication de différents types de produits en utilisant le même équipement, ainsi qu'une approche qui unifie la manipulation des matériaux et des informations. JIT est une implémentation de la fabrication pull qui convient souvent aux scénarios de fabrication en atelier.

II.4.2 Flexibilité dans la fabrication

Le besoin de flexibilité de fabrication est le résultat du changement de paradigme de la production de masse aux opérations Job-Shop, où les marchandises sont fabriquées essentiellement sur commande, plutôt que d'être fabriquées à l'avance. Ainsi, la flexibilité des équipements de fabrication est de plus en plus importante. Un système de production où des machines polyvalentes sont connectées à des machines de transport est appelé un système de fabrication flexible (FMS) (Butt 2020) (FMS). Les composants FMS sont des machines flexibles telles que des équipements NC (Numerical Control), des centres d'usinage, des machines de transport, des systèmes de contrôle informatique, etc. La Cellule de Fabrication Flexible (FMC) est une version à plus petite échelle du FMS. La figure II.1 montre un exemple FMC. L'équipement complet d'un FMS nécessite généralement un investissement financier important et ne peut pas nécessairement faire face à des changements dans les situations de fabrication. Dans de tels cas, des combinaisons de FMC modifiées de manière flexible peuvent souvent être utilisées. Dans les situations où une fabrication flexible est requise, des équipements de fabrication qui peuvent être facilement reconfigurés pour une large gamme de quantités fabriquées sont nécessaires pour faire face aux niveaux fluctuants de la demande et à une large gamme de produits et de volumes. Pour répondre au besoin d'un tel équipement, des machines composites ont été développées dans lesquelles plusieurs processus sont intégrés de sorte que divers travaux ou opérations peuvent être effectués par une seule machine.

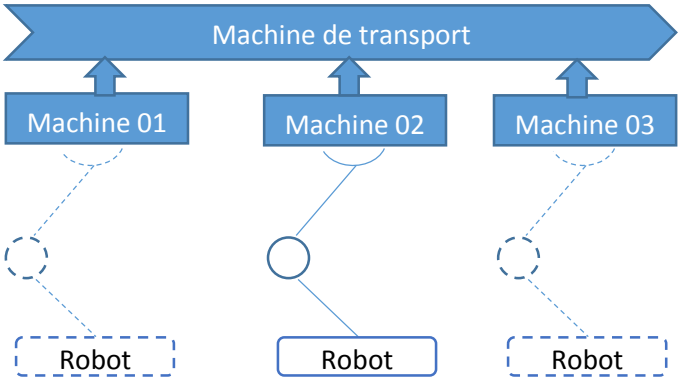


Figure II.1 FMC composé d'un robot industriel et de machines

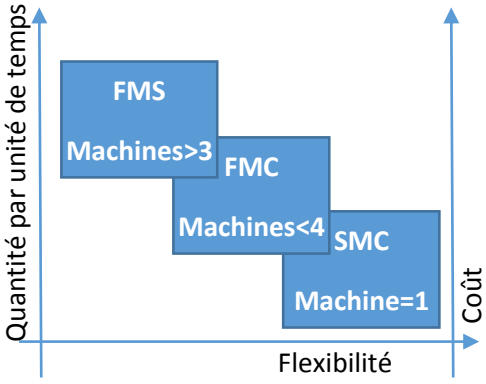


Figure II.2 Diagramme conceptuel

La figure II.2 montre un diagramme conceptuel de la relation entre les méthodes de fabrication, la flexibilité requise et le coût des installations.

II.4.3 Méta-heuristiques et problèmes de production

Dans la première phase de notre enquête, nous avons cherché dans les publications, y compris les mots-clés «production» et «méta-heuristique» (Figure II.3) dans leurs titres ou des résumés ou des mots clés.

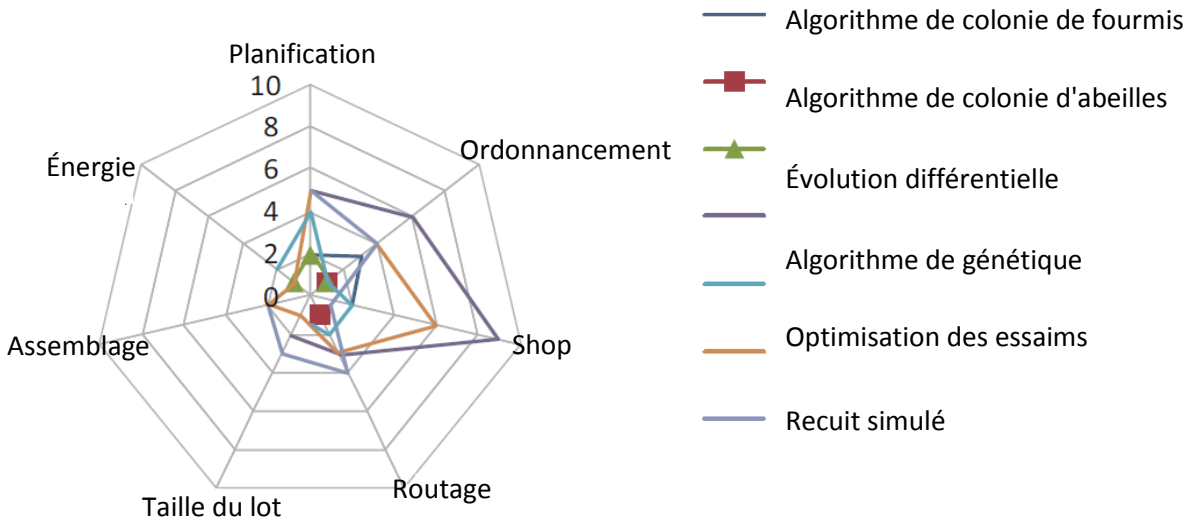


Figure II.3 Diagramme radar pour les mots clés "production" et "méta-heuristiques" [x1000] (2017, Septembre). https://scholar.google.com/

Il est clair que cette recherche ne pouvait pas détecter les documents en utilisant les noms spécifiques des techniques méta-heuristiques, plutôt que d'utiliser le terme général «méta-heuristiques».

II.5 Familles de systèmes manufacturiers

Nous avons trois types de systèmes manufacturiers :

Les systèmes manufacturiers dédiés (DMS) et les systèmes manufacturiers flexibles (FMS) sont les deux types de systèmes de fabrication de base couramment utilisés dans les industries, tandis que les systèmes manufacturiers reconfigurables (RMS) sont considérés comme l'avenir, on peut comparer qualitativement les DMS, FMS et RMS en termes de coût, d'adaptabilité, de complexité, de cadence de production et de temps de montée en puissance.

Les systèmes de production dédiés (DMS) sont des systèmes de production adaptés à un type de produit spécifique et à un volume de production fixe. Très automatisés, ces systèmes sont capables d'une productivité élevée, avec des coûts de production relativement faibles. Ce type de système de production est particulièrement bien adapté aux entreprises F2 (voir figure II.4). D'un autre côté, les DMS ne sont pas du tout flexibles. Leur structure est «fixe», c'est-à-dire très peu ou pas modifiable (Koren & al. 1999).

Le développement des technologies de l'information et de la communication (TIC) a conduit à l'émergence de systèmes de fabrication flexibles (FMS). Ces systèmes de production sont essentiellement constitués de machines à commande numérique (Eguia & al.2017) contrôlées par des contrôleurs logiques programmables (Lafou 2016). Les FMS ont une flexibilité générale, correspondant à toutes les possibilités fonctionnelles offertes par leurs machines.

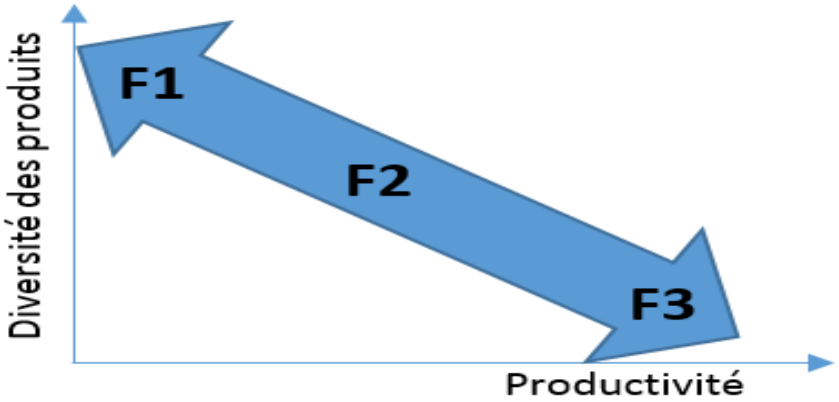


Figure II.4 Flexibilité entre diversité et productivité.

Les variations des caractéristiques des produits fabriqués par FMS se situent dans une fourchette correspondant à ces possibilités. Les machines ont la capacité de produire de nombreux produits différents, un changement de programme permet de passer d'un type de produit à un autre. Très développée dans le domaine de l'usinage, la fabrication additive en augmentant encore la diversité des produits manufacturés est aujourd'hui très représentative de FMS. Pourtant, les quantités produites sont relativement faibles, alors que les coûts d'investissement sont souvent élevés. En conséquence, leur rentabilité est difficile à atteindre.

Les FMS offrent une flexibilité sur les types de produits fabriqués, mais ils sont limités par leur coût et leur volume de production (Koren & al. 1999). Les FMS sont particulièrement adaptés aux entreprises F1 (voir figure II.4).

Ces deux types de système représentent deux extrema pour la flexibilité. Les DMS ne sont pas flexibles mais très productifs, tandis que les FMS sont très flexibles mais chers. La tendance actuelle est de s'orienter vers les entreprises de la famille F2, c'est-à-dire d'avoir des systèmes de production qui s'adaptent aux évolutions, tout en restant rentables. Le système de fabrication reconfigurable (RMS) est une troisième famille de systèmes de production adaptés à la famille F2.

Les systèmes de fabrication reconfigurables (RMS) peuvent répondre rapidement aux changements du marché, à un coût acceptable. Pour y parvenir, RMS combine la productivité du DMS avec la flexibilité du FMS. Leur flexibilité est maîtrisée, seulement ce qui est nécessaire et rien de plus. Cela signifie qu'en réponse à un changement de la demande ou de l'environnement du système de production, le système s'adapte pour répondre à ce changement. Si la quantité à produire double, le système sera configuré uniquement pour doubler son volume de production. S'il s'agit d'une fonction du produit qui change, le système sera configuré pour prendre en compte cette modification. Et si une machine du système tombe en panne, le système sera configuré pour continuer la production malgré l'échec. En offrant une solution aux

entreprises de la famille F2, RMS est souvent la seule alternative pour les entreprises qui ont besoin d'allier flexibilité et productivité. La reconfigurabilité est la capacité de s'adapter rapidement et à moindre coût aux changements. Les RMS sont des systèmes dont les structures physiques et logiques, à tous les niveaux, peuvent être modifiées rapidement et à moindre coût pour ajuster la capacité de production et la fonctionnalité autour d'une famille de produits en réponse aux changements soudains du marché (Koren & al.1999). Des changements dans la structure physique et / ou logique sont effectués afin de satisfaire les changements de la demande et de l'environnement du système de production. Pour y parvenir, les RMS sont dotés de caractéristiques qui leur permettent de faire varier leur structure dès le départ. Chacune de ces caractéristiques à un rôle dans la capacité de RMS à être flexible, productif et rentable. Huit caractéristiques sont répertoriées dans la littérature : modularité, évolutivité, convertibilité, personnalisation, diagnosticabilité, intégrabilité, mobilité et adaptabilité (Koren & al.1999; Mehrabi & al.2000; Lameche 2018).

II.5.1 Classification des caractéristiques RMS

Les caractéristiques RMS se produisent à différents niveaux de reconfigurabilité. Certains facilitent la détection d'un besoin de reconfigurabilité, d'autres facilitent une reconfiguration rapide et efficace et d'autres définissent les objectifs du système reconfigurable. Selon Capawa Fotsoh (2020) Ces trois aspects de RMS peuvent être utilisés pour faire une classification : les objectifs des RMS, le fonctionnement des RMS et la conception de RMS.

Caractéristiques relatives aux objectifs RMS

La convertibilité est la capacité de modifier la fonctionnalité d'un système pour répondre aux nouvelles exigences de production (Maganha & al.2018). Il peut s'agir de machines, de la structure globale du système ou du système de contrôle de production. Dans le cas de la convertibilité des machines, cela peut correspondre à un changement d'axes d'usinage, un changement d'outil ou un changement de fixation. Appliquée aux machines, cette fonctionnalité étend leur ensemble de fonctions, par ex. changement d'outil. La convertibilité doit être contrôlée par la diversité des produits. Par exemple, une machine capable de fraiser et de percer ne peut réaliser que des produits qui nécessitent un fraisage ou un perçage. Si l'évolution du système nécessite de nouveaux processus de production, la convertibilité peut conduire à une nouvelle structure du système en ajoutant ou en supprimant des postes de travail qui offrent des opportunités pour de nouveaux produits. La convertibilité peut également être obtenue en modifiant le système de contrôle de la production.

La personnalisation fait référence à la capacité du système à produire, au sein d'une famille de produits, un produit avec des fonctionnalités spécifiques à un client particulier. La personnalisation est l'une des causes des modifications du produit. Au sein d'une famille de produits, la personnalisation est contrôlée par le catalogue des variantes possibles (Koren & Shpitalni 2010). RMS doit être en mesure de produire chaque produit personnalisé, sans nécessairement changer radicalement sa configuration. Pour cela, la configuration RMS peut être personnalisée en fonction des plages de caractéristiques de la famille de produits. La personnalisation RMS peut être effectuée en utilisant plusieurs outils sur la même machine ou par des machines dédiées (par exemple une station de marquage LASER pour placer le logo du

client). Pourtant, la personnalisation peut être considérée comme un objectif motivant de la convertibilité.

L'évolutivité est la capacité d'un système à ajuster son volume de production. Cela peut impliquer l'ajout ou la suppression de ressources techniques ou humaines. Dans ce cas, l'ajustement doit être fait dans le but d'équilibrer la charge de chaque ressource. L'ajustement du volume de production peut également nécessiter une optimisation du transport du système. En effet, les chemins entre les ressources ont également une grande influence sur le délai de livraison. L'évolutivité d'un système est caractérisée par le nombre de ressources ajoutées et / ou supprimées pour augmenter la capacité de production globale, le temps nécessaire pour effectuer ces changements et le temps consacré au transport des produits entre les ressources.

L'adaptabilité fait référence à la capacité du système à réagir aux changements du volume de production et des caractéristiques du produit (Maganha & al.2018). L'adaptabilité est donc une combinaison d'évolutivité et de convertibilité. La décision de fabriquer un nouveau produit (convertibilité) peut dépendre du volume de production envisagé (évolutivité). Une entreprise capable de reconfigurer son système de production pour fabriquer un nouveau produit au volume demandé par le marché tout en restant rentable est une entreprise adaptable. L'adaptabilité peut être caractérisée par le volume minimum pour lequel le système peut être reconfiguré.

Caractéristiques liées au fonctionnement du RMS

De nombreux problèmes peuvent survenir lors du fonctionnement d'un système de production : perturbations soudaines, par ex. défaillance de la machine ou interruption progressive, par ex. un taux de non-conformité qui augmente lentement. Les causes des problèmes qui se posent doivent être rapidement identifiées et une stratégie appropriée mise en œuvre. Nous devons donc répondre à deux questions : où et comment agir. Cela nécessite un diagnostic du système afin de décider de reconfigurer si nécessaire. Par conséquent, une autre caractéristique du fonctionnement de RMS apparaît : la stratégie de reconfiguration. En effet, selon le diagnostic, cette caractéristique permet de résoudre rapidement le problème. Koren & Shpitalni (2010) définissent la diagnosticabilité comme la capacité de lire automatiquement l'état du système pour détecter les causes des défaillances afin qu'elles puissent être rapidement corrigées. Cette fonctionnalité permet de localiser et de traiter les pannes de manière à ce que le système ne soit pas pénalisé. Pour y parvenir, il est nécessaire d'adopter une stratégie de reconfiguration spécifique à la situation. Selon les causes de défaillance observées, le système peut être légèrement ou profondément modifié. Nous introduisons ici la notion de niveau de reconfiguration, ce que Ketfi (2005) & Kanso (2010) appelaient catégorie de reconfiguration :

- Le niveau 0 consiste à agir sans arrêter le système. Ce niveau de reconfiguration ne nécessite aucune modification de la structure du système. À ce niveau, le système peut être corrigé pendant le fonctionnement ou pendant un temps d'arrêt très court, par ex. un capteur qui ne fonctionne plus ou une palette collée sur un convoyeur. Le niveau 0 de la reconfiguration est en fait une correction du système, sans avoir à le modifier. Il s'agit de contourner l'échec du système à le faire fonctionner, par ex. dans le cas où deux machines remplissent la même fonction si l'une échoue, l'autre prend le relais. Il s'agit d'une reconfiguration corrective, qui se fait avec un minimum d'effort.
- Le niveau 1 consiste à modifier le système pour répondre aux changements de la demande ou de l'environnement du système. Ce niveau de reconfiguration nécessite des

changements dans les fonctionnalités du système : soit en ajoutant des composants déjà disponibles soit en repositionnant / réaffectant des composants. Cette reconfiguration implique des changements dans la structure du système, par ex. une machine ajoutée ou supprimée du système ou un camion déplacé d'une zone à une autre. La reconfiguration sera effectuée lorsque le système sera arrêté. En fait, c'est une évolution du système, car il passe d'un état de fonctionnement à un autre. Il s'agit d'une reconfiguration évolutive, qui nécessite plus d'efforts que le niveau 0. L'effort de reconfiguration peut être déterminé par le coût et le temps requis pour achever une reconfiguration (Huang & al.2018).

- Le niveau 2 consiste également à modifier le système, mais avec un effort accru. Par exemple, vous pouvez ajouter un composant, spécialement conçu pour la reconfiguration, au système, par ex. un porte-palette trapézoïdal qui n'existe pas dans les ressources. En effet, il donne à tout moment des informations sur l'état du système, permettant de définir un diagnostic et d'évaluer des stratégies de reconfiguration.
  La diagnosticabilité est utilisée pour déterminer le niveau de reconfiguration. Il permet également de déterminer si la reconfiguration concerne le niveau machine ou système.
  En mesurant l'effort de reconfiguration, la diagnosticabilité permet d'évaluer différentes reconfigurations. Ces évaluations peuvent être effectuées à l'aide d'un outil de simulation.
  La diagnosticabilité est la caractéristique permettant le déclenchement d'une reconfiguration.

Caractéristiques liées à la conception du RMS

La modularité, l'intégrabilité et la mobilité sont des caractéristiques liées à la conception du RMS. Ils permettent le déploiement de la stratégie de configuration choisie à l'issue de la phase de diagnostic.La modularité décrit l'utilisation d'unité standard et interchangeable pour satisfaire une variété de fonctions (Lameche & al.2017). Dans un système modulaire, les composants sont complètement séparés du système afin que l'ajout, le remplacement et / ou la modification d'un module (composant) soient possibles (Benderbal 2018). Cette caractéristique de RMS permet des changements dans les fonctions des machines ou des changements dans le volume de production.La modularité peut être contrôlée par le nombre de modules indépendants disponibles dans le système.

L'intégrabilité est la capacité d'intégrer des modules entre eux. Cela est possible grâce à des interfaces communes aux deux parties (Koren & al. 1999). Les interfaces peuvent être physiques ou logiques afin de faciliter la communication entre les différents éléments (Lameche 2018). Les interfaces standard facilitent l'interchangeabilité des modules et réduisent le temps et les coûts de reconfiguration. L'intégrabilité est particulièrement importante dans le cas d'une reconfiguration de niveau 2 où un nouveau module doit être conçu. La définition correcte des interfaces facilitera la conception et l'intégration de ce nouveau module avec les modules existants. Dans Lameche (2018) la mobilité est définie comme la capacité de déplacer des produits à travers le système. Elle peut être caractérisée par le nombre de produits transportés par unité de temps ou le nombre de trajets possibles d'un point à un autre. Une reconfiguration du système peut consister à modifier la trajectoire d'un produit, soit pour gagner du temps, soit pour ajouter une opération dans le processus de fabrication d'un produit. Cette définition

d'itinéraire peut être effectuée en utilisant les éléments déjà présents dans le système ou en ajoutant de nouvelles ressources.

Un bon diagnostic du système conduira à un bon choix de stratégie de reconfiguration, ce qui réduira l'effort de reconfiguration, plus les modules seront indépendants, plus il sera possible de définir de nouvelles fonctions pour le système et, par conséquent, plus de produits à fabriquer. La modularité favorise la convertibilité et la personnalisation. La modularité est facilitée par la présence d'interfaces standardisées, c'est-à-dire l'intégrabilité.

II.6 Exigences relatives aux critères

Pour les tendances du paradigme de fabrication des produits, la concurrence pour le développement de produits était basée sur l'optimisation de critères qui étaient populaires pendant certaines périodes historiques. Ces exigences en matière de critères (Butt 2020) ont considérablement évolué au fil du temps et sont devenus plus complexes au cours de l'histoire de la fabrication des produits. Cette progression est décrite ci-dessous :

1. Optimisation d'un seul objectif, comme la minimisation du temps nécessaire à l'achèvement d'un processus et la minimisation du coût de fabrication dans une certaine plage de valeurs.
2. Prise en compte des relations conflictuelles, où l'amélioration d'un objectif peut entraîner une dégradation de certains autres objectifs. La nécessité d'exploiter des points de vue plus larges et plus flexibles lors de l'optimisation a été reconnue.
3. Reconnaissance de la nécessité d'aborder la fabrication des produits et l'utilisation des impacts sur les environnements naturels, et de considérer l'épuisement des ressources naturelles lors de l'élaboration des critères de conception des produits.
4. Le degré de confort et de satisfaction mentale que ressentent les utilisateurs du produit est inclus dans les critères de conception du produit. Une maximisation de la satisfaction du client est nécessaire, et les facteurs esthétiques doivent être pris en compte ainsi que les performances, les qualités et les coûts.

Les facteurs 3 et 4 qui précèdent incluent des critères pour lesquels les évaluations quantitatives ne sont pas faciles, mais des solutions avancées de conception de produits doivent être déterminées en considérant systématiquement tous les nombreux facteurs.

Les facteurs évaluatifs dans la fabrication des produits sont définis comme les caractéristiques des critères (Butt 2020), qui sont classées en caractéristiques pour lesquelles des valeurs plus élevées sont plus préférables et caractéristiques pour lesquelles des valeurs plus petites sont plus préférables :

1. Caractéristiques pour lesquelles de plus grandes valeurs sont plus préférables : bénéfices, niveaux de satisfaction, utilités, niveaux d'efficacité, précisions, les efficacités (lorsque le «temps» est utilisé pour évaluer les efficacités, il correspond à des caractéristiques pour lesquelles des valeurs plus petites sont plus préférables. Lorsque la quantité de tâches accomplies est utilisée pendant les évaluations d'efficacité, «la quantité de tâches accomplies» est classée en caractéristiques pour lesquelles de plus grandes valeurs sont plus préférables).

2. Caractéristiques pour lesquelles des valeurs plus petites sont plus préférables : coût, temps (temps nécessaire pour terminer un travail «Makespan»), niveau d'insatisfaction, temps d'arrêt, besoins en énergie.

Les conceptions de produits visent généralement à fournir des niveaux de satisfaction plus élevés. Les critères des niveaux de satisfaction peuvent exprimer non seulement des attributs objectifs tels que le coût et le poids, mais aussi des attributs subjectifs tels que les préférences humaines. Les niveaux d'insatisfaction inclus dans les caractéristiques pour lesquelles des valeurs plus petites sont préférables peuvent convenir à des problèmes spécifiques où les insatisfactions spécifiques doivent être minimisées, mais peuvent ne pas convenir lorsque l'on vise à concevoir des produits plus avancés et plus préférables. Ces valeurs sont souvent utilisées dans le domaine de la prise de décision (une méthode pour exprimer quantitativement les préférences subjectives des décideurs ou les niveaux de satisfaction pour les alternatives et les attributs).

Par exemple, les décideurs de conception/fabrication de produits sont susceptibles d'avoir des exigences contradictoires pour maximiser les performances des produits et minimiser les coûts de fabrication des produits. Ces exigences peuvent être exprimées à l'aide de fonctions de satisfaction et lorsque le coût de fabrication est maintenu constant, les décideurs de conception peuvent ignorer en toute sécurité les conceptions qui ne satisfont pas un niveau minimum de performance d'un produit particulier lorsque le niveau de satisfaction est faible. Lorsque la conception commence à satisfaire les performances requises, le niveau de satisfaction peut soudainement approcher une valeur de 1. Certaines caractéristiques de la fonction de satisfaction sont représentées sur la figure II.5 ou l'axe horizontal montre d_{ij} qui exprime la distance entre la valeur réelle z_j de l'attribut j et la valeur de l'objectif z_{ij}^*, tandis que l'axe vertical exprime le niveau de satisfaction s_{ij} du décideur, i pour l'attribut et j ∈ [0,1]

$$s_{ij} = \frac{1}{\pi} tan^{-1}\{-\alpha(d_{ij} - \beta)\} + 0.5 \quad d_{ij} = \left\{\frac{z_{ij} - z_{ij}^*}{z_{ij}^*}\right\} \qquad \textit{II.3}$$

α et β sont les valeurs des coefficients.

La plus petite valeur de coefficient α donne une courbe à pente plus douce, tandis que la plus grande valeur augmente la pente. Le coefficient β correspond à la valeur de *d* où s = 0.5.

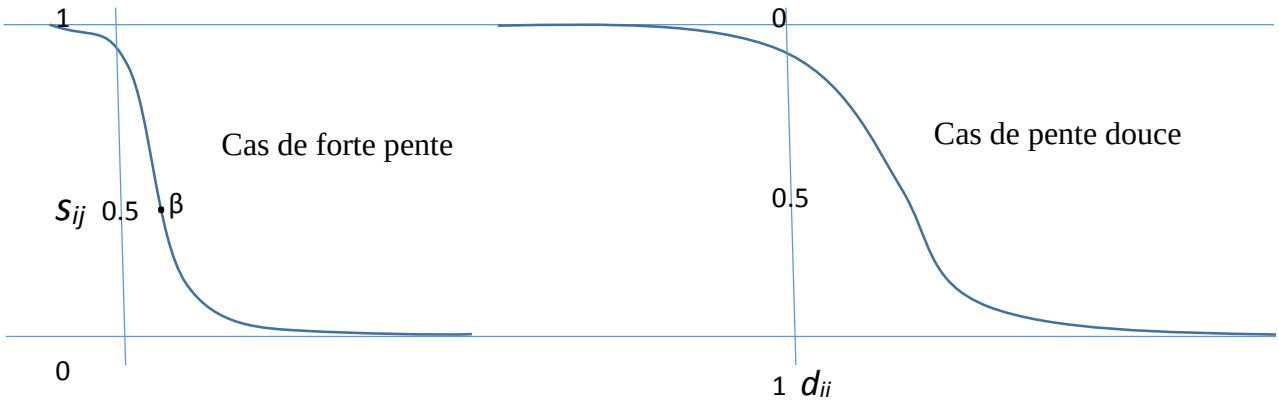


Figure II.5 Courbe de la fonction de satisfaction forte pente & pente douce

II.7 Planification et Ordonnancement

L'un des défis des systèmes de fabrication actuels est leur adaptation aux exigences des environnements changeants soumis à des contraintes multiples de tâches et de ressources. Le bon fonctionnement de ces systèmes nécessite une planification et un ordonnancement avancés connus sous l'acronyme APS (Advanced Planning & Scheduling). La figure II.6 montre la situation des APS dans les systèmes de fabrication. APS sont particulièrement bien adaptés aux environnements où des méthodes de planification plus simples ne peuvent pas traiter correctement les compromis complexes entre des priorités concurrentes. Cependant, la plupart des problèmes d'ordonnancement des APS dans le monde réel sont confrontés à des contraintes inévitables telles que **la date d'échéance**, la capacité, le coût du transport, le coût d'installation et **les ressources disponibles**.

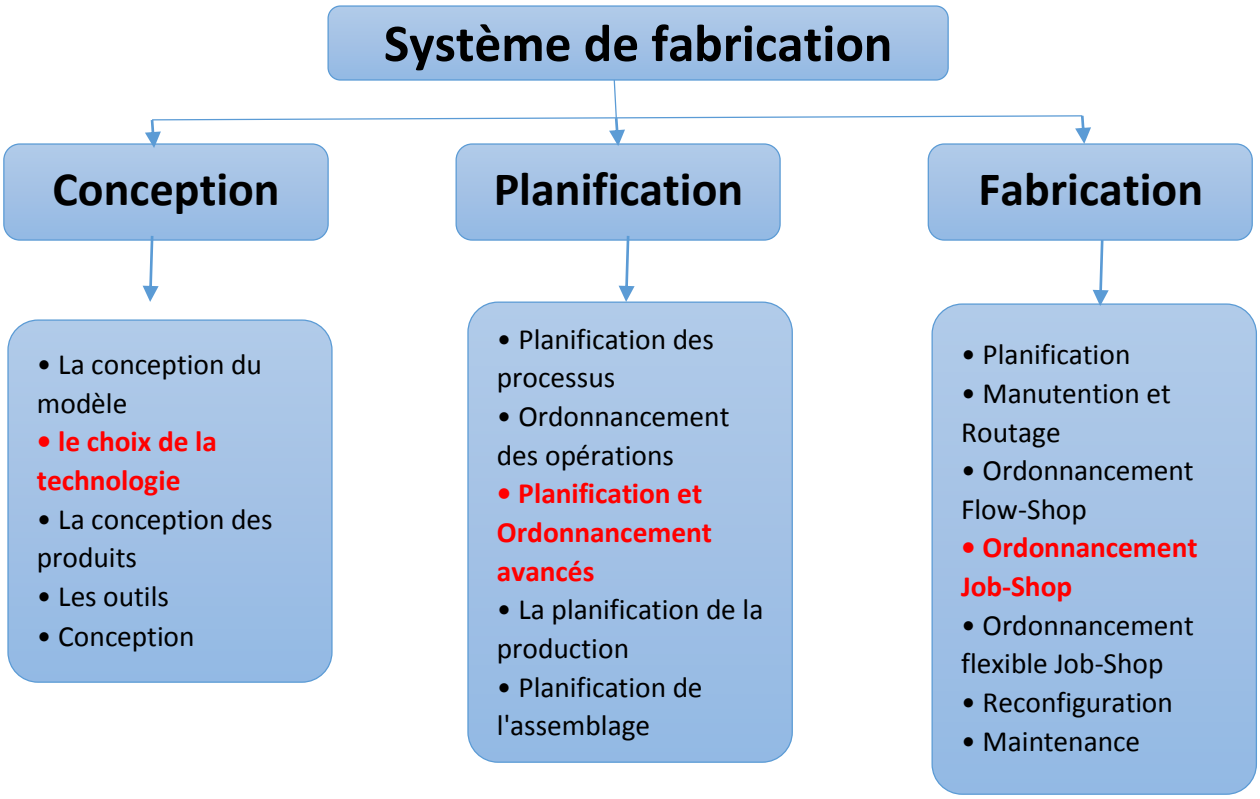


Figure II.6 Les trois premières phases dans les systèmes manufacturiers

D'une manière générale, nous devons obtenir une «flexibilité» efficace non seulement en réponse à l'environnement complexe réel mais aussi pour satisfaire toutes les contraintes combinatoires. Ainsi, comment formuler les problèmes complexes de l'APS et trouver des solutions satisfaisantes jouent un rôle important dans les systèmes de fabrication. APS utilisent des techniques de planification et d'ordonnancement qui prend en compte un large éventail de contraintes (Figure II.6) pour produire un plan optimisé englobant l'ensemble des facteurs limitant la performance d'une entreprise tels que :

- La Disponibilité des machines
- La capacité des machines
- Les exigences du client (dates d'échéance)

• Le niveau de stock de sécurité
• Le coût
• La planification avancée

Il est important de comprendre la différence entre la planification et l'ordonnancement dans les systèmes manufacturier. La planification consiste à décomposer un travail en opérations et à définir une séquence logique d'événements qui livreront un produit terminé. La planification consiste donc à définir ce qui doit être fait et comment, par contre L'ordonnancement prend en compte les heures et les dates d'échéance pour appliquer un ordre chronologique au plan, de sorte qu'il peut être visualisé sous une forme de chronologie. Cela signifie également que les opérations sont affectées à une ressource, ce qui garantit qu'elle sera réellement disponible à la date et à l'heure requises. L'ordonnancement fournit efficacement la spécificité et les objectifs du plan.

II.7.1 Représentation des ordonnancements

Un ordonnancement est traditionnellement représenté par les diagrammes de Gantt (Gantt 1974). Tout en mappant toujours horizontalement l'échelle de temps, par rapport à l'axe vertical, les diagrammes de Gantt peuvent être orientés machine ou orientés Job. La figure II.7 montre ces deux types de diagrammes pour la même configuration. La longueur de chaque barre représente la durée de l'opération dans le Job ou la machine,

selon le diagramme. Notez que le premier des deux diagrammes indique la séquence dans laquelle les Jobs sont traités sur chaque machine, c'est-à-dire que la première machine traite le Job 1, puis le Job 3 et enfin le Job 2.

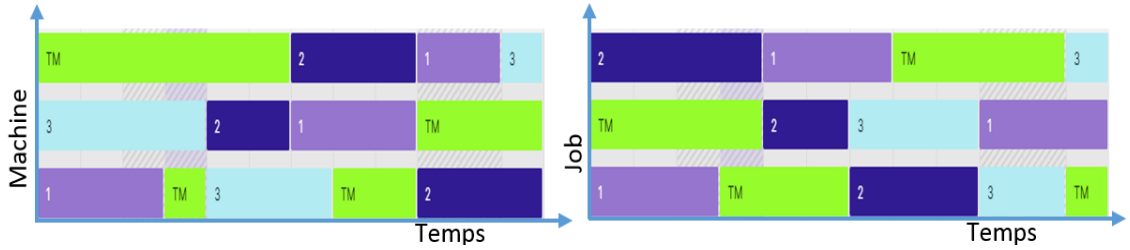


Figure II.7 Diagrammes de Gantt orientés machine et orientés Job (Domschke et al. 1997)

Cette séquence de travaux peut être représentée par un vecteur, c'est-à-dire que la séquence de travaux précédente est désignée par (1, 3, 2).Une représentation graphique disjonctive plus adéquate (Bła˙zewicz & al. 2000) qui sera brièvement décrite ci-dessous :

Un graphe disjonctif $G = (V, C \cup D)$ où V désigne un ensemble de sommets correspondant aux opérations des Jobs. C est un ensemble d'arcs conjonctifs reliant toutes les deux opérations consécutives du même Job. Les bords non orientés de l'ensemble D connectent des opérations non ordonnées mutuellement qui nécessitent la même machine pour leur exécution. Chaque sommet est pondéré avec le temps de traitement de l'opération à son démarrage. Les bords ont un poids de 0. Dans la représentation graphique disjonctive, l'obtention d'un programme équivaut à sélectionner un arc dans chaque disjonction, c'est-à-dire à transformer chaque bord disjonctif non dirigé en un arc conjonctif dirigé. En fixant les directions de tous les bords disjonctifs, l'ordre d'exécution de toutes les opérations conflictuelles nécessitant la même machine est déterminé et un ordonnancement complet est obtenu. De plus,

le graphique résultant doit être acyclique. Un exemple de graphe disjonctif est présenté par la Figure II.8. Dans cet exemple, il y a trois machines M = {M1, M2, M3} et un ensemble de trois tâches J = {J1, J2, J3} qui sont décrites par les séquences suivantes des opérations : J1: T1 → T2 → T3,

J2: T4 → T5, J3: T6 → T7 → T8. Pour toute opération Ti, la machine M (Ti) requise et le temps de traitement pi sont supposés connus.

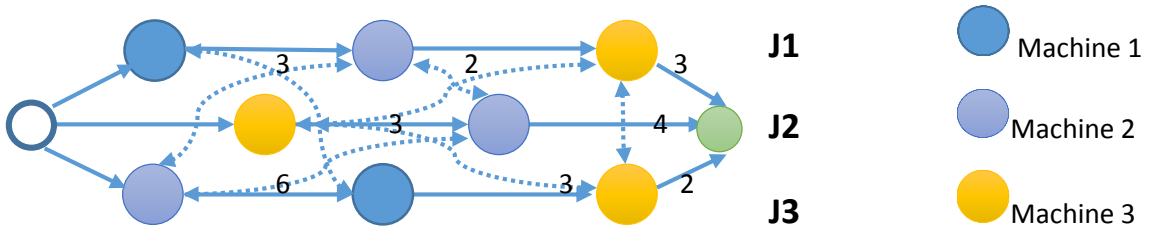


Figure II.8 Exemple pour un graphe disjonctif (Bła˙zewicz et al. 2000)

II.7.2 Classes d'ordonnancement

Pour les problèmes d'ordonnancement, plusieurs classes peuvent être identifiées, ces classes sont utilisées pour réduire le nombre des solutions possibles ce qui réduit l'effort de recherche de bonnes ou optimales solutions et pour identifier les propriétés de l'espace de solution (des solutions bonnes ou optimales) afin d'ajuster les méthodes de solution en conséquence.

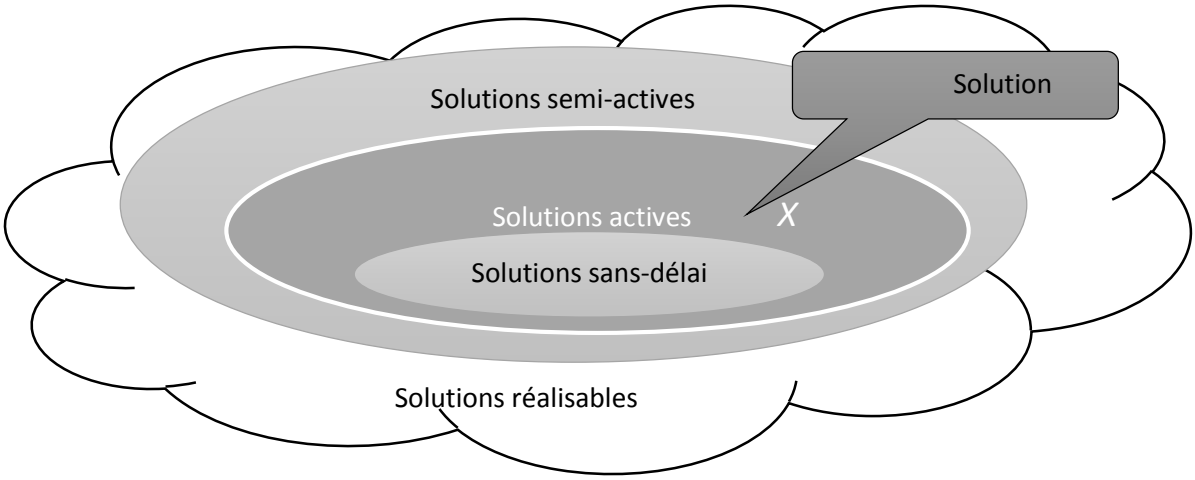


Figure II.9 Classes d'ordonnancement pour les problèmes Job-Shop

Ordonnancement sans-délai : Une solution réalisable est appelée sans-délai (Baker 1974) si aucune machine n'est maintenue inactive pendant qu'une opération est en attente de traitement (c'est-à-dire qu'elle interdit l'inactivité non forcée).

Ordonnancement actif : Une solution réalisable est dite active (Gen & Cheng 1997 ; Baker 1974) si aucune opération ne peut démarrer plus tôt sans retarder au moins une autre opération. Les solutions sans-délai sont actives tandis que l'inverse ne tient pas (Figure II.9). Notez que pour les problèmes avec un critère régulier, il existe une solution active qui est optimale (cela ne s'applique pas aux critères non réguliers).

Ordonnancement semi-actif : Une solution réalisable est appelée semi-active (Gen & Cheng 1997 ; Baker 1974) si aucune opération ne peut être effectuée plus tôt sans modifier l'ordre de traitement sur l'une des machines. De toute évidence, une solution active doit être semi-active (l'inverse n'est pas nécessairement vrai). Intuitivement, les ordonnancements semi-actifs sont caractérisés par le fait que chaque opération est décalée vers la gauche autant que possible, c'est-à-dire, étant donné la séquence actuelle des opérations sur chaque machine, chaque opération est démarrée dès que possible.

II.8 Problème d'ordonnancement de projet

L'ordonnancement est un processus décisionnel qui est utilisé régulièrement dans de nombreuses industries manufacturières Figure II.10. Comme problème il traite de l'allocation des ressources aux tâches sur des périodes de temps données et son but est d'optimiser un ou plusieurs objectifs. Les ressources et les tâches d'une organisation peuvent prendre de nombreuses formes différentes. Les ressources peuvent être des machines dans un atelier, des pistes dans un aéroport, des équipages sur un chantier de construction, des unités de traitement dans un environnement informatique, etc. Les tâches peuvent être des opérations dans un processus de production, des décollages et des atterrissages dans un aéroport, des étapes d'un projet de construction, des exécutions de programmes informatiques, etc. Chaque tâche peut avoir un certain niveau de priorité, une heure de début et une date d'échéance. Les objectifs peuvent également prendre de nombreuses formes différentes. Un objectif peut être la minimisation du temps d'achèvement de la dernière tâche et un autre peut être la minimisation du nombre de tâches terminées après leur date d'échéance respective. L'ordonnancement, en tant que processus décisionnel, joue un rôle important dans la plupart des systèmes de fabrication et de production ainsi que dans la plupart des environnements de traitement de l'information. Il est également important dans les milieux de transport et de distribution et dans d'autres types d'industries. Les problèmes d'ordonnancement surviennent généralement au sein d'une hiérarchie décisionnelle dans laquelle il suit certaines décisions antérieures et plus fondamentales.

II.8.1 Les entrées

Vous avez besoin de plusieurs types d'entrées pour créer un ordonnancement :

- Planification : Comprendre le temps libre, les quarts de Job et la disponibilité des ressources.
- La portée du projet : À partir de là, vous pouvez déterminer les dates clés de début et de fin, les principales hypothèses derrière le plan et les principales contraintes et restrictions. Vous pouvez également inclure les attentes des parties prenantes, qui détermineront souvent les jalons du projet.
- Flexibilité : Vous devez les comprendre pour vous assurer qu'il reste suffisamment de temps pour traiter les risques identifiés - et les risques non identifiés (les risques sont identifiés grâce à une analyse approfondie des risques).
- Listes d'activités et de ressources nécessaires : Encore une fois, il est important de déterminer s'il existe d'autres contraintes à prendre en compte lors de l'élaboration de la

planification. La compréhension des capacités en ressources et de l'expérience dont vous disposez.

II.8.2 Outils d'ordonnancement

Une bonne analyse doit prendre en compte les problèmes de temps et de disponibilité des ressources qui peuvent rendre la planification moins flexible. Voici quelques outils et techniques pour mieux développer l'algorithme :

- Planifier l'analyse : Il s'agit d'une représentation graphique des tâches du projet, du temps nécessaire à leur exécution et de la séquence dans laquelle elles doivent être exécutées. Les diagrammes de Gantt et PERT sont des formats courants.
- Analyse du chemin critique : C'est le processus consistant à examiner toutes les tâches à accomplir et à calculer le «meilleur chemin» à suivre pour terminer le projet dans les plus brefs délais. La méthode calcule les heures de début et de fin des tâches de projet et estime les dépendances entre elles pour créer un ordonnancement.
- Optimisation : cet outil minimise la durée totale d'un projet en réduisant le temps alloué à certaines tâches. Vous pouvez utiliser deux méthodes ici :

1. **Crashing**: c'est là que vous allouez plus de ressources à une tâche, réduisant ainsi le temps nécessaire pour la terminer.
2. **Fast-Tracking** : Cela implique de réorganiser les tâches pour permettre un travail plus parallèle. Signifie que les choses que vous feriez normalement l'une après l'autre se font maintenant en même temps. Cependant, gardez à l'esprit que cette approche augmente le risque de ne pas répondre aux changements.

II.8.3 Contraintes du projet

Le principal impact des contraintes du projet est la probabilité de retarder l'achèvement du projet. Il existe trois types de contraintes : technologique, de ressource et physique (Baptiste & al. 2001).

Les contraintes technologiques sont liées à l'ordre dans lequel les activités individuelles du projet doivent être achevées. Par exemple, lors de la fabrication d'un produit, la finition doit avoir lieu avant l'emballage. Les contraintes de ressources sont liées au manque de ressources adéquates qui peuvent forcer des activités parallèles à être exécutées en séquence. La conséquence d'un tel changement est le retard dans la date d'achèvement du projet. Les contraintes physiques sont causées par des conditions contractuelles ou environnementales. Par exemple, en raison des contraintes d'espace, une activité telle que l'emballage d'un produit peut ne pas être exécuté par une seule machine. En général, du point de vue ordonnancement, les projets peuvent être classés en termes de temps ou de ressources. Un projet est classé comme limité dans le temps dans les situations où le chemin critique est retardé et l'ajout de ressources peut ramener le projet dans les délais et le projet terminé à la date requise. Cependant, l'utilisation des ressources supplémentaires ne devrait pas être supérieure à ce qui est absolument nécessaire. L'objectif principal des projets soumis à des contraintes de temps est l'utilisation des ressources. D'un autre côté, un projet est limité en ressources si le niveau de disponibilité des ressources ne peut pas être dépassé. Dans les situations où les ressources sont

insuffisantes, le retard du projet est acceptable, mais le retard devrait être minime. L'ordonnancement dans ces situations a pour objectif de hiérarchiser et d'allouer les ressources de manière à ce qu'il y ait un délai de projet minimal. Cependant, il est également important de s'assurer que la limite de ressources n'est pas dépassée et que les relations dans le réseau ne sont pas modifiées.

II.8.4 Ressources

Les ressources les plus importantes dont le projet disposent pour planifier et gérer au jour le jour sont les machines et les matériaux (Brucker & Knust 2011). De toute évidence, si ces ressources sont disponibles en abondance, le projet pourrait être accéléré pour réduire la durée du projet. D'un autre côté, si ces ressources sont sévèrement limitées, alors le résultat sera probablement un retard dans le temps d'achèvement du projet. Selon le type de ressources, les coûts de fourniture d'une telle abondance pour accélérer le temps d'achèvement du projet peuvent être très élevés.

Cependant, si les ressources sont facilement disponibles et qu'aucune prime excédentaire n'est engagée pour les utiliser dans le projet, le coût du projet devrait être faible, car certains coûts du projet sont liés aux ressources tandis que d'autres sont susceptibles de dépendre du temps.

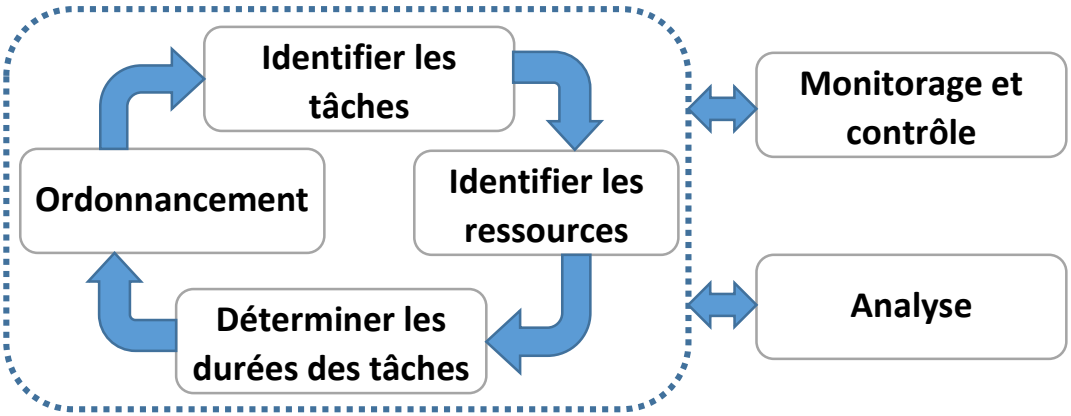


Figure II.10 Etapes d'ordonnancement du projet.

En général, les projets d'une durée plus courte sont moins chers. Plus la durée du projet est longue, plus le coût global du projet sera élevé. La réalité est que tant que le travail sur un projet est en cours, il continuera à attirer des ressources sur son orbite. Quels que soient les paramètres du projet, il est peu probable que la relation entre coût et durée soit linéaire. Pour tout projet particulier, la décision de placer le projet sur la courbe entre le point de moindre durée avec ses besoins en ressources plus élevés associés et un point de durée accrue avec ses besoins en ressources inférieurs associés dépend des paramètres particuliers du projet.

En générale, il existe deux approches pour lisser les ressources requises :

Time-limited resource considerations :

Dans ce cas, l'accent sera mis sur l'achèvement du projet dans un délai spécifié, généralement déterminé par l'analyse du chemin critique (Kenneth & Dan 2018). Des ajustements dans la planification de toute tâche, et les ressources nécessaires à un moment

donné, doivent être effectués. De toute évidence, il ne peut y avoir d'ajustement des tâches qui se trouvent sur le chemin critique.

Resource-limited resource considerations

Dans ce cas, le projet doit être achevé avec les ressources disponibles même si cela signifie prolonger la durée du projet. Si la demande totale de ressources dépasse la disponibilité des ressources à tout moment (Kenneth & Dan 2018), certaines des activités doivent être retardées jusqu'à ce que la disponibilité des ressources soit suffisante.

II.9 Classification des ateliers

Une classification des problèmes d'ordonnancement dans un atelier peut s'opérer selon le nombre de machines et leur ordre d'utilisation pour fabriquer un produit, qui dépend de la nature de l'atelier considéré. Un atelier est caractérisé par le nombre de machines qu'il contient et par son type.

II.9.1 Machine Unique

Dans ce cas, l'ensemble des tâches à réaliser est fait par une seule machine. Les tâches alors sont composées d'une seule opération qui nécessite la même machine. L'une des situations intéressantes où on peut rencontrer ce genre de configurations est le cas où on est devant un système de production comprenant une machine goulot qui influence l'ensemble du processus. L'étude peut alors être restreinte à l'étude de cette machine.

II.9.2 Machines parallèles

Dans ce cas, on dispose d'un ensemble de machines identiques pour réaliser les travaux. Les travaux se composent d'une seule opération et un travail exige une seule machine. L'ordonnancement s'effectue en deux phases : la première phase consiste à affecter les travaux aux machines et la deuxième phase consiste à établir la séquence de réalisation sur chaque machine.

II.9.3 Les ateliers de type flow-shop

Appelés également ateliers à cheminement unique, ce sont des ateliers où une ligne de fabrication est constituée de plusieurs machines en série; toutes les opérations de toutes les tâches passent par les machines dans le même ordre. Dans les ateliers de type flow-shop hybride, une machine peut exister en plusieurs exemplaires identiques fonctionnant en parallèle.

II.9.4 Les ateliers de type job-shop

Appelés également ateliers à cheminement multiple, 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 extension du modèle job-shop classique ; sa particularité réside dans le fait que plusieurs machines sont potentiellement capables de réaliser un sous-ensemble d'opérations.

II.9.5 Les ateliers de type open-shop

Ce type d'atelier est moins contraint que celui de type flow-shop ou de type job-shop. Ainsi, l'ordre des opérations n'est pas fixé a priori ; le problème d'ordonnancement consiste, d'une part, à déterminer le cheminement de chaque produit et, d'autre part, à ordonnancer les produits en tenant compte des gammes trouvées, ces deux problèmes pouvant être résolus simultanément. Comparé aux autres modèles d'ateliers, l'open-shop n'est pas couramment utilisé dans les entreprises.

II.10 Les différents types de contraintes de tâches

Les contraintes typiques qui peuvent être spécifiées sur les tâches en temps réel sont de trois classes (Buttazzo 2011) : les contraintes de synchronisation, les relations de précédence et les contraintes d'exclusion sur les ressources partagées.

II.10.1 Contraintes de temps

Les systèmes sont caractérisés par des activités avec des contraintes de temps strictes qui doivent être respectées pour obtenir le comportement souhaité. Une contrainte de synchronisation typique sur une tâche est la date limite, qui représente le temps avant lequel un processus doit terminer son exécution sans endommager le système. Si une échéance est spécifiée par rapport à l'heure d'arrivée de la tâche, elle est appelée échéance relative, tandis que si elle est spécifiée par rapport à l'heure zéro, elle est appelée échéance absolue. En fonction des conséquences d'une échéance manquée, les tâches sont généralement distinguées en trois catégories :

Hard : une tâche en temps réel est dite difficile si le non-respect de son échéance peut avoir des conséquences catastrophiques sur le système.

Firm : Une tâche en temps réel est dite firm si le non-respect de son échéance ne cause aucun dommage au système, mais la sortie n'a aucune valeur.

Soft : Une tâche en temps réel est dite Soft si le non-respect de sa date limite a encore une utilité pour le système, bien que provoquant une dégradation des performances.

En général, une tâche peut être caractérisée par :

L'heure d'arrivée c'est l'heure à laquelle une tâche est prête à être exécutée; il est également appelé temps de demande ou temps de libération.

Le temps de calcul c'est le temps nécessaire au processeur pour exécuter la tâche sans interruption.

Le délai absolu c'est le délai avant lequel une tâche doit être terminée pour éviter d'endommager le système.

Date limite relative c'est la différence entre la date limite absolue et l'heure de la demande.

L'heure de début c'est l'heure à laquelle une tâche commence son exécution;

L'heure de fin c'est l'heure à laquelle une tâche termine son exécution;

Le temps de réponse c'est la différence entre le temps de fin et le temps de demande.

II.10.2 Contraintes de préséance

Dans certaines applications, les activités de calcul ne peuvent pas être exécutées dans un ordre arbitraire mais doivent respecter certaines relations de précédence définies au stade de la conception (Buttazzo 2011). Ces relations de précédence sont généralement décrites par un graphe acyclique dirigé, où les tâches sont représentées par des nœuds et les relations de précédence par des flèches. Un graphe de précédence induit un ordre partiel sur l'ensemble de tâches.

La notation Ja $\prec$ Jb spécifie que la tâche Ja est un prédécesseur de la tâche Jb, ce qui signifie que G contient un chemin dirigé du nœud Ja au nœud Jb.

La notation Ja $\rightarrow$ Jb spécifie que la tâche Ja est un prédécesseur immédiat de Jb, ce qui signifie que G contient un arc dirigé du nœud Ja vers le nœud Jb.

II.10.3 Contraintes de ressources

Du point de vue du processus, une ressource est toute structure qui peut être utilisée par le processus pour faire avancer son exécution. Typiquement, une ressource peut être une structure de données, un ensemble de variables, une zone de mémoire principale, un fichier, un morceau de programme ou un ensemble de registres d'un périphérique. Une ressource dédiée à un processus particulier est dite privée, tandis qu'une ressource qui peut être utilisée par plusieurs tâches est appelée une ressource partagée.

Pour maintenir la cohérence des données, de nombreuses ressources partagées n'autorisent pas les accès simultanés par des tâches concurrentes, mais nécessitent leur exclusion mutuelle. Cela signifie qu'une tâche ne peut pas accéder à une ressource R si une autre tâche est à l'intérieur de R manipulant ses structures de données. Dans ce cas, R est appelé une ressource mutuellement exclusive. Un morceau de code exécuté sous des contraintes d'exclusion mutuelle est appelé une section critique.

II.11 Classification des algorithmes d'ordonnancement

Parmi la grande variété d'algorithmes proposés pour la planification des tâches on peut identifier les principales classes suivantes (Yahyaoui 2013):

II.11.1 Préemptif vs non préemptif.

- Dans les algorithmes préemptifs, la tâche en cours peut être interrompue à tout moment pour affecter le processeur à une autre tâche active, selon une politique d'ordonnancement prédéfinie.

- Dans les algorithmes non préemptifs, une tâche, une fois lancée, est exécutée par le processeur jusqu'à son achèvement. Dans ce cas, toutes les décisions de planification sont prises lorsque la tâche termine son exécution.

II.11.2 Statique vs dynamique.

- Les algorithmes statiques sont ceux dans lesquels les décisions d'ordonnancement sont basées sur des paramètres fixes, assignés aux tâches avant leur activation.

- Les algorithmes dynamiques sont ceux dans lesquels les décisions d'ordonnancement sont basées sur des paramètres dynamiques qui peuvent changer au cours de l'évolution du système.

II.11.3 Hors ligne vs en ligne.

- Un algorithme de planification est utilisé hors ligne s'il est exécuté sur l'ensemble des tâches avant l'activation des tâches. Le programme ainsi généré est stocké dans une table et exécuté ultérieurement par un répartiteur.

- Un algorithme de planification est utilisé en ligne si des décisions de planification sont prises au moment de l'exécution chaque fois qu'une nouvelle tâche entre dans le système ou lorsqu'une tâche en cours se termine.

II.11.4 Optimal vs Non optimal.

- Un algorithme est dit optimal s'il minimise une fonction donnée définie sur l'ensemble de tâches. Lorsque le seul souci est de parvenir à un calendrier réalisable, alors un algorithme est dit optimal s'il est capable de trouver un calendrier réalisable, s'il en existe un.

- Un algorithme est dit non optimal s'il est guidé par une fonction heuristique dans la prise de ses décisions d'ordonnancement. Un algorithme heuristique tend vers le planning optimal, mais ne garantit pas sa recherche.

De plus, un algorithme est dit clairvoyant s'il connaît l'avenir, c'est-à-dire s'il connaît à l'avance les heures d'arrivée de toutes les tâches. Bien qu'un tel algorithme n'existe pas dans la réalité, il peut être utilisé pour comparer les performances d'algorithmes réels avec les meilleurs possibles.

II.12 Ordonnancement sous contraintes de ressources

De nombreux documents traitant l'ordonnancement ont été confrontés au problème de la gestion de l'exécution d'un ensemble de tâches qui doivent être effectuées en fonction d'un besoin de ressources préétabli et d'une disponibilité maximale des ressources dans le système. Presque toutes les recherches effectuées dans ce domaine se sont concentrées sur la minimisation du temps maximum d'achèvement de toutes les tâches, en respectant les contraintes, cela signifie qu'en général, ces modèles ne tiennent pas compte de la répartition des ressources dans le temps, ce qui révèle leur inadéquation dans de nombreux scénarios de la vie

réelle où une répartition équilibrée des ressources est nécessaire pour éviter des pics indésirables qui peuvent être préjudiciables en termes de coûts. La catégorie de problèmes liés à l'ordonnancement des tâches dont l'objectif est de minimiser une mesure de variation de l'utilisation des ressources est connue sous le nom d'ordonnancement de lissage des ressources.Formellement, nous pouvons le définir comme suit :

Donné un ensemble $R= \{R_1, \ldots R_K\}$ de K types de ressources et un ensemble $T = \{T_1, \ldots T_n\}$ de n tâches à effectuer à l'aide de ces ressources. En particulier, l'exécution de chaque tâche $T_i \in T$ nécessite la disponibilité simultanée d'une quantité constante préfixée de $r_{ik} \geq 0$ unités de chaque type de ressource R_k, avec $k = 1, \ldots K$. En outre, les relations d'incompatibilité entre des paires de tâches sont définies de telle sorte que lorsqu'une incompatibilité se produit, les tâches concernées ne peuvent pas être s'exécutées simultanément. Nous désignons un ordonnancement S des tâches dans T comme un vecteur $(t_1, ..., t_n)$ d'entiers positifs, où t_i est l'heure de début de la tâche T_i et t_{i+1} est son heure de fin. Dans ce qui suit, nous ne considérerons que les échéanciers réalisables, c'est-à-dire on prend en considération que les solutions faisables. Ce paramètre peut être interprété comme un cas particulier d'ordonnancement disjonctif et d'ordonnancement avec des ensembles. Les ensembles interdits sont utilisés pour décrire les contraintes de compatibilité dans les tâches d'ordonnancement sur différents types de machines. Par exemple, si n tâches nécessitent le même type de machine, et qu'il existe exactement m machines, alors au maximum m de ces n tâches peuvent être programmées simultanément à tout moment. Cette situation peut être modélisée par des ensembles interdits, c'est-à-dire des sous-ensembles de tâches qui ne peuvent pas être programmées simultanément.

II.12.1 Minimisation du Makespan

L'objectif est d'aborder simultanément le Makespan, l'équilibrage des ressources et le pic au moyen d'un algorithme d'optimisation ; en fait, le but est de minimiser le pic d'utilisation des ressources et la durée totale de l'exécution.

Le problème de base est comment affecter des tâches au nombre minimum de tranches de temps (instants de temps) de manière à ce qu'il n'y ait pas de conflits. Soit x_{it} une variable binaire indiquant si la tâche i est planifiée à t, et soit T une limite supérieure du nombre de tranches de temps disponibles dans lesquels toutes les tâches doivent être exécutées. De plus, soit A l'ensemble des paires de tâches incompatibles. La formulation mathématique associée à la minimisation du Makespan est la suivante :

Minimiser τ

Avec

$$\begin{cases} \tau \geq tx_{it}, & i= 1, \ldots, n \quad t = 1, \ldots, T \\ \sum_{t=1}^{T} x_{it} = 1 \quad i = 1, .. , n \\ x_{it} + x_{jt} \leq 1 \quad \forall (i, j) \in A \quad t = 1, \ldots, T \\ x_{it} \in \{0, 1\} \quad i= 1, \ldots, n \quad t = 1, \ldots, T \end{cases}$$

II.4

Cela signifie que l'objectif est une fonction min-max, chaque tâche doit être planifiée dans l'intervalle {0..T} et si i et j sont incompatibles, ils ne peuvent pas être planifiés dans le même intervalle de temps. Il est facile de voir qu'avec ces contraintes le problème est NP-Hard.

II.12.2 L'approche gourmande

L'approche la plus simple consiste à ordonner les tâches selon une sorte de règle puis, selon l'ordre trouvé, à les allouer à la tranche de temps qui minimise localement la fonction objectif.

Tenez compte des trois paramètres suivants associés aux besoins en ressources:

1. Total Resource Requirement (TRR): c'est la totalité des ressources requises par tous les types.
2. Maximum Resource Requirement (MRR): il s'agit du maximum de ressources requises par tous les types.
3. Besoin moyen en ressources (ARR): c'est la moyenne des ressources requises par tous les types.

Tenez compte des règles de fonctionnement suivantes (Dorndorf & al. 2000) :

• Smallest Total Resource Requirement First (STRRF) :	les tâches sont classées selon les besoins en ressources totales les plus faibles.
• Largest Total Resource Requirement First (LTRRF) :	les tâches sont classées selon les besoins en ressources totales les plus élevés.
• Alternate Total Resource Requirement(ATRR) :	les tâches sont ordonnées en alternant les besoins en ressources totales les plus importants et les plus petits.
• Smallest Maximum Resource Requirement First (SMRRF) :	les tâches sont ordonnées en fonction des besoins en ressources maximales les plus petits.
• Largest Maximum Resource Requirement First (LMRRF) :	les tâches sont classées selon les besoins en ressources maximales les plus élevés.
• Alternate Maximum Resource Requirement First (AMRR) :	les tâches sont ordonnées en alternant les besoins en ressources maximales les plus grands et les plus petits.
• Smallest Average Resource Requirement (SARRF) :	les tâches sont classées selon les besoins moyens en ressources les plus faibles.
• Largest Average Resource Requirement (LARRF) :	les tâches sont classées selon les besoins moyens en ressources les plus élevés.
• Alternate Average Resource Requirement (AARR) :	les tâches sont ordonnées en alternant les besoins moyens en ressources les plus grands et les plus petits.

Soit *R* l'ensemble de toutes les règles présentées, l'algorithme gourmand peut être écrit comme :

a) Construisez la liste *L*, où les tâches sont ordonnées selon l'une des règles de *R*.
b) Tant que $L \neq$ Vide
    a. Prenez la première tâche *i* de *L* et affectez-la au premier intervalle de temps.
    b. Supprimer *i* de *L*.

II.12.3 L'approche méta-heuristique

En général, lorsque la durée doit être minimisée, des techniques qui tendent à éviter les similitudes de solutions sont conçues. En effet, dans le cas des approches gourmandes, la probabilité de trouver des plannings de même longueur avant de trouver un Makespan inférieur est très élevée. Compte tenu de la complexité informatique des problèmes limités en ressources, les exigences nous obligent des méthodes rapides et efficaces, ce qui nous ouvre à utiliser les méthodes de descentes qui conduisent à une amélioration de la fonction objectif. Cette restriction conduit souvent à déterminer un optimum local avec une qualité de solution non satisfaisante. Afin de surmonter ce problème, des solutions représentant une détérioration intermédiaire de la valeur de la fonction objectif doivent également être examinées. Cependant, cela porte le danger qu'une solution localement optimale soit revisitée pendant le processus de recherche et que certaines régions de l'espace de solution soient examinées à plusieurs reprises tandis que d'autres ne sont pas inspectées du tout. Pour cette raison, des méthodes de recherche heuristique modernes telles que le recuit simulé et la recherche tabou ont été introduites (Gen & al.2016). Ces méthodes d'amélioration représentent des méta-heuristiques qui peuvent être décrites comme des stratégies maîtresses pour contrôler et guider d'autres heuristiques afin de trouver des solutions au-delà de celles habituellement déterminées en identifiant un optimum local. Un autre groupe important de méta-heuristiques est constitué par les algorithmes évolutionnaires qui diffèrent des méthodes d'amélioration en modifiant et en combinant une collection de solutions au lieu de n'en transformer qu'une seule à chaque étape du processus de recherche. Au sein de ce groupe, les algorithmes génétiques sont les plus populaires. D'autres méta-heuristiques récentes sont, par exemple, les réseaux de neurones et les algorithmes de colonies de fourmis. Le but fondamental des méta-heuristiques (comme la recherche tabou ou AG (Kucuksayacigil & Ulusoy 2020)) consiste à examiner efficacement l'espace d'un problème défini par toutes ses solutions possibles afin de déterminer une solution avec une qualité satisfaisante, c'est-à-dire une valeur de fonction objectif quasi optimale, ceci est réalisé en transformant une solution actuelle *S* en une autre *S'*et en répétant ce processus pour un certain nombre d'itérations. Les opérations correspondantes qui sont nécessaires pour effectuer une telle transformation sont appelées migrations. Dans le cadre d'une gestion de projet limitée par les ressources, la solution actuelle peut, par exemple, être représentée sous la forme d'un ordonnancement réalisable (Diaz & al. 2018) .Ensuite, on applique une modification des heures de début d'un sous-ensemble de telle sorte qu'un autre ordonnancement réalisable soit obtenu. La collection de solutions réalisables qui peuvent être construites à partir de la solution actuelle *S* par un seul mouvement définit leur voisinage. Lorsque la recherche est guidée par une procédure de descente pour un problème de minimisation, seuls les migration qui permettent d'améliorer la valeur actuelle de la fonction objectif sont autorisés. La recherche se termine lorsqu'une solution de voisinage améliorée est atteinte.

II.13 Conclusion

Dans ce chapitre, un aperçu des aspects théoriques de l'optimisation et de la modélisation ainsi que des approches de solution pour le problème d'ordonnancement sont présentés. Bien qu'il n'y ait pas d'approche globale unique pour gérer tous les problèmes possibles, et que la majorité des modèles proposés sont caractérisés comme spécifiques au problème, une catégorisation est possible en identifiant les caractéristiques communes de base. Différentes instances de problèmes, ainsi que différents modèles d'optimisation sont classés, présentant leurs caractéristiques les plus importantes. En outre, un aperçu des formulations de modélisation dans des études de cas réels est présenté. Il est conclu que seul un nombre limité des contributions sont en mesure de résoudre des problèmes industriels à grande échelle. Une variété de techniques de décomposition et de règles heuristiques sont appliquées avec les modèles de programmation mathématique, et par conséquent ; de bonnes solutions mais sous-optimales sont obtenues. Un autre défi de planification à relever est la visualisation efficace des solutions qui permettent également des changements automatisés. Dans ce qui suit nous nous sommes concentrés sur la planification des ateliers de type Job-Shop (JSP) incluant la version de base qui était la plus générale et la plus complexe tout en étudiant les différentes techniques pour minimiser le temps total de planification

III. Chapitre III Modèle d'ordonnancement JOB SHOP (JSP) et méthodes de résolutions

III.1 Introduction

D'une manière générale, l'ordonnancement se réfère à un processus de prise de décision afin de résoudre un problème du monde réel comme cela a été discuté dans le chapitre précédent.

Depuis plusieurs années, le fameux problème d'ordonnancement des ateliers Job-shop et Flow-Shop retient l'attention des chercheurs en recherche opérationnelle, ingénierie et informatique. Ces dernières années, il y a eu un regain d'intérêt pour l'heuristique et la méta-heuristique pour résoudre ce problème. Ce chapitre est consacré aux modèles d'ordonnancement (formels) et aborde les éléments composant un modèle d'ordonnancement (Jobs / machines / opérations, contraintes, critères). Les principaux problèmes liés à la construction de modèles d'ordonnancement ; en particulier, les bases de la modélisation des problèmes de type Job-Shop seront évoqués dans ce chapitre ainsi que leurs méthodes de résolution.

III.2 Modèle d'ordonnancement JOB SHOP (JSP)

Selon Blazewicz,Ecker, Schmidt, & Weglarz (1994) les problèmes d'ordonnancement peuvent être définis au sens large comme «des problèmes d'allocation des ressources dans le temps pour effectuer un ensemble de tâches». Dans la littérature, le problème d'ordonnancement est très divers (Brucker 1998 ; French 1982), et qui a pris une part importante est le problème d'ordonnancement de type Job-Shop (Figure III.1) qui peut être défini comme une détermination des séquences d'opérations sur les machines afin de minimiser le temps de fabrication (makespan) et tout cela est basé sur les hypothèses suivantes (Cheng, Gen & Tsujimura 1996):

H1. Le temps de traitement de chaque opération est prédéterminé.
H2. Les opérations ne peuvent pas être interrompues.
H3. Une tâche ne peut visiter la même machine qu'une seule fois.
H4. Une seule tâche est affectée à une machine à la fois.
H5. Chaque machine ne traite qu'une seule tâche à la fois.
H6. Il n'y a pas de contraintes de priorité entre les opérations des différentes tâches.
H7. Ni le temps d'exécution ni les dates d'échéance ne sont spécifiées.

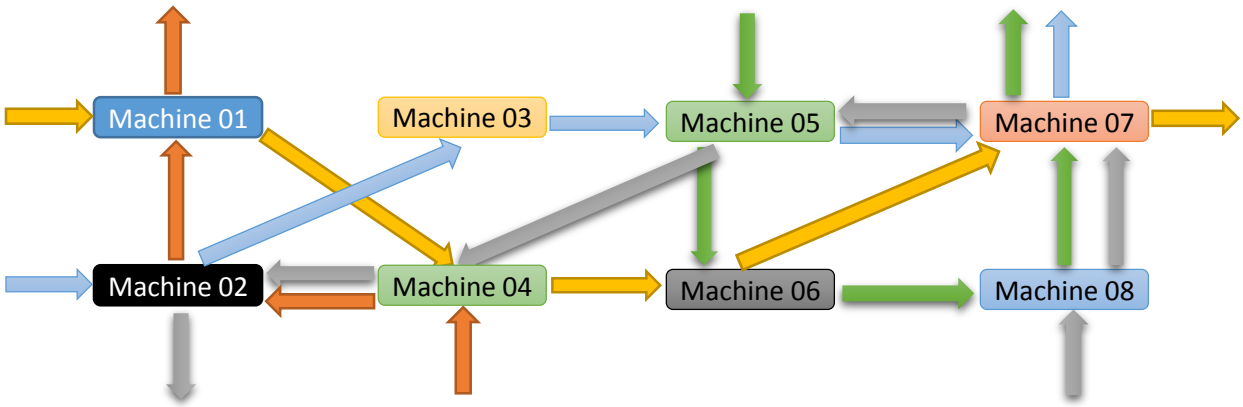


Figure III.1 Ateliers à cheminements multiples

III.3 Formulation mathématique de JSP

La formulation mathématique du problème Job-Shop (Özgüven, Özbakır & Yavuz 2010) s'appuie sur la notation suivante :

Indice :

i, l: index des Jobs, i, l = 1,2, . . . ,n

j, h: index des machines, j, h = 1,2, . . .,m

k: index des opérations, k = 1,2, . . .,m

Paramètre :

n: nombre total de Jobs

m: nombre total de machines

t_M: Makespan

M_j: la j^{ème} machine

J_i: le i^{ème} Job, i = 1,2,. . . , n

o_{ikj}: la k^{ème} opération du Job J_i opérée sur la machine M_j

p_{ikj}: temps de traitement de l'opération o_{ikj}

Variables de décision :

t_{ikj}: temps d'achèvement de l'opération o_{ikj} sur la machine M_j pour chaque Job J_i

Le problème que nous traitons est de minimiser le temps d'exécution total (fonction objectif), de sorte que le problème pourrait être décrit comme suit :

$$\min t_M = \max_{ikj}\{t_{ikj}\}$$

$$t_{ikg} + p_{ik+1j} \leq t_{ik+1j} \geq 0 \; \forall \; i,k,g,j \qquad \textbf{III.1}$$

La contrainte à l'équation est la contrainte de priorité d'opération, la k^{ème} opération du job i doit être traitée avant la (k+1)^{ème} opération du même job. Le chronogramme du modèle est illustré dans la Figure III.2.

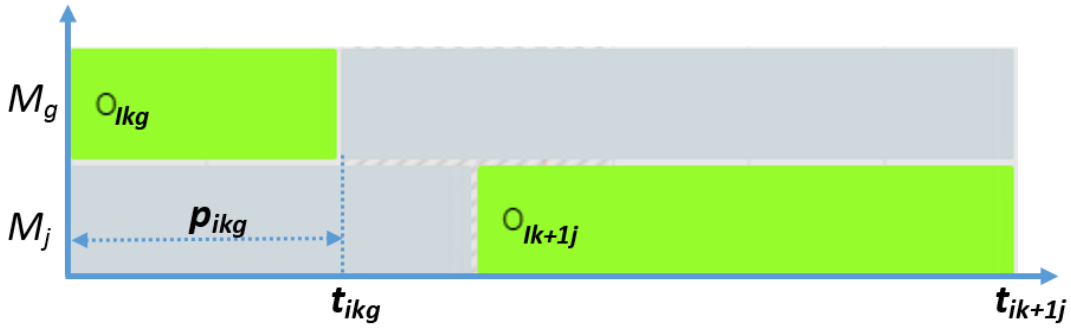


Figure III.2 Diagramme de temps/contrainte

III.4 Complexité de JSP

Ce problème a déjà été confirmé comme l'un des problèmes NP-HARD ((n!)^m solutions possibles). Il y a n Jobs et m machines à planifier ; en outre, chaque tâche est composée d'un ensemble d'opérations et l'ordre d'opération sur les machines est prédéfini, et chaque opération est caractérisée par la machine requise et le temps de traitement fixe (Cheng, Gen & Tsujimura 1996 ; Gen & Cheng 2002 ; Jain & Meeran 1998) .

La plupart des variantes du JSP, à l'exception de quelques formulations avec un nombre de machines ou de Jobs limité à 1 ou 2, sont connues pour être NP-hard (Brucker 1998 ; French 1982). En particulier, les JSP avec le nombre de machines m > 2 utilisant les critères de performance Cmax et FΣ sont NP-hard au sens fort (Garey & Johnson 1978 ; Garey, Johnson & Sethi 1976) . Étant donné que les problèmes Cmax peuvent être réduits de manière triviale aux problèmes Tmax et Lmax, et que les problèmes FΣ peuvent être réduits de manière triviale aux problèmes TΣ et LΣ, ces problèmes sont également NP-hard au sens fort. Une forte dureté (NP-hardness) d'un problème implique qu'il est impossible de créer une heuristique qui garantit de trouver une solution pour laquelle l'erreur relative est limitée par : $\frac{mesure\ des\ performances\ de\ la\ solution\ trouvée}{mesure\ des\ performances\ de\ la\ solution\ optimale} \leq 1 + \varepsilon$ et qui s'exécute en temps polynomial à la fois dans la taille du problème et 1 / ε. Ce résultat est très important, car il indique que des algorithmes d'approximation efficaces avec des performances garanties ne devraient pas être attendus pour ces problèmes à moins que P = NP. Pour cette raison, la plupart des recherches axées sur la recherche des solutions presque optimales ont été orientées vers des algorithmes d'énumération implicites (séparation et évaluation, ou branch and bound en anglais), des méthodes d'amélioration locale (goulot d'étranglement, ou shifting bottleneck en anglais) et des méthodes de recherche Meta-heuristique telles que les algorithmes génétiques, la recherche tabou et le recuit simulé.

III.5 Modélisation graphique du job shop

On peut distinguer plusieurs façons de modéliser ce problème. La modélisation la plus répondue et connu est celle par graphe disjonctif, tandis que les plus anciennes sont des modélisations en programmation mathématique. Mais il existe aussi des modélisations algébriques, par des graphes potentiels-tâches, diagrammes de Gantt, à l'aide de réseaux de pétri, modélisations basées sur UML...etc.

Roy et Sussmann ont été les premiers à proposer la représentation du graphe disjonctif (Roy & Sussmann 1964), et Balas (1969) a été le premier à appliquer une approche énumérative basée sur le graphe disjonctif. Depuis lors, de nombreux chercheurs ont essayé différentes stratégies pour résoudre ce problème. Dans ce cas de représentation, Le job shop est modélisé par graphe disjonctif G (X, C*D) où :

- **X** représente l'ensemble des sommets, chaque opération correspond à un sommet avec deux opérations fictives S (source) et P (puits) qui désignent le début et la fin de l'ordonnancement.
- **C** représente l'ensemble des arcs conjonctifs représentant les contraintes d'enchaînement des opérations d'une même tâche (gammes opératoires). Il y'a un arc entre tous les sommets (i,j) , (ij+1) pour i=1...n et j=1...ni ; n est le nombre de tâches, ni est le nombre d'opérations de la tâche i.
- **D** représente l'ensemble des arcs disjonctifs associés aux contraintes et aux conflits d'utilisation d'une machine.

Pour construire un ordonnancement admissible à l'aide d'un graphe, il est nécessaire de choisir pour chaque paire d'arcs, l'arc qui déterminera l'ordre de passage des deux opérations sur la machine.

Comme est figuré sur l'image III.3, le graphe disjonctif-conjonctif d'un problème job shop avec 3 jobs et 3 machines définit comme suit :

J1[M1:4; M2:3; M3:3]; J2[M1:1; M3:5; M2:3]; J3[M2:2; M1:4; M3:1]

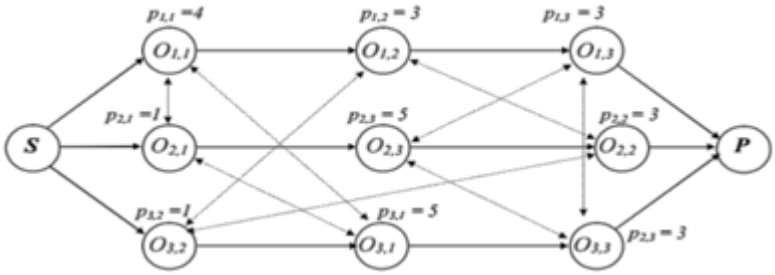


Figure III.3 Exemple d'un graphe disjonctif

L'ensemble des paires de disjonction associées à une même machine forment une clique de disjonction.

Le graphe disjonctif sous cette forme, modélise un problème d'ordonnancement. Pour construire un ordonnancement admissible sur ce graphe, il suffit de choisir pour chaque paire d'arcs, l'arc qui déterminera l'ordre de passage des deux opérations sur la machine ; c'est à dire arbitrer chacune des disjonctions. L'arbitrage doit être complet (toutes les disjonctions sont arbitrées), et compatible.

Pour un ordonnancement donné, certaines notions bien évidentes sur le graphe disjonctif sont intéressantes à évoquer :

L'opération critique :

Soit un ordonnancement actif, une opération est dite critique, si elle provoque nécessairement l'augmentation du makespan de l'ordonnancement, lorsqu'elle est retardée (alors que l'ordre d'exécution des opérations défini par l'ordonnancement ne change pas).

Le chemin critique :

Le chemin critique est une suite d'opérations critiques liées par des relations de précédence. La longueur d'un chemin critique est égale à la somme des durées des opérations qui le composent. Pour un ordonnancement donné, il peut exister plus d'un chemin critique.

Le bloc critique :

Le bloc critique est une succession d'opérations critiques s'exécutant sur la même machine. Tout chemin critique peut se décomposer en plusieurs blocs critiques.

III.6 Les méthodes de résolution

Nous avons vu précédemment que le problème d'ordonnancement de Job Shop à l'exception de certains cas, est NP-difficile. Différentes méthodes de résolution essentiellement approchées sont largement utilisées pour appréhender les problèmes de Job Shop NP-difficiles.

Récemment, de nombreux chercheurs ont essayé d'adapter différentes approches méta-heuristiques pour obtenir une solution quasi optimale de JSP. Monch et Driessel ont envisagé des versions distribuées d'une heuristique de goulot d'étranglement modifiée pour résoudre des Job-Shops complexes. Nowicki et Smutnicki fournissent un nouvel algorithme de Recherche Tabou (TS) approximatif qui est basé sur le phénomène de la grande vallée, et utilise certains éléments de la technique dite recomposition de chemin ainsi que de nouvelles propriétés théoriques des voisinages (Nowicki & Smutnicki 2005). Tavakkoli-Moghaddam et al utilisent l'approche du réseau de neurones (NN) pour générer des solutions réalisables initiales et adapté un algorithme de recuit simulé (SA) pour améliorer la qualité et les performances de la solution (Tavakkoli-Moghaddam, Jolai, Vaziri, Ahmed, & Azaron 2005).

Pour améliorer l'efficacité de la recherche de meilleures solutions, certaines recherches techniques locales spéciales ont été adaptées dans JSP. Ida & Osawa (2005) ont réformé le décalage à gauche traditionnel pour minimiser le temps d'inactivité et ont formulé un algorithme appelé Eshift. Gonc¸alves et al. (2005) ont proposé une autre technique basée sur le chemin critique pour valider l'espace de recherche, et ont échangé les opérations entre les différents blocs critiques, et cette approche peut également améliorer l'efficacité de l'algorithme pour trouver une solution active.

Une présentation de la totalité des méthodes et de leurs variantes apparaît une tâche difficile. Nous présentons dans cette partie, les grandes orientations des méthodes, en essayant de repérer les travaux de recherche essentiels.

III.7 Heuristique conventionnelle pour JSP

JSP est un problème pratique très important au quotidien, c'est l'un des problèmes d'optimisation combinatoire les plus difficiles, il est donc naturel de chercher des méthodes d'approximation qui produisent un ordonnancement dans un temps acceptable. Les méthodes heuristiques pour résoudre ce type de problème peuvent être catégorisées en deux classes Single-pass heuristic & Multi-pass heuristic.

L'heuristique de la première classe crée simplement une solution complète unique en fixant une opération dans l'ordonnancement à la fois en fonction des *«Règles de répartition prioritaires : priority dispatching rules »*. Il existe de nombreuses règles pour choisir une opération dans un sous-ensemble spécifié à planifier ensuite. Cette heuristique est rapide et trouve généralement des solutions qui ne sont pas trop mauvaises. De plus, l'heuristique susdite peut être utilisée à plusieurs reprises pour construire une heuristique de la deuxième classe plus sophistiquée afin d'obtenir de meilleurs résultats à un coût de calcul supplémentaire.

III.7.1 Heuristiques basées sur les règles de priorité

Les règles de priorité sont probablement les plus fréquemment appliquées pour résoudre les problèmes d'ordonnancement dans la pratique en raison de leur facilité de mise en œuvre et de leur faible complexité. Les algorithmes de Giffler et Thompson peuvent être considérés comme la base commune de toutes les heuristiques basées sur des règles de priorité (Storer, Wu & Vaccari 1992). Giffler et Thompson ont proposé deux algorithmes pour générer un ordonnancement : les procédures de génération d'ordonnancement actif et sans-délai (Giffler & Thompson 1960).Les ordonnancement actifs forment un ensemble beaucoup plus large et incluent les ordonnancements sans-délai comme sous-ensemble. La procédure de génération de Giffler et Thompson est une approche arborescente. Les nœuds de l'arbre correspondent à des solutions partielles, les arcs représentent les choix possibles et les feuilles de l'arbre sont l'ensemble des ordonnancements énumérés. Pour une solution partielle donnée, l'algorithme identifie essentiellement tous les conflits de traitement, c'est-à-dire les opérations en concurrence pour la même machine, et une procédure d'énumération est utilisée pour résoudre ces conflits de toutes les manières possibles à chaque étape. En revanche, l'heuristique résout ces conflits avec des règles de priorité, c'est-à-dire spécifie une règle pour sélectionner une opération parmi les opérations en conflit.

Les procédures de génération fonctionnent avec un ensemble d'opérations programmables à chaque étape. Les opérations d'ordonnancements sont des opérations non organisées mais avec des prédécesseurs immédiatement planifiés et cet ensemble peut être simplement déterminé à partir de la structure de priorité. Le nombre d'étapes pour une procédure en un seul passage est égal au nombre d'opérations m × n. À chaque étape, une opération est sélectionnée pour s'ajouter à la solution partielle. Les conflits entre les opérations sont résolus par des règles de priorité. Suite aux notations de Baker (1974), nous avons :

PS_t = une solution partielle contenant t opérations d'ordonnancements,

S_t = l'ensemble des opérations programmables à l'étape t, correspondant à un PSt donné,

σ_i = le moment auquel l'opération $i \in S_t$ a pu être démarrée,

φ_i = le moment auquel l'opération $i \in S_t$ a pu être terminée.

Pour une solution partielle active donnée, l'heure de début potentielle σ_i est déterminée par l'heure de fin du prédécesseur direct de l'opération i et la dernière heure de fin sur la machine requise par l'opération i. La plus grande de ces deux valeurs est σ_i. Le temps de terminaison potentiel φ_i est simplement $\sigma_i + t_i$, où t_i est le temps de traitement de l'opération i. La procédure pour générer un ordonnancement actif fonctionne comme suit :

Procédure Active

Entrée : données JSP

Sortie : un ordonnancement complet

1. Initialisation $t = 0$ et $PS_t = nul$ S_t inclut toutes les opérations sans prédécesseurs.
   Tant que ! Ordonnancement complet soit généré
2. Déterminer $\varphi_t^* = \min_{i \in S_t}\{\varphi_i\}$ et la machine m* sur laquelle φ_t^* pourrait être réalisée.
3. Pour chaque opération $i \in S_t$ qui nécessite la machine m^* et pour laquelle $\sigma_i < \varphi_t^*$, calculez un indice de priorité selon une règle de priorité spécifique. Recherchez les opérations avec le plus petit index et ajoutez cette opération à PS_t dès que possible, créant ainsi une nouvelle solution partielle PS_{t+1}.
4. Pour PS_{t+1}, mettez à jour l'ensemble de données comme suit :
   a. supprimer les opérations i de S_t
   b. forme S_{t+1} en ajoutant le successeur direct de l'opération j à S_t
   c. $t++$

Fin Tant que

Si nous remplaçons les étapes 2 et 3 de l'algorithme par celles données dans l'algorithme suivant, la procédure peut générer un ordonnancement sans-délai, comme illustré ci-dessous :

Procédure Sans-délai

Entrée : données JSP

Sortie : un ordonnancement complet

1. Initialisation $t = 0$ et $PS_t = nul$ S_t inclut toutes les opérations sans prédécesseurs.
   Tant que ! Ordonnancement complet soit généré
2. Déterminer $\sigma_t^* = \min_{i \in S_t}\{\sigma_i\}$ et la machine m* sur laquelle σ_t^* pourrait être réalisée.
3. Pour chaque opération $i \in S_t$ qui nécessite la machine m^* et pour laquelle $\sigma_i < \sigma_t^*$, calculez un indice de priorité selon une règle de priorité spécifique. Recherchez les opérations avec le plus petit index et ajoutez cette opération à PS_t dès que possible, créant ainsi une nouvelle solution partielle PS_{t+1}.
4. Pour PS_{t+1}, mettez à jour l'ensemble de données comme suit :
   a. supprimer les opérations i de S_t
   b. forme S_{t+1} en ajoutant le successeur direct de l'opération j à S_t
   c. $t++$

Fin Tant que

Le problème restant est d'identifier une règle de priorité efficace. Pour un résumé et une discussion approfondis, voir Panwalkar et Iskander (1977), Haupt (1989) et Blackstone et al. (1982).

Le tableau III.1 contient certaines des règles de priorité couramment utilisées dans la pratique.

Règle	Description
SPT (Shortest Processing Time)	Sélectionnez l'opération avec le temps de traitement le plus court
LPT (Longest Processing Time)	Sélectionnez l'opération avec le temps de traitement le plus long
LRT (Longest Remaining Processing Time)	Sélectionnez l'opération appartenant à la tâche avec le temps de traitement restant le plus long
SRT (Shortest Remaining Processing Time)	Sélectionnez l'opération appartenant à la tâche avec le temps de traitement restant le plus court
LRM (Longest ReMaining excluding the operation under consideration)	Sélectionnez l'opération appartenant à la tâche avec le temps de traitement restant le plus long, à l'exclusion de l'opération considérée

Tableau III.1 La liste des règles de priorité du problème de type Job-Shop

Exemple avec la règle Shortest Processing Time (SPT)

En utilisant la table de séquence (Tableau III.3) pour le processus de sélection d'opération, la solution de l'exemple donné (Tableau III.2) peut être construite comme suit, et la figure III.4 illustre un diagramme de Gantt montrant l'ordonnancement pour la règle de répartition SPT.

	1	**2**	**3**	**Machines**		
J1	15	20	11	M3	M1	M2
J2	14	19	8	M3	M2	M1
J3	7	17	21	M2	M1	M3

Tableau III.2 Exemple d'un problème de trois machines et trois Jobs.

$S = \{(o_{ikj}(t_{tkj} - t^F_{ikj})\}$:

$S = \{o_{312}(t_{312} - t^F_{312}), o_{213}(t_{213} - t^F_{213}), o_{113}(t_{113} - t^F_{113}), o_{321}(t_{321} - t^F_{321}), o_{222}(t_{222} - t^F_{222}), o_{231}(t_{231} - t^F_{231}), o_{121}(t_{121} - t^F_{121}), o_{132}(t_{132} - t^F_{132}), o_{333}(t_{333} - t^F_{333})\}$

$S = \{o_{312}(0-7), o_{213}(0-14), o_{113}(14-29), o_{321}(7-24), o_{222}(14-33), o_{231}(33-41), o_{121}(41-61), o_{132}(61-72), o_{333}(29-50)\}$

III.2

Operations	P_{ikj}	O_{ikj}^{*}	**Solution**
{113, 213, 312}	{15, 14, 7}	312	{312,}
{113, 213, 321}	{15, 14, 17}	213	{312, 213}
{113, 222, 321}	{15, 19, 17}	113	{312, 213, 113}
{121, 222, 321}	{20, 19, 17}	321	{312, 213,113, 321}
{121, 222, 333}	{20, 19, 21}	222	{312, 213,113, 321, 222}
{121, 231, 333}	{20, 8, 21}	231	{312, 213,113, 321, 222, 231}
{121, 333}	{20, 21}	121	{312, 213,113, 321, 222, 231, 121}
{132, 333}	{11, 21}	132	{312, 213,113, 321, 222, 231, 121, 132}
{333}	{21}	333	{312, 213,113, 321, 222, 231, 121, 132, 333}

Tableau III.3 Solution SPT.

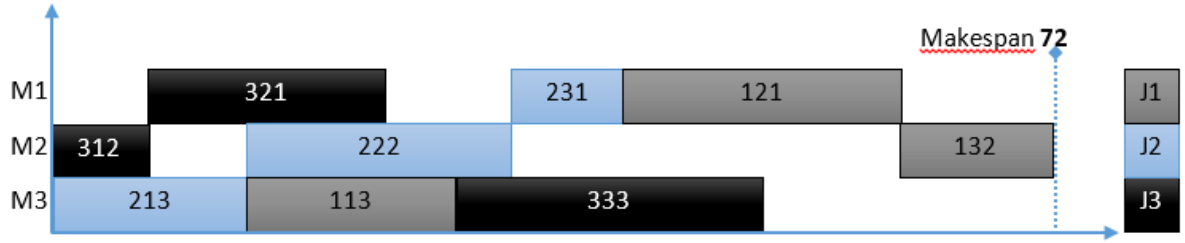


Figure III.4 Diagramme de Gantt pour la règle SPT

III.7.2 Heuristique et distribution aléatoire

L'heuristique en un seul passage se limite à construire une solution unique, l'heuristique multi-passe essaie d'obtenir des meilleures solutions en générant beaucoup d'entre elles, généralement au détriment d'un temps de calcul beaucoup plus élevé. Des techniques telles que la méthode branch-and-bound et la programmation dynamique peuvent garantir une solution optimale, mais ne sont pas pratiques pour les problèmes à grande échelle. L'heuristique aléatoire est une première tentative pour fournir des solutions plus précises (Baker 1974). L'idée de la répartition aléatoire est de commencer par des règles de répartition. À chaque sélection d'une opération à exécuter, choisissez la règle de répartition au hasard, répétée tout au long d'une génération d'un ordonnancement. Répétez l'ensemble du processus plusieurs fois et choisissez le meilleur résultat. La procédure pour générer un ordonnancement actif est donnée comme suit :

Procédure Active

Entrée : données JSP

Sortie : un ordonnancement complet

MS = nul //Meilleur solution

ST = nul //Solution trouvé

Tant que ST meilleur ou égale à BS

1. Initialiser le meilleur ordonnancement par nul.

   Initialiser t = 0 et commencez par PS_t comme nul.

   Initialiser S_t par toutes les opérations sans prédécesseurs.

   Tant que ! Ordonnancement complet soit généré
2. Déterminer $\varphi_t^* = \min_{i \in S_t}\{\varphi_i\}$ et la machine m^* sur laquelle φ_t^* pourrait être réalisée.
3. Sélectionnez une règle de répartition au hasard. Pour chaque opération $i \in S_t$ qui nécessite la machine m^* et pour laquelle $\sigma_i < \varphi_t^*$ calculez un indice de priorité selon la règle spécifiée. Recherchez les opérations avec le plus petit index et ajoutez cette opération à PS_t dès que possible, créant un nouveau ordonnancement partiel PS_{t+1}.
4. Pour PS_{t+1}, mettez à jour l'ensemble des données comme suit :

   a. Supprimer les opérations *i* de S_t

   b. Formuler S_{t+1} en ajoutant le successeur direct de l'opération *j* à S_t

   c. $t++$;

   Fin Tant que.

   Fin Tant que.

III.7.3 Procédure de goulot d'étranglement (Shifting Bottleneck Procedure)

L'heuristique du goulot d'étranglement d'Adams et al. (1987) est probablement la procédure la plus puissante connue à ce jour parmi toutes les heuristiques pour le JSP. Il séquence les machines une à une, successivement, en prenant à chaque fois la machine identifiée comme goulot d'étranglement parmi les machines non encore séquencées. Chaque fois qu'une nouvelle machine est séquencée, toutes les séquences précédemment établies sont localement ré-optimisées. L'identification des goulots d'étranglement et les procédures de ré-optimisation locales sont basées sur la résolution répétée d'un certain problème d'ordonnancement d'une seule machine qui est une relaxation du problème d'origine. La méthode de résolution des problèmes d'une seule machine n'est pas nouvelle, bien qu'ils aient considérablement accéléré le temps nécessaire pour générer ces problèmes. Au lieu de cela, la principale contribution de leur approche est la façon d'utiliser cette relaxation pour décider de l'ordre dans lequel les machines doivent être séquencées. Ceci est basé sur l'idée classique de donner la priorité aux machines à goulot d'étranglement. Soit M_0 l'ensemble des machines déjà séquencées (M_0 = nul au départ). Un bref énoncé de la procédure de goulot d'étranglement est le suivant :

Procédure
Entrée : données JSP
Sortie : un ordonnancement complet
REPETER

1. Identifier une machine m parmi les machines privées de M_0 et la séquencer de manière optimale. Mettez $M_0 \leftarrow M_0 \cup \{m\}$.
2. Ré-optimisez la séquence de chaque machine $k \in M_0$, tout en conservant les autres séquences fixes.

JUSQU'A $M_0 = M$

III.8 Méthodes méta-heuristiques pour la résolution du JSP

Dès l'avènement des Méta-heuristiques comme approche de résolution importante des problèmes difficiles offrant un bon compromis entre le coût de résolution et la qualité des solutions, le problème d'ordonnancement de Job Shop constitue l'un des domaines d'application de ces méthodes intensivement investiguées.

Les premiers algorithmes méta-heuristiques génériques incluent l'algorithme génétique et la méthode de recherche Tabou (Jones & al. 2002). Beaucoup d'autres algorithmes, tels que l'algorithme de colonies de fourmis, l'algorithme d'optimisation par essaim de particules, l'algorithme à évolution différentielle, l'algorithme des lucioles, etc., qui sont tous des méthodes d'imitation de la nature ou des cercles biologiques, appliqués dans le problème d'ordonnancement Job-shop et a donné de bons résultats.

L'algorithme génétique (GA) est l'une des méta-heuristiques populaires qui opèrent directement sur la structure du problème sans dérivation ni limitation de la continuité de fonction. Les AG ont également le parallélisme implicite inhérent et la capacité de recherche globale et peuvent ajuster les directions de recherche automatiquement et de manière auto-adaptative. Le AG d'origine a été utilisé pour JSP par (Davis 1985) qui a formé une séquence d'opérations préférée pour chaque machine dans laquelle AG est utilisée comme méthode indirecte. Après cela, divers efforts ont été faits pour adapter les algorithmes génétiques afin de résoudre différents JSP et ont amélioré les performances de la recherche génétique en intégrant d'autres méthodes heuristiques. (Falkenauer & Bouffouixm 1991) ont amélioré cette méthode en codant toutes les opérations de chaque machine comme une chaîne de symboles préférée. (Kuczapski & al. 2015) ont proposé une méthode pour générer la bonne population initiale en faisant évoluer les répartitions basées sur les règles de priorité jusqu'à l'arrivée de solutions finales optimales. (Jalilvand-Nejad & Fattahi 2015) ont proposé un modèle d'ordonnancement linéaire intégré pour le FJSP (problème job-shop flexible) ainsi qu'une comparaison avec le SA (recuit simulé) et a montré que le AG est plus efficace que le second. (Zhang & Chong 2016) ont proposé un algorithme génétique multi-objectif avec deux stratégies d'amélioration locale spécifiques au problème pour résoudre un problème d'optimisation bi-objectif. (Zhang & al. 2017) ont pris en compte le temps d'achèvement le plus court et l'utilisation équilibrée des

machines, et ont proposé un algorithme génétique multi-population basé sur l'ordonnancement multi-objectifs de job-shop flexible.

La recherche tabou (TS) est une technique d'optimisation itérative globale et également l'une des méta-heuristiques qui applique une recherche intelligente et une mémoire pour aider à atteindre l'optimum global plutôt que d'être piégée à l'optimum local. La première étude de TS a été réalisée par (Glover 1986). Bien que TS soit une procédure de recherche simple, elle peut interdire les changements qui sont identiques ou similaires aux solutions précédemment obtenues en fonction des informations stockées dans la mémoire, puis éviter les solutions optimales locales.

(Meeran & Morshed 2012) ont consacré leurs efforts à combiner AG avec TS pour gérer JSP en mettant en évidence l'avantage de la recherche parallèle globale de la première approche et l'avantage d'une évitement optimal local de la dernière approche. (Peng & al. 2015) ont intégré un processus TS avec la technique de recomposition de chemin (PR) pour obtenir de meilleures solutions pour résoudre JSP en construisant un chemin qui pourrait connecter les solutions initiatrices à des solutions optimisées et sélectionner des solutions d'une façon meilleur et efficace. (Li & Gao 2016) ont aussi proposé un algorithme hybride efficace qui hybride le AG et le TS pour le FJSP dans le but de minimiser le Makespan.

L'optimisation des colonies de fourmis (ACO) qui imite le processus de recherche de colonies de fourmis est une autre méta-heuristique présentée par (Colorni & al. 1991), qui est également le premier chercheur à utiliser cette approche pour résoudre le problème NP-Hard du voyageur de commerce. (Bullnheimer & al. 1999) ont appliqué cette approche pour le problème de tournées de véhicules. (Merkle & al. 2002), (Blum & Sampels 2004) ont utilisé l'ACO pour résoudre différents problèmes d'ordonnancement. Certains chercheurs améliorent les procédures optimisées en intégrant d'autres algorithmes à ACO pour obtenir la meilleure qualité de solution ou une meilleure efficacité, comme la recherche tabou (Huang & Liao 2008), la recherche en faisceau (Blum 2005), algorithme basé sur le système immunitaire artificiel (Xue & al. 2015), l'approche pareto (Zhao & al. 2015). Comme l'algorithme de la colonie de fourmis est une sorte d'algorithme parallèle auto-organisé avec une rétroaction positive, certains chercheurs se sont consacrés ces dernières années à traiter le JSP dynamique par l'ACO comme (Saidi-Mehrabad & al.2015; Huang & al.2013). (Wang & al. 2017) ont proposé un algorithme ACO amélioré, les principales améliorations comprennent la sélection des règles de la machine, l'initialisation du mécanisme de distribution uniforme pour les fourmis, la modification du mécanisme de guidage des phéromones, la sélection de la méthode des nœuds et la mise à jour du mécanisme des phéromones. (Huang & Yu 2017) ont proposé un algorithme ACO efficace, cinq améliorations sont apportées aux algorithmes proposés : un nouveau type de phéromone et une fonction heuristique gourmande ; trois nouvelles fonctions des règles de transition; un algorithme agile de recherche locale pour l'amélioration de la qualité de la solution, un nouveau mécanisme de mutation, et un algorithme basé sur l'optimisation par essaims de particules (PSO) pour le réglage adaptatif des paramètres.

L'optimisation par essaims de particules (PSO) résultant du comportement des proies des oiseaux est une technique méta-heuristique d'évolution informatique, qui a été initialement proposée par (Kennedy & Eberhart 1995). Basé sur l'observation de la régularité des activités

proies des oiseaux volants, le principe est le partage des informations individuelles dans la population afin d'obtenir la solution optimale. Comparé à l'algorithme génétique, en raison de l'absence d'opérations de croisement et de mutation et de peu de paramètres à ajuster, l'avantage de PSO est qu'il est facile à mettre en œuvre. L'inconvénient inhérent de PSO est l'absence de convergence globale en raison d'une forte réduction des valeurs.

(Xia & Wu 2005) ont combiné PSO qui attribue les opérations sur les machines à SA qui planifie les opérations sur chaque machine pour résoudre FJSP de manière hiérarchique. Afin de surmonter l'inconvénient de PSO, les chercheurs ont récemment exploré des méthodes combinatoires de PSO avec d'autres pour résoudre JSP et FJSP, comme (Baykaso˘glu & al. 2014), (Yin & al. 2015), (Nouiri & al. 2015), (Teekeng & al. 2016), (Huang & al. 2016). Récemment, (Singh & Mahapatra 2016) ont utilisé un opérateur d'algorithme génétique dans l'opération de mutation et la cartographie logistique pour générer des nombres chaotiques plutôt que des nombres aléatoires. Les nombres chaotiques signifient généralement des nombres aléatoires et pseudo-aléatoires avec de bonnes propriétés statistiques. Comparée à plusieurs algorithmes populaires, cette méthode est plus efficace pour minimiser le Makespan. (Muthiah & al. 2016) ont proposé l'hybridation des techniques de l'algorithme des colonies d'abeilles artificielles (ABC) et PSO pour optimiser le Makespan. Dans le même esprit, (Nouiri & al. 2017) ont proposé une optimisation par essaims de particules en deux étapes pour résoudre le problème en supposant qu'il n'y a qu'une seule panne.

L'évolution différentielle (DE) a été proposée par (Storn & Price 1995) et est une technique méta-heuristique évolutive en imitant les organismes d'évolution et en répétant l'itération pour réserver les individus qui se sont adaptés à l'environnement. Par rapport à GA, cette approche peut être réalisée plus facilement et converger plus efficacement pour une optimisation continue.

(Ponsich & al. 2009) ont montré que le DE seul ne pouvait pas trouver de solution aussi bien que AG ou TS et la raison peut être que DE a un manque d'intégrité et d'auto-adaptation sur l'approche de représentation de permutation et l'opérateur de mutation pour les problèmes discrets ou un JSP, bien que c'est pratiquement bien pour un problème d'optimisation continu. Une approche améliorée axée sur l'hybridation d'une approche de recherche des voisinages avec DE pour rendre la recherche locale plus efficace. (Ponsich & Coello 2013) ont combiné l'évolution différentielle et l'approche de recherche Tabou pour résoudre les problèmes JSP et ont montré que cet algorithme DE / TS était comparable aux autres techniques avancées actuelles. La solution optimale a également indiqué une efficacité élevée commune de cet algorithme par de nombreux exemples, en particulier pour la plupart des problèmes JSP de taille médiane, et il pourrait rechercher la solution avec une répétabilité satisfaisante. (Zhao & al. 2016) ont trouvé un équilibre entre l'exploration globale et l'efficacité de l'espace de recherche local en intégrant une procédure de recherche de voisinage accélérée pour rechercher des chemins clés dans l'algorithme d'évolution différentielle pour résoudre FJSP. (Zhang & al. 2016) ont proposé un algorithme d'évolution différentielle chaotique (CDEA) avec un critère de minimisation de Makespan, dans le CDEA, la cartographie est utilisée pour générer des nombres chaotiques pour l'initialisation car il est utile de diversifier la population du CDEA et

d'améliorer ses performances dans la prévention de la convergence prématurée vers les minima locaux.

L'algorithme des lucioles (FA) est une approche plus récente présentée à l'origine par (Yang 2008) et qui provient du comportement des populations de lucioles. (Łukasik & ˙ Zak 2009) ont proposé de poursuivre les recherches sur l'algorithme Firefly pour résoudre les problèmes d'optimisation continue.

Pour les problèmes d'optimisation continue et les problèmes NPhard continus, il est très efficace. Cependant, cette approche ne peut pas être appliquée pour résoudre directement les problèmes d'optimisation discrets car son processus d'apprentissage est basé sur le nombre réel. Pour résoudre les problèmes discrets, un ensemble d'approches de conversion des fonctions continues doit être construit, par exemple l'attractivité, la distance et le mouvement doivent être transformés en fonctions discrètes. L'algorithme discret des lucioles (DFA) a été introduit par (Sayadi & al. 2010) pour résoudre les problèmes d'ordonnancement Flow-shop. (Khadwilard & al. 2012) ont combiné DFA avec les règles SPV pour traiter les problèmes Flow-shop hybrides multi-objectifs. (Marichelvam & al. 2014) ont d'abord utilisé cet algorithme pour résoudre JSP et ont discuté du réglage des différents paramètres en fonction de leurs performances. (Karthikeyan & al. 2015) ont développé un DFA hybride avec une approche de recherche locale pour résoudre un FJSP multiobjectif en utilisant des règles pour la population initiale. Pour les méthodes hybrides, l'algorithme discret de luciole se concentre généralement sur une recherche approfondie de l'espace de la solution tandis que l'algorithme de recherche local est généralement utilisé pour re-planifier les résultats pour une convergence rapide et précise. (Marichelvam & Geetha 2016) proposent un algorithme hybride discret de luciole (HDFA) pour résoudre les FSP afin de minimiser le Makespan.

Optimisation des colonies d'abeilles (BCO), qui est un algorithme de recherche basé sur la population, introduit par (Pham & al. 2005) et proposé pour la première fois par (Chong & al. 2006) pour avoir résolu un problème Job-shop par le modèle de butinage des abeilles et avoir montré un peu moins de temps de réponse par rapport à d'autres approches heuristiques.

Le Tableau III.4 montre les avantages et les limitations des méthodes approximatives ainsi que les axes à améliorer au futur.

Méthodes	Avantages	Limites	Améliorations
Méthode constructive	Solution très rapide pour des problèmes moins complexes	Génération de solutions infaisables	• Éviter ou corriger les solutions infaisables.
AI	Efficace pour les problèmes dynamiques et pour la gestion des pannes	Apprentissage insatisfaisant	• Améliorer leur efficacité en termes de temps d'exécution et de consommation de ressources.
Méthodes de recherche locale	Une solution optimale peut être obtenue si suffisamment de temps est alloué	Besoin de temps	• Améliorer (paralléliser) ou hybrider les méthodes pour obtenir une bonne convergence vers l'optimum.
Méta-heuristiques	La plupart ont une bonne efficacité pour la recherche globale (hybridation)	Certaines méthodes convergent vers l'optimum local	

Tableau III.4 Avantage et les limitations des méthodes approximatives.

Un très grand nombre d'algorithmes méta-heuristiques hybrides et non hybrides pour JSSP ont été proposés dans le Tableau III.4.1. La liste des applications du JSSP comprend les algorithmes méta-heuristiques les plus connus, notamment les algorithmes évolutifs, l'optimisation des essaims de particules, l'optimisation des colonies de fourmis, la recherche par dispersion, les réseaux de neurones, le recuit simulé, la recherche taboue et de nombreuses formes de recherche exploratoire locale. La plupart de ces méthodes utilisent des algorithmes heuristiques plus simples pour générer une population initiale de solutions. Concernant la mise en œuvre d'algorithmes génétiques (AG) dans le JSSP, de nombreux chercheurs ont conclu que les AG génériques ne parviennent pas à étendre la recherche aux zones les plus prometteuses d'un quartier. C'est pourquoi les mises en œuvre réussies d'algorithmes génétiques intègrent généralement une procédure de recherche locale pour l'intensification de la recherche.

Algorithmes	Axes de recherches & inconvénients
Algorithmes génétiques (AG)	• hybridation avec une procédure de recherche de voisinage qui utilise également une série de règles de répartition des priorités dans le processus d'évolution génétique. • Inconvénient des opérateurs d'algorithmes génétiques à perturber la recherche dans des zones proches des optima locaux ou globaux

Optimisation des essaims de particules (PSO)	• La plupart des chercheurs ont présenté des méthodes de transformation spéciales ou des variations du PSO d'origine afin que cette méthode puisse également être appliquée au JSSP • PSO puisse généralement être utilisé avec des variables continues et, par conséquent, ne soit pas adapté aux variables discrètes •
Ant Colony Optimization (ACO)	• Tout comme l'algorithme PSO, les implémentations trouvées dans la littérature sont très récentes • hybridation avec une procédure de recherche de voisinage • Utilisation d'une nouvelle méthode de décomposition inspirée de la procédure de décalage des goulots d'étranglement et d'un mécanisme de ré-optimisation occasionnelle
Réseaux de neurones (NN)	• Des résultats médiocres ont été obtenus et seuls les travaux de Sabuncuoglu et Gurgun ont produit de bons résultats dans les tests
Le recuit simulé	• Le recuit simulé est incapable de trouver rapidement de bonnes solutions aux problèmes JSSP, parce qu'il s'agit d'une technique générique et sans mémoire. • Les recherches sont concentrées sur des versions hybrides
Tabu Search (TS)	• L'une des méta-heuristiques les plus puissantes pour le JSSP. Ceci est confirmé par le fait que la plupart des algorithmes pour le JSSP incluent une sorte de fonctionnalité de recherche tabou • La principale force de la recherche Tabou est l'utilisation de la mémoire qui accélère le processus de recherche d'espace de solution

Tableau III.5.1 Axes de recherches des méthodes approximatives.

III.9 Représentations génétiques pour JSP

En raison de l'existence des contraintes de précédence des opérations, JSP n'est pas aussi facile que le problème du voyageur de commerce (TSP) que nous pouvons le codifier avec une représentation de la nature. Il n'y a pas de bonne représentation avec un système d'inégalités pour les contraintes de précédence. Par conséquent, l'approche de pénalisation n'est pas facilement appliquée pour gérer ce type de contraintes. Orvosh et Davis (1994) ont montré que, pour de nombreux problèmes d'optimisation combinatoire, il est relativement facile de réparer un chromosome irréalisable ou illégal et la stratégie de réparation a en effet dépassé d'autres stratégies telles que la stratégie de rejet ou la stratégie de pénalisation. La plupart des chercheurs AG et JSP préfèrent adopter une stratégie de réparation pour gérer l'infaisabilité et l'illégalité. Un problème très important dans la construction d'un algorithme génétique pour un JSP est de

concevoir une représentation appropriée des solutions ainsi que des opérations génétiques spécifiques au problème afin que tous les chromosomes générés dans la phase initiale ou le processus évolutif produisent des solutions réalisables. Il s'agit d'une phase cruciale qui conditionne toutes les étapes ultérieures des algorithmes génétiques. Au cours des dernières années, les six représentations suivantes pour le problème d'ordonnancement des Job-Shops ont été proposées :

- Représentation basée sur les opérations.
- Représentation basée sur la tâche.
- Représentation basée sur la liste de préférences.
- Représentation basée sur les règles de priorité.
- Représentation basée sur le temps d'achèvement.
- Représentation basée sur les clés aléatoires.

Ces représentations peuvent être classées selon l'approche directe ou indirecte.

Dans l'approche directe, la solution de JSP est codée dans un chromosome et des algorithmes génétiques sont utilisés pour faire évoluer ces chromosomes afin de trouver un meilleur ordonnancement. Les représentations, telles que la représentation basée sur les opérations, la représentation basée sur les tâches, la représentation basée sur les relations de paires de tâches, la représentation basée sur le temps d'achèvement et la représentation basée sur les clés aléatoires appartiennent à cette classe.

Dans l'approche indirecte, telle que la représentation basée sur les règles de priorité, une séquence de règles de répartition pour l'affectation des tâches est codée dans un chromosome, mais pas un ordonnancement, et des algorithmes génétiques sont utilisés pour faire évoluer ces chromosomes afin de découvrir une meilleure séquence de règles de répartition. Un ordonnancement est ensuite construit avec la séquence de règles de répartition. La représentation basée sur la liste de préférences, la représentation basée sur les règles de priorité, la représentation basée sur le graphe disjonctif et la représentation basée sur la machine appartiennent à cette classe.

III.9.1 Représentation basée sur les opérations

Cette représentation code un ordonnancement comme une séquence d'opérations et chaque gène représente une opération. Il existe deux façons possibles de nommer chaque opération. Une façon naturelle consiste à utiliser un nombre naturel pour nommer chaque opération, comme la représentation de permutation pour TSP. Malheureusement, en raison de l'existence des contraintes de priorité, toutes les permutations des nombres naturels ne définissent pas des calendriers réalisables. Gen, Tsujimura et Kubota ont proposé une alternative: ils nomment toutes les opérations pour un travail avec le même symbole et les interprètent ensuite selon l'ordre d'apparition dans la séquence pour un chromosome donné (De Jong 1994 ; Kubota 1995). Pour un problème de n-machine m-job, un chromosome contient n × m gènes. Chaque travail apparaît dans le chromosome exactement m fois et chaque répétition (chaque gène) n'indique pas une opération concrète d'un travail mais se réfère à une opération

qui dépend du contexte. Il est facile de voir que toute permutation du chromosome donne toujours un calendrier réalisable.

Figure III.5 Un exemple de représentation basée sur les opérations.

Considérez le problème à trois tâches et trois machines indiqué dans le tableau III.2. Supposons qu'un chromosome soit donné comme le montre la figure III.5, où 1 représente Job1, 2 pour le Job2 et 3 pour le Job3. Parce que chaque tâche comporte trois opérations, il se produit exactement trois fois dans le chromosome. Par exemple, il y a trois 2 dans le chromosome qui représente les trois opérations de la deuxième tâche, le premier 2 correspond à la première opération qui sera traité sur la machine 3, le second 2 correspond à la deuxième opération qui sera traité sur la machine 2 et le troisième 2 correspondent à la troisième opération qui sera traitées sur la machine 1. On voit que toutes les opérations du job2 sont nommées avec le même symbole 2 puis interprétées selon leurs ordres d'occurrence dans la séquence de ce chromosome.

Les relations correspondantes aux opérations des Jobs et des machines sont données dans la figure III.6. La figure III.7 illustre un diagramme de Gantt montrant une représentation d'ordonnancement basée sur les opérations.

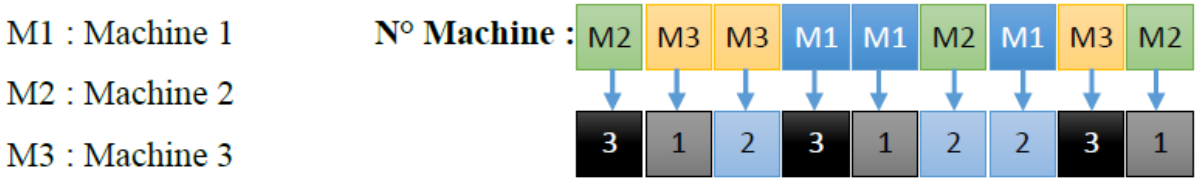


Figure III.6 Correspondance tâches / machines.

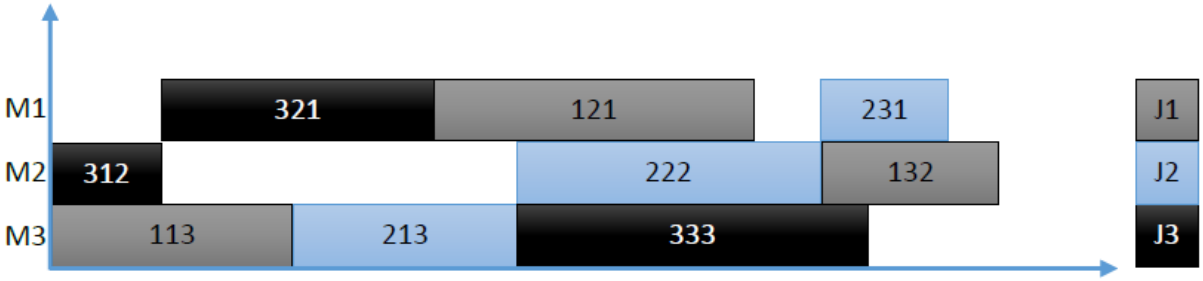


Figure III.7 Diagramme de Gantt pour une représentation basée sur les opérations.

III.9.2 Représentation basée sur la tâche (Job)

Cette représentation consiste en une liste de n Jobs et un ordonnancement est construit selon la séquence des tâches. Pour une séquence de tâche donnée, toutes les opérations du premier Job de la liste sont planifiées en premier, puis les opérations du deuxième Job de la liste sont prises en compte. La première opération de la tâche en cours de traitement est allouée dans le meilleur temps de traitement disponible pour la machine correspondante, puis la deuxième opération, et ainsi de suite jusqu'à ce que toutes les opérations de la tâche soient planifiées.

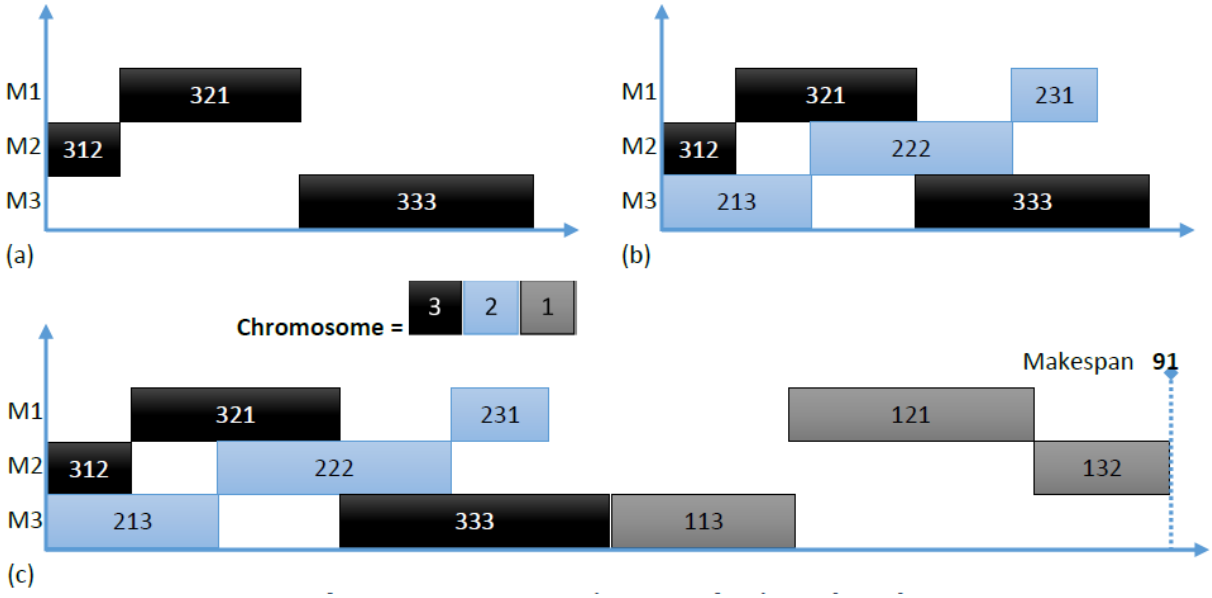


Figure III.8 Diagramme de Gantt pour une représentation basée sur la tâche.

Le processus est répété avec chacun des Jobs de la liste considérés dans l'ordre approprié. Toute permutation correspond à un ordonnancement est réalisable. Holsapple, Jacob, Pakath et Zaveri ont utilisé cette représentation pour traiter un problème d'ordonnancement statique dans un contexte de fabrication flexible (Baker & Scudder 1990).

Considérez le problème à trois tâches et trois machines indiqué dans le tableau III.2. Supposons qu'un chromosome soit donné comme le montre la figure III.5, où 1 représente le J1, 2 pour le J2 et 3 pour le J3. Le premier Job à traiter est le J3. La contrainte de priorité d'opération pour J3 est [M2, M1, M3] comme le montre la figure III.8(a). Le J2 est ensuite traité, sa priorité de fonctionnement sur les machines est [M3, M2, M1] comme indiqué sur la figure III.8(b). Chacune des opérations est planifiée dans le meilleur temps de traitement disponible pour la machine correspondante. Enfin, la solution finale ou le J1 est planifié comme le montre la figure III.8(c).

III.9.3 Représentation basée sur la liste de préférences

Cette représentation a été proposée par Davis pour une sorte de problème d'ordonnancement (Davis 1985). Falkenauer et Bouffouix (1991) l'ont utilisé pour traiter un problème Job-Shop avec des dates d'échéance. Croce et al (2015) l'on appliqué sur les problèmes classique de type Job-Shop.

Pour un problème d'ordonnancement Job-shop avec n Jobs et m machines, un chromosome est formé de m sous-chromosomes, chacun pour une machine. Chaque sous-chromosome est une chaîne de longueur n, et chaque symbole identifiant une opération qui doit être traitée sur la machine concernée. Les sous-chromosomes ne décrivent pas la séquence d'opération sur la machine, ce sont les listes de préférences ; chaque machine a sa propre liste de préférences. L'ordonnancement réel est déduit du chromosome grâce à une simulation, qui analyse l'état des files d'attente en face de la machine et utilise si nécessaire les listes de

préférences pour déterminer la solution ; c'est-à-dire que l'opération qui apparaît en premier dans la liste des préférences sera choisie.

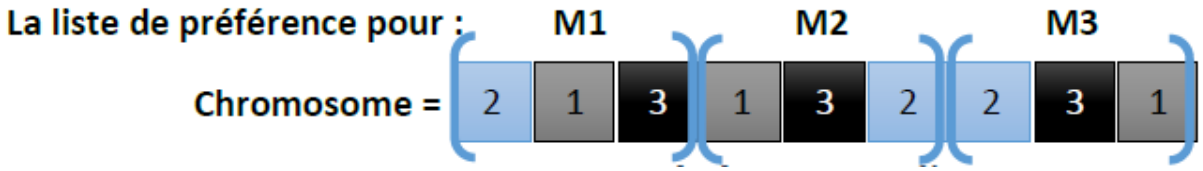


Figure III.9 Représentation basée sur la liste de préférences.

Considérez le problème à trois tâches et trois machines indiqué dans le tableau III.2. Supposons qu'un chromosome soit donné comme indiqué sur la figure III.9, le premier gène [2,1,3] est la liste de préférence pour la machine M1, [1,3,2] pour la machine M2 et [2,3,1] pour la machine M3. À partir de ces listes de préférences, nous pouvons savoir que les premières opérations sont le J2 sur la machine M1, J1 sur M2 et J2 sur M3. Selon les contraintes de priorité d'opération données, seul J2 sur M1 peut être planifié, il est donc planifié sur M1 en premier comme indiqué sur la figure III.10(a).

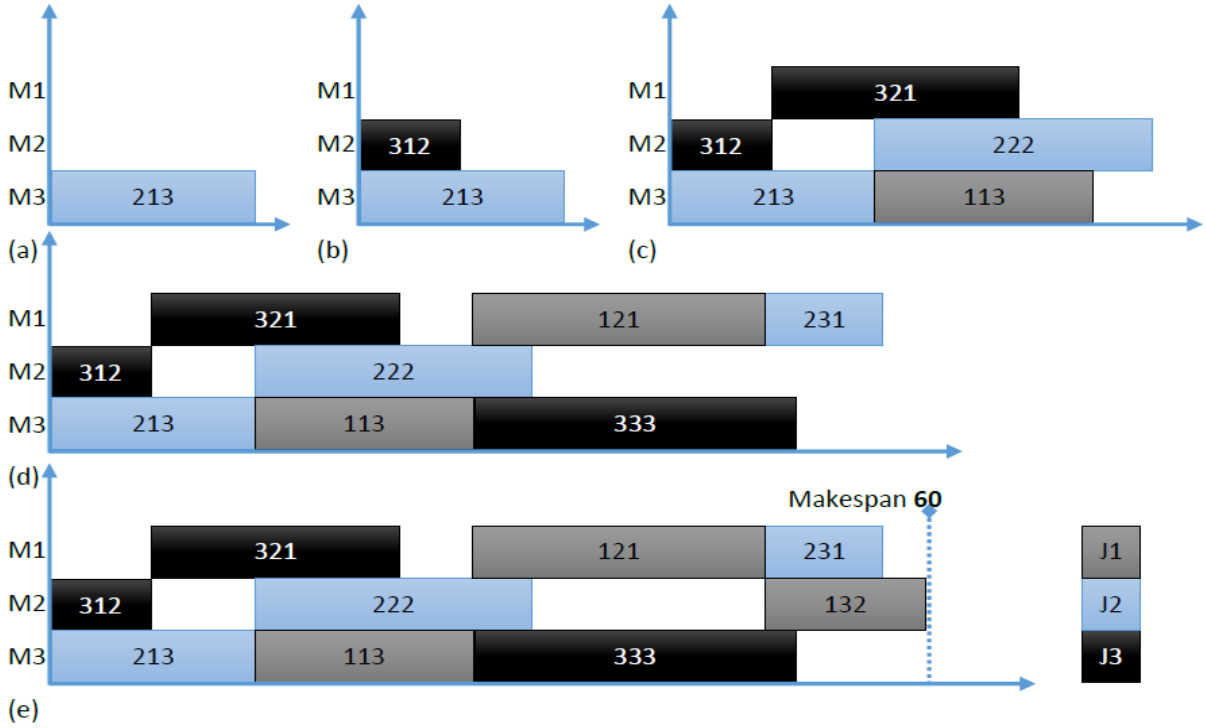


Figure III.10 Diagramme de Gantt pour une représentation basée sur la liste de préférences.

L'opération suivante est J3 sur M2 comme le montre la figure III.10 (b). Désormais, les opérations préférentielles actuelles sont J1 sur M2 et J2 sur M1. Parce qu'ils ne sont pas tous programmables à l'heure actuelle, nous recherchons la deuxième opération préférentielle dans chaque liste. Ils sont J3 sur M3, J3 sur M2, J1 sur M1. L'opération programmable est J3 sur M2. Maintenant, les opérations préférentielles actuelles sont J1 sur M2 et J2 sur M1, J3 sur M3, J1 sur M1, et les opérations préférentielles suivantes sont J1 sur M3 et J2 sur M2, J3

sur M1. J1 sur M3 et J2 sur M2, J3 sur M1 sont programmables comme le montre la figure III.10 (c). Après cela, les opérations préférentielles actuelles sont J1 sur M2, et J2 sur M1, J3 sur M3, J1 sur M1. J1 sur M1 et J2 sur M1, J3 sur M3 sont programmables comme indiqué sur la figure III.10(d). La dernière opération à programmer est J1 sur M2, comme illustré à la figure. III.10 (e).

III.9.4 Représentation basée sur les règles de priorité

Dorndorf et Pesch ont proposé une représentation basée sur des règles de priorité (Dorndorf & Pesch 1995), où un chromosome est codé comme une séquence de règles de répartition pour l'affectation des tâches et un ordonnancement est construit avec des heuristiques de répartition prioritaires basées sur la séquence de règles de répartition. Les AGs ici sont utilisés pour faire évoluer ces chromosomes afin de découvrir une meilleure séquence de règles de répartition.

Les règles de répartition sont probablement l'heuristique la plus fréquemment appliquée pour résoudre les problèmes d'ordonnancement dans la pratique en raison de leur facilité de mise en œuvre et de leur faible complexité. Les algorithmes de Giffler et Thompson peuvent être considérés comme la base commune de toutes les heuristiques basées sur des règles de priorité (Storer, Wu & Vaccari 1992). Le problème est d'identifier une règle de priorité efficace. Pour un résumé détaillé et une discussion sur les règles de répartition des priorités, voir Panwalkar et Iskander , Haupt et Blackstone et al. (1982).

Pour un problème m machine à n tâches, un chromosome est une chaîne de $n \times m$ variables

$[p_1, p_2, \ldots, p_i, \ldots, p_{nm}]$. Une entrée p_i représente une règle parmi l'ensemble des règles de répartition de priorité prédéfinies. L'entrée en $i^{ème}$ position indique qu'un conflit dans la $i^{ème}$ itération de l'algorithme de Giffler et Thompson doit être résolu en utilisant la règle de priorité p_i. Plus précisément, une opération de l'ensemble de conflits doit être sélectionnée par la règle pi. Les liens sont rompus par un choix aléatoire.

Variables :

PS_t : une solution partielle contenant t opérations.

S_t : l'ensemble des opérations à l'itération t, correspondant à PS_t.

σ_i : la première heure à laquelle l'opération $i \in S_t$ a pu être démarrée.

φ_i : la première heure à laquelle l'opération $i \in S_t$ a pu être achevée.

C_t : l'ensemble des opérations en conflit dans l'itération t.

La procédure pour déduire un ordonnancement d'un chromosome donné $[p_1, p_2, \ldots, p_{nm}]$ est la suivante :

Procédure

Entrée : chromosome

Sortie : un ordonnancement complet

1. Initialiser t = 1.

   Initialiser PS_t comme ordonnancement partiel nul.

   Initialiser S_t par toutes les opérations sans prédécesseurs.

   Tant que ! Ordonnancement complet soit généré

2. Déterminer $\varphi_t^* = \min_{i \in S_t}\{\varphi_i\}$ et la machine m^* sur laquelle φ_t^* pourrait être réalisée. S'il existe plusieurs machines de ce type, l'égalité est rompue par un choix aléatoire.
3. Formuler l'ensemble conflictuel C_t qui comprend toutes les opérations $i \in S_t$ avec $\sigma_i < \varphi^*_t$ qui nécessite la machine m^*. Sélectionnez une opération dans C_t par règle de priorité p_t et ajoutez cette opération à PS_t le plus tôt possible, créant ainsi une nouvelle solution partielle PS_{t+1}. S'il existe plus d'une opération selon la règle de priorité p_t, le lien est rompu par un choix aléatoire.

4. Pour PS_{t+1}, mettez à jour l'ensemble des données comme suit :

   d. Supprimer l'opération sélectionnée de S_t

   e. Ajouter le successeur direct de l'opération à S_t

   f. t++ ;

   Fin Tant que.

Nous utilisons cinq règles de priorité données dans le tableau III.1. Supposons qu'un chromosome soit donné comme le montre la figure III.11. À l'étape initiale, nous avons S1 = $\{o_{113}, o_{213}, o_{312}\}$, déterminez $\varphi^*_1 = min\ \{15, 14, 7\} = 7$ et la machine $m^* = 2$ sur laquelle φ^*_1 pourrait être réalisée. L'ensemble des opérations en conflit $C_1 = \{o_{312}\}$. L'opération o_{312} est prévue sur la machine M2. Ensuite, nous avons mis à jour S2 = $\{o_{113}, o_{213}, o_{321}\}$, déterminons $\varphi^*_2 = min\ \{15, 14, 24\} = 14$ et la machine $m^* = 3$ sur laquelle φ^*_2 pourrait être réalisée. L'ensemble des opérations conflictuelles $C_2 = \{o_{113}, o_{213}\}$. Désormais, les opérations o_{113} et o_{213} rivalisent pour la machine M3. Parce que le deuxième gène dans le chromosome donné est 2 (ce qui signifie la règle de priorité SPT), l'opération o_{213} est programmée sur la machine M3. Répétez ces étapes jusqu'à ce qu'un ordonnancement complet soit déduit du chromosome donné, $\{o_{312}, o_{213}, o_{321}, o_{113}, o_{222}, o_{231}, o_{333}, o_{121}, o_{132}\}$.

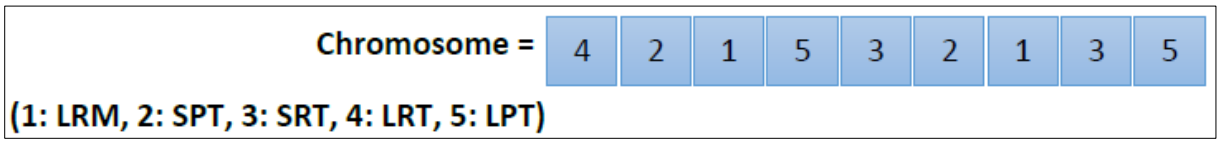


Figure III.11 Représentation basée sur les règles de priorité.

III.9.5 Représentation basée sur le temps d'achèvement

Yamada et Nakano ont proposé une représentation basée sur le temps d'achèvement (Gen, Tsujimura & Kubota 1994). Un chromosome est une liste ordonnée des temps d'achèvement des opérations. Pour le même exemple donné dans le tableau III.2, le chromosome peut être représenté comme suit : chromosome = [c_{113} c_{122} c_{131} c_{213} c_{221} c_{232} c_{312} c_{323} c_{331}] où c_{ikm} signifie le temps d'achèvement de l'opération *k* de la tâche *i* sur la machine *m*. Il est facile de savoir qu'une telle représentation ne convient pas à la plupart des opérateurs génétiques car elle entraînera une solution infaisable. Yamada et Nakano ont conçu un opérateur de croisement spécial pour cela.

III.9.6 Représentation basée sur les clés aléatoires

La représentation aléatoire des clés est d'abord donnée par Bean (1994). Avec cette technique, les opérations génétiques peuvent produire des nouveaux individus réalisables sans traitements supplémentaires pour une grande variété de problèmes d'ordonnancement et d'optimisation. Norman et Bean ont en outre généralisé avec succès l'approche du problème d'ordonnancement Job-Shop (Norman & Bean 1995). La représentation des clés aléatoires code une solution avec un nombre aléatoire. Ces valeurs sont utilisées comme clés de tri pour décoder la solution. Pour un problème de *n* Jobs et *m* machine, chaque gène (une clé aléatoire) se compose de deux parties : un entier dans l'ensemble {*1, 2, ... , m*} et une fraction générée aléatoirement à partir de (0,1). La partie entière de toute clé aléatoire est interprétée comme l'affectation de la machine pour une tâche. Le tri des pièces fractionnées fournit la séquence de tâches sur chaque machine. Prenons le même exemple donné dans le tableau III.2. Supposons qu'un chromosome soit : {1.02, 1.66, 1.84, 2.25, 2.35, 2.10, 3.55, 3.54, 3.28}

Triez dans l'ordre croissant pour obtenir la séquence de tâches 1 → 2 → 3 sur la machine M1, pour la machine M2 la séquence de tâches 3 → 1 → 2 et pour la machine M3 la séquence de tâches 3 → 2 → 1. Soit o_{ikj} une opération *k* du Job *i* sur la machine *j*. Le chromosome peut être traduit en une liste unique d'opérations comme suit :

[o_{111}, o_{211}, o_{331}, o_{312}, o_{132}, o_{222}, o_{323}, o_{233}, o_{123}]

On peut voir que les séquences de tâches indiquées ci-dessus peuvent violer les contraintes de priorité, une autre couche est ajoutée par Norman et Bean pour gérer les contraintes de précédence (Norman & Bean 1995).

III.10 Approches génétiques pour la résolution du problème job shop

III.10.1 Approche de Gen-Tsujimura-Kubota

Gen, Tsujimura et Kubota ont proposé une implémentation d'AG pour résoudre les problèmes d'ordonnancement Job-Shop (Okamoto, Gen & Sugawara 2005). Soit *P(t)* et *C(t)* respectivement parents et offspring de la génération actuelle *t*. La procédure globale de l'approche AG de Gen-Tsujimura-Kubota pour résoudre JSP est décrite comme suit :

Paramètres Algorithme Génétique : dimension de la population, maximum génération, probabilité mutation & croisement.

Procédure :
Entrée : données JSP, paramètres AG (popSize, maxGen, pM, pC)
Sortie : le meilleur ordonnancement
Debut
t := 0;
Initialiser *P(t)* par un encodage basé sur les opérations ;
Evaluer *P(t)* par un décodage basé sur les opérations ;
Tant que ! Fin Condition
Créer *C(t)* à partir de *P(t)* par un croisement partiel ;
Créer *C(t)* à partir de *P(t)* par une mutation (Echange de paires) ;
Evaluer *P'(t)* et *C'(t)* par un décodage basé sur les opérations ;
Sélectionner *P(t +1)* parmi *P(t)* et *C(t)* ;
t++ ;
Fin Tant que
Fin

Croisement d'échange partiel

Gen, Tsujimura et Kubota ont conçu un opérateur de croisement d'échange partiels pour une représentation basée sur les opérations, qui considère les ordonnancements partiels comme les blocs de construction naturels et a l'intention d'utiliser ce croisement pour maintenir les blocs de construction dans l'offspring de la même manière que Holland l'a décrit (1995). L'ordonnancement partiel est identifié avec le même Job dans la première et dernière positions. La procédure de croisement d'échange partiel est définie comme suit :

Procédure
Entrée : parents
Sortie : offspring
Début

1. Sélectionnez des ordonnancements partiels
   a. Choisissez au hasard une position de Job i dans le parent1 ;
   b. Trouvez le voisin le plus proche de Job i dans le même parent1, et obtenez le partiel1 comme [i xxx i] ;
   c. L'ordonnancement partiel suivant n'est pas généré de manière aléatoire à partir du parent2. Il doit commencer et se terminer avec le Job i comme le premier partiel et obtenir le partiel2 ;
2. Échangez des ordonnancements partiels entre les parents 1 et 2 ;
3. Vérifiez les gènes manqués et les gènes dépassés dans chaque offspring ;
4. Normalisez l'offspring 1 et 2 en supprimant les gènes dépassés et en ajoutant les gènes manqués.

Fin

La figure III.12 illustre un exemple de croisement d'échange partiel. Tout d'abord, nous choisissons une position dans le parent1 au hasard. Supposons que c'est la troisième position à laquelle se trouve le J2, et découvrez le prochain J2 le plus proche dans le même parent, qui est en position 7. Maintenant, nous obtenons le partiel 1 comme [2 3 3 1 2]. Trouvez ensuite le partiel 2 du parent2, que les deux J2 sont le troisième et le cinquième parent2 : [2 1 2]. Après avoir échangé les partiels, nous pouvons trouver que les gènes dépassés d'offspring 1 sont {3,3}, et les gènes manqués d'offspring 2 sont {3,3}. Enfin, légalisez l'offspring 1 et 2 en supprimant les gènes dépassés {3,3} et en ajoutant les gènes manqués {3,3}, respectivement.

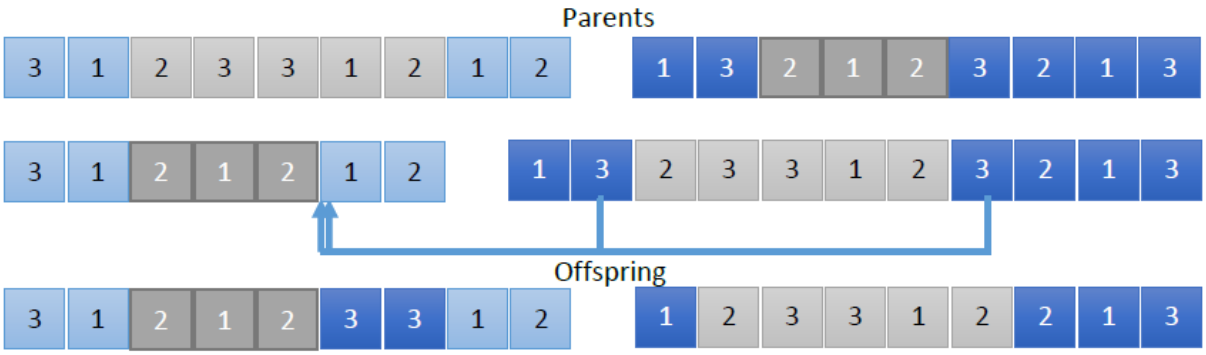


Figure III.12 Croisement d'échange partiel.

Mutation (échange de paires)

Gen, Tsujimura et Kubota utilisent un échange de paires aléatoire, c'est-à-dire ramasser au hasard deux Jobs non identiques, puis échanger leurs positions comme le montre la figure III.13 Pour la représentation basée sur les opérations, la relation entre le chromosome et l'ordonnancement est une relation n→1 et l'offspring obtenue en échangeant les deux positions les plus proches peut produire la même solution que le parent. Le mieux est une séparation plus distante entre deux Jobs non identiques.

Figure III.13 Mutation

III.10.2 Approche de Cheng-Gen-Tsujimura

Cheng, Gen et Tsujimura ont encore modifié la méthode de Gen, Tsujimura et Kubota afin d'améliorer son efficacité. Soit *P(t)* et *C(t)* respectivement parents et offspring de la génération actuelle *t*. La procédure globale de l'approche AG de Cheng-Gen-Tsujimura pour résoudre JSP est décrite comme suit:

Procédure :
Entrée : données JSP, paramètres AG (popSize, maxGen, pM, pC)
Sortie : le meilleur ordonnancement
Début
t := 0 ;
Initialiser *P(t)* par un encodage basé sur les opérations ;
Evaluer *P(t)* par un décodage basé sur les opérations ;
Monter *C(t)* par un décalage à gauche ;
Tant que ! Fin Condition
Créer *C(t)* à partir de *P(t)* par un croisement partiel ;
Créer *C(t)* à partir de *P(t)* par une mutation (Echange de paires) ;
Evaluer *P'(t)* et *C'(t)* par un décodage basé sur les opérations ;
Sélectionner *P(t +1)* parmi *P(t)* et *C(t)* ;
t++ ;
Fin Tant que
Fin

Encodage et décodage basés sur les opérations modifiés

Dans la méthode de Gen, Tsujimura et Kubota, lors du décodage d'un chromosome en une solution, ils ne prennent en compte que deux paramètres : l'ordre des opérations dans le chromosome donné et les contraintes de priorité des opérations des tâches, et spécifient l'ordre des opérations sur chaque machine dans l'ordonnancement. La procédure de décodage peut simplement garantir de générer une solution semi-active mais pas une solution active ; c'est-à-dire qu'il peut exister un décalage à gauche dans l'ordonnancement. Étant donné que la solution optimale est un ordonnancement actif, la procédure de décodage inhibera l'efficacité de l'algorithme proposé pour les problèmes Job-Shop à grande échelle.

Nous utilisons toujours le même problème à trois Jobs et trois machines indiqué dans le tableau III.2. Supposons que nous ayons un chromosome = [211123233] ; par la procédure de décodage, nous pouvons obtenir une liste de machines correspondante comme [331222113].

Selon la liste des machines, nous pouvons faire un ordonnancement comme indiqué dans le tableau III.5, qui spécifie l'ordre des opérations sur chaque machine.

Machines	Jobs		
M1	1	2	3
M2	1	2	3
M3	2	1	3

Tableau III.6 Proposition d'ordonnancement

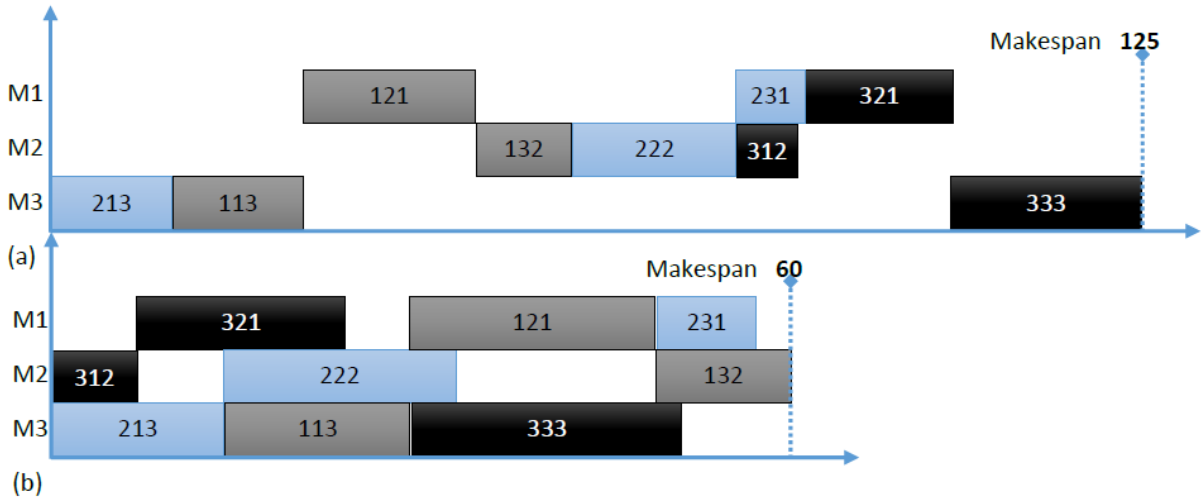


Figure III.14 Effet des décalages à gauche sur un ordonnancement semi-actif.

La figure III.14(a) montre le diagramme de Gantt de l'ordonnancement semi-actif obtenu par la procédure de décodage pour le chromosome donné avec un Makespan de 125. Il y a deux décalages à gauche autorisés dans la solution semi-active :

1. Le Job3 peut démarrer au temps 0 sur la machine M2.
2. L'opération o_{222} peut démarrer au temps 14 sur la machine M2.

En effectuant ces décalages vers la gauche, nous obtenons un ordonnancement actif illustré sur la figure III.14(b), qui a un Makespan de 60. Cheng, Gen et Tsujimura ont modifié la procédure de décodage pour garantir de générer une solution active à partir d'un chromosome donné. Pour la représentation basée sur les opérations, chaque gène indique de manière unique une opération. La procédure de décodage modifiée traduit d'abord le chromosome en une liste d'opérations ordonnées. Une planification est générée par une heuristique à un seul passage basée sur la liste. La première opération de la liste est planifiée en premier, puis la deuxième opération de la liste est prise en compte, etc. Chaque opération en cours de traitement est allouée dans le meilleur temps de traitement disponible pour la machine correspondante à l'opération requise. Le processus est répété jusqu'à ce que toutes les opérations de la liste soient planifiées aux endroits appropriés. Un ordonnancement généré par cette procédure modifiée peut être garanti comme une solution active. Un chromosome ici peut être envisagé pour attribuer une priorité à chaque opération. Étiquetons la position de gauche à droite de la liste du plus bas au plus haut. Une opération avec une position plus basse dans la liste a une priorité plus élevée et sera planifiée en premier avant celles avec une position plus élevée.

Croisement étendu

L'opérateur de croisement donné dans la dernière section est plutôt complexe car il essaie de garder les ordres d'opérations d'un ordonnancement partiel dans l'offspring les mêmes que dans le parent. L'approche modifiée utilise un chromosome pour déterminer la priorité de chaque opération et un ordonnancement est généré par une heuristique basée sur les règles de priorité en un seul passage. Par conséquent, il n'est pas nécessaire de conserver les mêmes ordres pour les opérations d'ordonnancement partiel chez les deux parents, et l'offspring et le croisement d'échange de la solution partielle peuvent être ensuite simplifiés comme dans la figure III.15 qui illustre un exemple de croisement étendu.

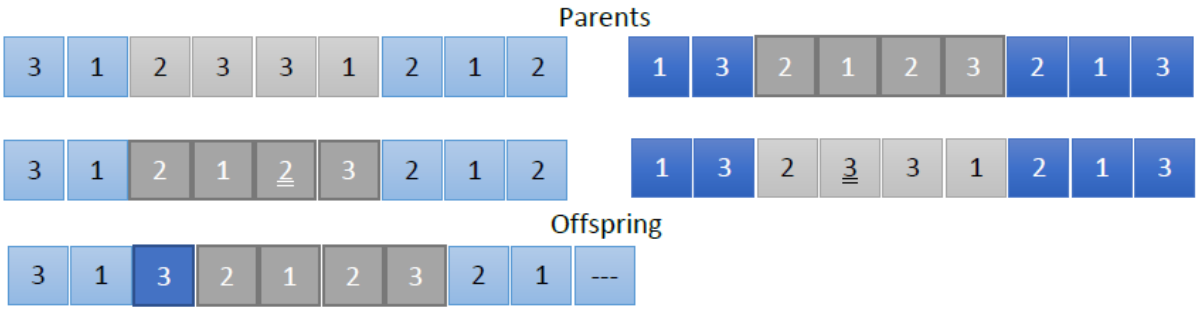


Figure III.15 Effet de croisement étendu.

III.10.3 Approche de Gonc¸alves-Magalhacs-Resende

Gonc¸alves et al. (2005) ont proposé une autre technique basée sur le chemin critique pour améliorer l'efficacité de l'algorithme de recherche d'un ordonnancement actif (Gonc¸alves et al. 2005). Soit *P(t)* et *C(t)* respectivement parents et offspring de la génération actuelle *t*. La procédure globale de l'approche AG de Goncalves-Magalhacs-Resende pour résoudre JSP est décrite comme suit :

Procédure
Entrée : parents
Sortie : offspring
Début
1. Choisissez deux partiels des parents 1&2 (Les deux partiels contiennent le même nombre de gènes) ;
2. Échangez des ordonnancements partiels entre les parents 1 et 2 ;
3. Vérifiez les gènes manqués et les gènes dépassés dans l'offspring ;
4. Normalisez l'offspring en supprimant les gènes dépassés et en ajoutant les gènes manqués de manière aléatoire ;

Fin

Recherche locale basée sur le chemin critique

La relation entre l'ordonnancement actif, semi-actif et sans-délai est déjà vue dans la section des classes d'ordonnancements illustrée par le diagramme de Venn de la figure III.4. Ou la solution optimale fait partie de l'ensemble des ordonnancements actifs (Gen & Cheng 2002).

La durée d'un ordonnancement est égale à la longueur du chemin critique. Une opération critique ne peut être retardée sans augmenter le Makespan. Pour le voisinage, seulement lorsque ces deux opérations adjacentes sont sur le chemin critique, la nouvelle solution est peut-être supérieure à la précédente.

Procédure :
Entrée : données JSP, paramètres AG (popSize, maxGen, pM, pC)
Sortie : le meilleur ordonnancement
Début
t := 0 ;
Initialiser *P(t)* par un encodage basé sur les clés aléatoire ;
Evaluer *P(t)* par un décodage basé sur les clés aléatoire ;
Monter *C(t)* par un recherche locale basé sur le chemin critique ;
Tant que ! Fin Condition
Créer *C(t)* à partir de *P(t)* par un croisement uniforme ;
Créer *C(t)* à partir de *P(t)* par une génération aléatoire ;
Monter *C(t)* par un recherche locale basé sur le chemin critique ;
Evaluer *P'(t)* et *C'(t)* par un décodage basé sur les clés aléatoire ;
Sélectionner *P(t +1)* parmi *P(t)* et *C(t)* ;
t++ ;
Fin Tant que
Fin

Gonc¸alves, Magalhacs et Resende utilisent une recherche locale basée sur le modèle de graphe disjonctif de Roy et Sussmann et le voisinage de Nowicki Smutnicki (1996) pour améliorer les solutions du problème. La procédure de recherche locale commence par identifier le chemin critique dans la solution obtenue par la procédure de décodage. Toute opération sur le chemin critique est appelée opération critique. Il est possible de décomposer le chemin critique en un certain nombre de blocs où un bloc est une séquence maximale d'opérations critiques adjacentes qui nécessitent la même machine. Si un prédécesseur de tâche et un prédécesseur de machine d'une opération critique sont tous les deux critiques, choisissez le prédécesseur (parmi ces deux alternatives) qui a une priorité plus élevée dans la solution actuelle. Pour JSP, une solution de voisinage peut être générée en échangeant la séquence de deux opérations critiques adjacentes qui appartiennent au même bloc. Dans chaque bloc, nous n'échangeons que les deux dernières et les deux premières opérations. Pour le premier (dernier) bloc, nous n'échangeons que les deux dernières (premières) opérations du bloc. Dans les cas où le premier et / ou le dernier bloc ne contient que deux opérations, ces opérations sont permutées. Si un bloc ne contient qu'une seule opération, aucun échange n'est effectué. Si aucun échange de première ou dernière opération dans un bloc de chemin critique n'améliore le Makespan, la recherche locale se termine.

Définitions :

M : Makespan de l'ordonnancement réalisable Initial ;
Opr : opérations critiques ;
Nbr-Opr : nombre total des opérations critiques ;
TMp : Makespan ;

Procédure
Entrée : données JSP, ordonnancement réalisable et Makspan M.
Sortie : un ensemble d'opérations critiques et son nombre total.
Début
Nbr-Opr=0 ;TMp=M ;
Pour i :=0 ; i=n ; i++ Pour k :=0 ; k=m ; k++
Si (TMp==t_{ik}^{F}) Début
Insérer o_{ik} dans Opr ;
Nbr-Opr ++ ;
TMp := TMp -p_{ikj} ; Fin
Fin

De plus, Gonc¸alves et al. a proposé une procédure de recherche locale efficace pour réduire le temps d'inactivité. Dans cette procédure, nous définissons d'abord les blocs non traités, le premier bloc critique du côté gauche et le dernier bloc critique à droite.

À titre d'exemple sur la figure III.16, nous pouvons obtenir un chemin critique et découvrir les échanges d'opérations possibles à partir de la solution initiale (Makespan est 72) en suivant la table de trace du tableau III.6. Il n'a que le bloc critique [231 −121] dans M1 pour l'échange. Si nous échangeons les deux opérations, nous obtenons un meilleur ordonnancement avec un Makespan 60, puis générons le nouveau chemin critique.

Nbr-Opr	**TMp**	**O_{ikj}**	**P_{ikj}**	**M_j**	**Opr**
1	72	132	11	2	132
2	61	121	20	1	132, 121
3	41	231	8	1	132, 121, 231
4	33	222	19	2	132, 121, 231, 222
5	14	213	15	3	132, 121, 231, 222, 213

Tableau III.7 Tableau de traçabilité pour le chemin critique de la solution initiale.

Croisement uniforme

Des croisements uniformes paramétrés sont utilisés à la place du croisement traditionnel à un ou deux points. Après que deux parents ont été choisis au hasard, nous lançons une monnaie équitable pour sélectionner le parent qui contribuera à l'allèle. Il suppose qu'un tirage au sort d'une tête sélectionne le gène du premier parent, une queue choisit le gène du deuxième parent, et que la probabilité de lancer une tête est par exemple de 0.6, (cette valeur est déterminée empiriquement)

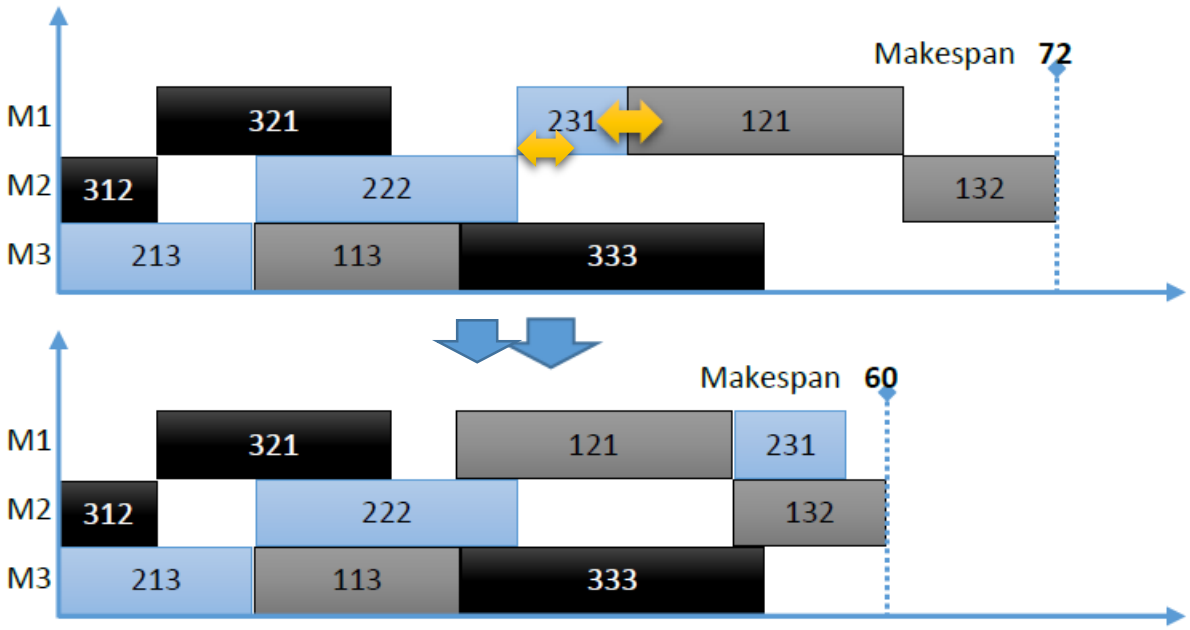


Figure III.16 Amélioration après l'échange d'un bloc critique.

III.11 Du séquentiel au parallèle

La plupart des méta-heuristiques de base sont séquentielles. Bien que leur utilisation permette une réduction significative de la complexité temporelle du processus de recherche, cette dernière est couramment utilisée pour les problèmes du monde réel qui se posent dans les domaines académiques et industriels. Celle-ci reste chronophage pour les problèmes industriels, par conséquent, le parallélisme est le meilleur moyen naturel non seulement pour réduire le temps de traitement, mais également pour l'amélioration de la qualité des solutions fournies. Dans le chapitre3, nous incluons une étude approfondie et organisée dans le domaine des Algorithmes génétiques parallèles (PGA). Nous discutons de nouveaux modèles d'algorithmes et de la façon dont ces techniques peuvent tirer parti des fonctionnalités des nouvelles plates-formes parallèles. Plus tard, nous décrivons les avancées par rapport aux nouvelles plates-formes informatiques parallèles, telles que les grilles de calculs, le GPU (unité de traitement graphique) et les FPGA (réseau de portes programmables), indiquant comment les algorithmes génétiques parallèles peuvent bénéficier de leurs nouvelles fonctionnalités.

III.12 Conclusion

Dans ce chapitre, la plupart des axes de problèmes d'ordonnancement Job-Shop sont traités, nous avons d'abord discuté de la complexité des problèmes d'ordonnancement à partir d'une approche simple en évitant autant que possible les aspects techniques de la complexité de calcul. La conclusion de ceci est que la plupart des problèmes d'ordonnancement sont complexes sur le plan informatique et ont un nombre astronomique de solutions possibles. Nous avons également souligné l'importance relative d'obtenir la solution optimale exacte pour un problème d'ordonnancement, étant donné que ce que nous résolvons réellement est un modèle à partir de la réalité étudiée et les approximations et simplifications apportées à ce modèle pourraient parfaitement compenser les gains d'une solution optimale obtenue. Les principales

méthodes d'ordonnancement ont également été présentées, chaque méthode à ses avantages et ses inconvénients. En résumé, les approches exactes fournissent la solution optimale mais ne sont viables que pour des problèmes simples et pour de petites instances également. À moins qu'un problème très spécifique ne soit étudié pour lequel une approche d'optimisation exacte très efficace existe, il n'est pas recommandé de les utiliser dans des problèmes de production complexes. Les méta-heuristiques sont souvent adaptées à certains problèmes et objectifs spécifiques, et elle fournit généralement de bien meilleures solutions par rapport aux autres approches. La dernière partie du chapitre est consacrée à la partie représentation et operateurs génétiques proposer et adapté pour la résolution du problème. Contrairement aux algorithmes exacts dont la complexité temporelle dans le pire des cas est connue, les méta-heuristiques ne fournissent pas ce type de borne. Ils peuvent être très efficaces sur une instance d'un problème et, en même temps, montrer une longue disponibilité sur une autre sans trouver de solution satisfaisante. Nous verrons dans la suite comment pallier ces insuffisances, notamment en s'appuyant sur des techniques parallèles.

IV. Chapitre IV Algorithmes génétiques parallèles (AGP) : concepts et application au problème JSP

IV.1 Introduction

Dans la pratique, les problèmes d'optimisation sont souvent difficiles à gérer, complexes et chronophages sur le processeur. Deux approches principales sont traditionnellement utilisées pour résoudre ces problèmes : les méthodes exactes et la méta-heuristique. Les méthodes exactes permettent de trouver des solutions exactes mais elles ne sont pas pratiques car elles prennent beaucoup de temps. Inversement, les méta-heuristiques fournissent des solutions sous-optimales dans un délai raisonnable. Ils permettent de répondre aux délais de résolution souvent imposés dans le domaine industriel. Bien que l'utilisation de la méta-heuristique permette de réduire de manière significative la complexité temporelle du processus de recherche, cette dernière reste chronophage pour les problèmes industriels. Par conséquent, le parallélisme est nécessaire non seulement pour réduire le temps de résolution mais également pour améliorer la qualité des solutions fournies. Pour chacune des deux familles de méta-heuristiques, différents modèles parallèles ont été proposés dans la littérature. Chacun d'eux illustre une approche alternative pour gérer et déployer la parallélisation.

Dans ce chapitre, nous allons aborder les concepts de base des métaheuristiques parallèles. Ensuite nous décrivons les algorithmes génétiques parallèles et leur application pour la résolution du Job-Shop.

IV.2 Le parallélisme et les méta-heuristiques

Pour résoudre les problèmes d'optimisation, les méta-heuristiques pourraient être considérées comme des recherches de trajectoire à travers les différents domaines de solutions des problèmes (Menouer 2018). Les méta-heuristiques parallèles utilisent souvent des techniques classiques comme base de leurs travaux. Dans la littérature deux modèles parallèles classiques ont été distingués, des méta-heuristiques basées sur la trajectoire et des méta-heuristiques basées sur la population, malgré le fait que les modèles parallèles appliqués à chacun soient légèrement différents.

IV.2.1 Méta-heuristique basée sur la trajectoire

Les familles méta-heuristiques bien connues basées sur la manipulation d'une solution unique incluent : le recuit simulé (SA), la recherche tabou (TS), la recherche locale itérée (ILS), la recherche locale variable (VNS) et les procédures de recherche adaptative randomisées (GRASP).

Les déroulements sont effectués par des procédures itératives qui permettent de passer d'une solution à une autre dans l'espace de la solution (voir Figure IV.1). Ce type de méta-heuristique effectue les mouvements au voisinage de la solution actuelle. Les parcours partent d'une solution générée aléatoirement ou obtenue à partir d'un autre algorithme d'optimisation (en cas d'hybridation). A chaque itération, la solution actuelle est remplacée par une autre sélectionnée dans l'ensemble de ses candidats voisins. Le processus de recherche est arrêté lorsqu'une condition donnée est satisfaite (par exemple, atteint un nombre maximum de mouvements, trouvé une solution avec une qualité cible ou bloqué pendant un temps donné).

Un moyen puissant pour atteindre une efficacité de calcul élevée avec des méthodes basées sur la trajectoire est l'utilisation du parallélisme. Différents modèles parallèles ont été proposés pour les méta-heuristiques basées sur la trajectoire, et trois d'entre eux sont couramment utilisés dans la littérature (voir Figure. IV.1) : (a) l'exploration parallèle et l'évaluation du voisinage (ou modèle de mouvements parallèles), (b) l'évaluation parallèle d'une solution unique (ou modèle d'accélération de mouvements), et (c) le modèle parallèle à plusieurs départs.

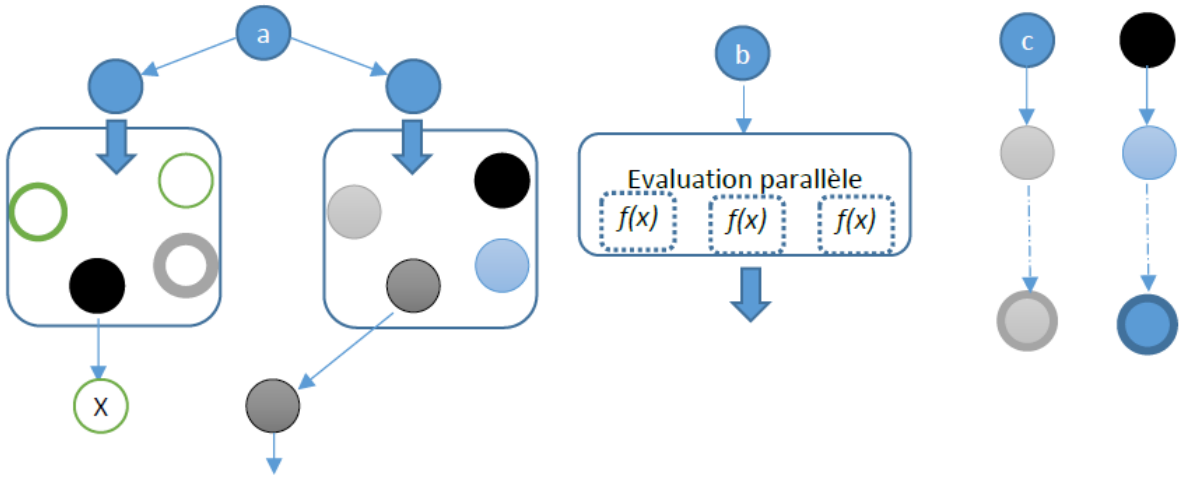


Figure IV.1 Modèles parallèles pour la méta-heuristique basée sur la trajectoire.

1. **Modèle de mouvements parallèles :** Il s'agit d'un modèle maître-esclave de bas niveau qui ne modifie pas le comportement de l'algorithme. Une recherche séquentielle calculerait le même résultat mais lentement. Au début de chaque itération, le maître duplique la solution actuelle entre les nœuds distribués. Chacun gère séparément son candidat (la solution et les résultats sont retournés au maître).
2. **Modèle parallèle à plusieurs départs :** il consiste à lancer simultanément plusieurs méthodes basées sur la trajectoire pour calculer des solutions meilleures et robustes. Ils peuvent être hétérogènes ou homogènes, indépendants ou coopératifs, à partir des solutions identiques ou différentes, et configurés avec des paramètres identiques ou différents.
3. **Modèle d'accélération des mouvements :** la qualité de chaque mouvement est évaluée de manière centralisée parallèle. Ce modèle est particulièrement intéressant lorsque la fonction d'évaluation peut être mise en parallèle car son processeur prend du temps et/ou des Entré/Sortie. Dans ce cas, la fonction peut être considérée comme une agrégation d'un certain nombre de fonctions partielles pouvant être exécutées en parallèle.

IV.2.2 Méta-heuristiques basées sur la population

Les méta-heuristiques basées sur la population sont des techniques de recherche stochastiques qui ont été appliquées avec succès dans de nombreuses applications réelles et complexes. Un algorithme basé sur la population est une technique itérative qui applique des opérateurs stochastiques sur un pool d'individus (population) (voir Figure IV.3). Chaque individu de la population est la version codée d'une solution provisoire. Une fonction d'évaluation associe une valeur de fitness à chaque individu indiquant son adéquation au

problème. De façon itérative, l'application probabiliste sur des individus sélectionnés guide la population vers des solutions provisoires de meilleure qualité. Les familles méta-heuristiques bien connues basées sur la manipulation d'une population comprennent : les algorithmes génétiques (AG), l'optimisation des colonies de fourmis (ACO), l'optimisation des essaims de particules (PSO) et l'évolution différentielle (DE).

Au début de l'histoire de la parallélisation de ces algorithmes, la méthode bien connue «maître-esclave» (également appelée «parallélisation globale») a été utilisée. Dans cette approche, un processeur central effectue les opérations de sélection tandis que les processeurs esclaves associés exécutent l'opérateur de variation et l'évaluation de la fonction de fitness. Cet algorithme a le même comportement que celui séquentiel, bien que son efficacité de calcul soit améliorée, en particulier pour les fonctions objectifs chronophages. D'un autre côté, de nombreux chercheurs utilisent un pool de processeurs pour accélérer l'exécution d'un algorithme séquentiel, uniquement parce que des «exécutions indépendantes» peuvent être effectuées plus rapidement en utilisant plusieurs processeurs qu'en utilisant un seul. Dans ce cas, aucune interaction n'existe entre les différents calculateurs.

Cependant, la plupart des techniques basées sur la population parallèles trouvées dans la littérature utilisent une sorte de disposition spatiale pour les individus, qui sont déployés sur un pool de processeurs. Parmi les types de méta-heuristiques structurées les plus connues, les algorithmes «distribués» (coarse-grain) et «cellulaires» (fine-grain) sont des procédures d'optimisation très populaires (Alba & Tomassini, 2002). Dans le cas de celles distribuées, la population est partitionnée en un petit ensemble de sous-populations dans lesquelles des algorithmes série isolés sont exécutés. De rares échanges individuels sont effectués entre ces îles dans le but d'introduire une certaine diversité dans les sous-populations, empêchant ainsi la recherche de rester coincé dans les optima locaux. Pour concevoir une méta-heuristique distribuée, plusieurs paramètres sont à prendre en compte. Parmi eux, la décision principale est de déterminer les paramètres liés à la politique de migration : liens entre les îles, taux de migration, période de migration (nombre d'étapes effectuées dans chaque sous-population entre deux échanges successifs), et la sélection/remplacement des individus. Dans le cas d'une méthode cellulaire, le concept de «voisinage» est introduit, de sorte qu'un individu ne peut interagir qu'avec ses voisins proches. Le pool qui se chevauche dans l'algorithme aide à explorer l'espace de recherche car une diffusion lente des solutions à travers la population fournit une sorte d'exploration, tandis que l'exploitation a lieu à l'intérieur de chaque pool (Alba & Dorronsoro, 2008).

IV.3 Les métriques de performance des méta-heuristiques parallèles

Il existe différents facteurs pour mesurer les performances des algorithmes parallèles. Nous résumons quelques métriques trouvées dans la littérature, essayons de nous concentrer sur la métrique la plus courante, à savoir l'accélération, et discutons son utilisation significative dans les méta-heuristiques parallèles.

IV.3.1 Accélération (Speedup)

La métrique la plus importante d'un algorithme parallèle est l'accélération. Cette métrique compare deux temps : rapport entre les temps séquentiels et parallèles (Dorronsoro & al. 2013 ; Alba 2002). Par conséquent, la définition du temps est le premier aspect auquel nous devons faire face. Dans un système monoprocesseur, une performance courante est le temps CPU pour résoudre un problème ; il s'agit du temps que le processeur passe à exécuter les instructions de l'algorithme, en excluant généralement le temps d'entrée des données de problème, de sortie des résultats et des activités de surcharge du système. Dans le cas parallèle, le temps n'est ni une somme des temps CPU sur chaque processeur ni le plus grand d'entre eux. Étant donné que l'objectif du parallélisme est la réduction du temps réel, le temps devrait certainement inclure tout temps d'activité supplémentaire, car c'est le prix de l'utilisation d'un algorithme parallèle. Par conséquent, le choix le plus prudent pour mesurer les performances d'un code parallèle est le « wall-clock time » le temps pour résoudre un problème. Cela signifie utiliser le temps entre le début et la fin de l'algorithme entier.

L'accélération compare le temps série au temps parallèle pour résoudre un problème particulier. Si l'on note T_m le temps d'exécution d'un algorithme utilisant m processeurs, l'accélération est le rapport entre le temps d'exécution plus rapide sur un mono-processeur T_1 et le temps d'exécution sur m processeurs T_m : $s_m = T_1 / T_m$

Pour les algorithmes non déterministes, nous ne pouvons pas utiliser cette métrique directement. Pour ce type de méthodes, l'accélération doit plutôt comparer le temps d'exécution série moyen au temps d'exécution parallèle moyen : $s_m = \frac{M[T_1]}{M[T_m]}$

Ici, nous supposons une distribution normale du temps ou du moins que nous avons de nombreuses exécutions indépendantes pour l'approcher (théorème central limite). Sinon, la médiane ou un autre paramètre pourrait également être utilisé.

Avec cette définition, nous pouvons distinguer : accélération sub-linéaire ($s_m < m$), accélération linéaire ($s_m = m$) et accélération super-linéaire ($s_m > m$). La principale difficulté de cette mesure est que les chercheurs ne s'entendent pas sur la signification de T_1 et T_m.

Dans son étude, Alba (2002) distingue plusieurs définitions de l'accélération en fonction de la signification de ces valeurs.

Type1 : Une forte accélération compare le temps d'exécution parallèle au meilleur algorithme séquentiel jusqu'à présent. Il s'agit de la définition la plus exacte et la plus standard de l'accélération, mais en raison de la difficulté de trouver l'algorithme actuel le plus efficace, la plupart des concepteurs d'algorithmes parallèles ne l'utilisent pas et se retrouvent dans type2.

Type2 : Une accélération faible compare l'algorithme parallèle développé par un chercheur à sa propre version en série. Dans ce cas, il existe deux critères d'arrêt pour les algorithmes : la qualité de la solution ou l'effort maximum. L'auteur rejette ce dernier car il est contraire à l'objectif d'accélération de comparer des algorithmes ne donnant pas des résultats d'égale précision. En fait, après l'utilisation standard et historique de l'accélération, les deux algorithmes comparés devraient faire exactement le même travail de recherche, chose qui n'est pas clairement intéressante dans la plupart des méta-heuristiques. Il propose deux variantes de faible vitesse avec arrêt de solution : comparer l'algorithme parallèle à la version séquentielle ou comparer le temps d'exécution de l'algorithme parallèle sur un processeur avec le temps

d'exécution du même algorithme sur m processeurs. Dans le premier cas, nous comparons deux algorithmes clairement différents, ce qui pourrait encore soulever de sérieuses inquiétudes sur la signification de cette métrique. Ainsi, l'approche orthodoxe semble le moyen le plus juste d'évaluer les méta-heuristiques parallèles et en même temps d'utiliser une métrique standard trouvée dans d'autres applications parallèles.

Barr et Hickman (1993) ont montré une taxonomie différente : accélération, accélération relative et accélération absolue. Le Speedup mesure le rapport entre le temps du code série le plus rapide sur une machine parallèle et le temps du code parallèle en utilisant m processeurs sur une machine avec des caractéristiques similaires à celle utilisée dans celui série. L'accélération relative est le rapport entre le temps d'exécution en série avec du code parallèle sur un processeur et le temps d'exécution de ce code sur m processeurs. Cette définition est similaire au type2 illustré ci-dessus. L'accélération absolue compare le temps série le plus rapide sur n'importe quel ordinateur avec le temps parallèle sur les processeurs m. Cette métrique est la même que la forte accélération définie par Alba (2002).

Il est clair que les méta-heuristiques parallèles évaluées devraient calculer des solutions ayant une précision similaire à celles séquentielles. Cette précision pourrait être la valeur de fitness optimale (si elle est connue) ou un relâchement de celle-ci (par exemple 90 ~ 95 %), mais en tout cas la même valeur. Dans ce cas seulement, nous sommes autorisés à comparer les temps. Les temps utilisés sont des temps moyens : le code parallèle sur une machine versus le code parallèle sur m machines. Tout cela définit une bonne comparaison, pratique (aucun meilleur algorithme nécessaire) et orthodoxe (mêmes codes, même précision).

Nous pouvons nous attendre à obtenir parfois une accélération super-linéaire quel que soit le programme métrique parallèle. En fait, nous pouvons souligner plusieurs sources derrière l'accélération super-linéaire :

- Source d'implémentation : l'algorithme exécuté sur un processeur est «inefficace» d'une certaine manière. Par exemple, si l'algorithme utilise des données complexes, la solution parallèle peut être plus rapide car elle gère des listes plus courtes (ou des arbres, avec un accès plus rapide). En revanche, le parallélisme peut simplifier plusieurs opérations de l'algorithme.
- Source numérique : Étant donné que l'espace de recherche est généralement très grand, le programme séquentiel peut devoir rechercher une grande partie avant de trouver la solution requise. En revanche, la version parallèle peut trouver la solution plus rapidement en raison du changement de l'ordre dans lequel l'espace est recherché.
- Source physique : lors du passage d'une machine séquentielle à une machine parallèle, il arrive souvent que l'on obtienne plus qu'une simple augmentation de la puissance du processeur. D'autres ressources, telles que la mémoire, la mémoire cache, etc. peuvent également augmenter linéairement avec le nombre de processeurs. Une méta-heuristique parallèle peut atteindre une accélération super-linéaire en tirant parti de ces ressources supplémentaires. Autrement dit, dans une plate-forme parallèle, chaque sous-population est une machine différente et elle peut tenir dans des caches tandis que dans une exécution séquentielle (voir Figure IV.2), toutes les sous-populations sont dans la même machine, nous devons utiliser la mémoire principale pour les stocker en raison du manque de capacité de mémoire cache.

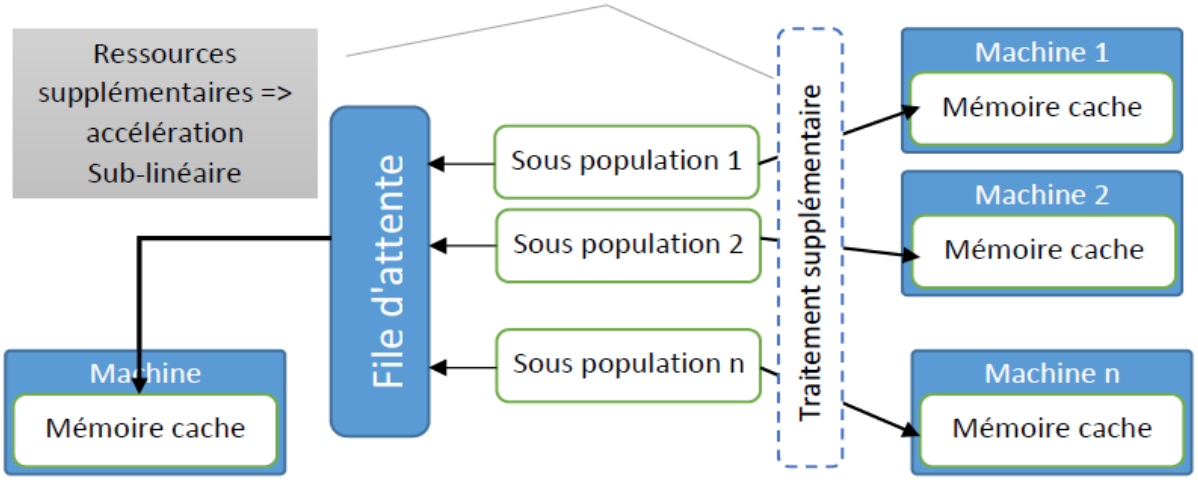


Figure IV.2 Utilisation parallèle et séquentielle des ressources.

L'accélération linéaire est rare dans la pratique, car il y a un travail supplémentaire impliqué dans la distribution du travail aux processeurs et leur coordination. De plus, un algorithme série optimal peut être capable de faire moins de travail globalement qu'un algorithme parallèle optimal pour certains problèmes, donc l'accélération peut être sub-linéaire en *m*, même sur des machines idéales. L'accélération linéaire est généralement considérée comme optimale car nous pouvons sérialiser l'algorithme parallèle, et l'exécuter sur une machine série avec un ralentissement linéaire comme pire scénario de référence. Cependant, pour la plupart, l'accélération sub-linéaire est la norme.

IV.3.2 Autres métriques

Bien que l'accélération soit une métrique largement utilisée, il existe d'autres mesures des performances d'une méta-heuristique parallèle.

L'efficacité

C'est une normalisation de l'accélération et permet de comparer différents algorithmes : $e_m = s_m / m$ ($e_m = 1$ signifie accélération linéaire)*;*

Il existe plusieurs variantes de la métrique d'efficacité. Par exemple, **l'efficacité incrémentielle** montre la fraction d'amélioration du temps par rapport à l'ajout d'un autre processeur ; il est également très utilisé lorsque les temps d'uni-processeur sont inconnus. Cette métrique a ensuite été généralisée pour mesurer l'amélioration atteinte en augmentant le nombre de processeurs de *n* à *m*.

$$ie_{n\to m} = \frac{n*M[T_n]}{m*M[T_m]} \qquad \textit{IV.1}$$

Accélération mise à l'échelle

Les métriques précédentes indiquent l'amélioration résultant de l'utilisation d'éléments de traitement supplémentaires, mais elles ne mesurent pas l'utilisation de la mémoire disponible.

L'accélération mise à l'échelle résout ce problème et permet de mesurer la pleine utilisation des ressources matérielles :

$$gs_m = \frac{Temps\ estimé\ pour\ résoudre\ le\ problème\ de\ taille\ n * m\ sur\ 1\ processeur}{Temps\ réel\ pour\ résoudre\ le\ problème\ de\ taille\ n * m\ sur\ m\ processeurs}$$

Où *n* est la taille du plus gros problème qui peut être stocké dans la mémoire associée à un processeur. Son inconvénient majeur est qu'il est difficile d'effectuer une estimation précise du temps série et qu'elle n'est pas pratique pour de nombreux problèmes.

Estimation du temps du processeur unique

La « mise à l'échelle » est étroitement liée à l'accélération mise à l'échelle, mais elle n'est pas basée sur une estimation du temps du processeur unique :

$$es_{m,n} = \frac{Temps\ pour\ résoudre\ k\ problèmes\ sur\ m\ processeurs}{Temps\ pour\ résoudre\ n * k\ problèmes\ sur\ n * m\ processeurs}$$

Cette métrique mesure la capacité de l'algorithme à résoudre une tâche n fois plus grande sur un système n fois plus grand en même temps que le système d'origine. Par conséquent, l'accélération linéaire se produit lorsque $es_{m,n} = 1$.

Fraction en série de l'algorithme

Karp et Flatt (1990) ont mis au point une métrique supplémentaire intéressante pour mesurer les performances de tout algorithme parallèle qui peut nous aider à identifier des effets beaucoup plus subtils que l'utilisation de l'accélération seule. Ils l'appellent la fraction en série de l'algorithme.

$$f_m = \frac{1/s_m - 1/m}{1 - 1/m} \qquad \textit{IV.2}$$

La fraction en série doit rester constante pour un algorithme lors de l'utilisation de différentes valeurs de *m*. Nous pouvons encore dire que le résultat est bon si f_m reste constant pour différentes valeurs de *m*, car la perte d'efficacité est due au parallélisme limité du programme. D'un autre côté, des incréments en douceur de f_m avec *m* est un avertissement que la granularité des tâches parallèles est trop fine. Un troisième scénario est possible, dans lequel une réduction significative de f_m se produit avec des valeurs croissantes de *m*, indiquant quelque chose qui ressemble à une accélération super-linéaire. Si une accélération super-linéaire se produit, alors f_m prendrait une valeur négative.

IV.4 Algorithmes génétiques structurés

IV.4.1 Panmixie

Dans le domaine des algorithmes génétiques (AGs), il est habituel de trouver des algorithmes dont la structure de population est panmictique. Ainsi, la sélection a lieu à l'échelle

globale et tout individu peut potentiellement se reproduire avec n'importe quel autre (Varghese & Raj 2016). Il en va de même pour l'opérateur de remplacement, où tout individu peut potentiellement être retiré du pool et remplacé par un nouveau. En revanche, il existe un modèle de sélection différent (décentralisé), dans lequel les individus sont disposés spatialement, donnant ainsi lieu à un AG structuré. La plupart des autres opérateurs, tels que la recombinaison ou la mutation, peuvent être facilement appliqués à ces deux modèles (panmictiques et structurées).

Il existe deux classes panmictiques populaires, dans le premier, appelé modèle **générationnel**, une toute nouvelle population de *X* individus remplace l'ancienne. Le deuxième type est appelé **stationnaire** car généralement un ou deux nouveaux individus sont créés à chaque étape de l'algorithme, puis ils sont réinsérés dans la population. Nous rapportons ces deux types en termes de nombre de nouveaux individus insérés dans la population de la prochaine génération.

Les versions centralisées de la sélection se trouvent généralement dans l'AG série bien que certaines implémentations parallèles puissent être trouvées comme l'approche du parallélisme global : évalue en parallèle les individus de la population tout en utilisant une sélection centralisée effectuée séquentiellement dans le processeur principal (Levine 1996). Cet algorithme est conceptuellement le même que celui séquentiel, bien qu'il soit plus rapide, en particulier pour les fonctions objectif chronophages. Habituellement, les autres parties de l'algorithme ne valent pas la parallélisation, à moins qu'un certain principe de structuration de population ne soit utilisé.

La plupart des AGP trouvés dans la littérature mettent en œuvre une sorte de disposition spatiale pour les individus, puis parallélisent les blocs résultants dans un pool de processeurs. Nous devons souligner que la parallélisation est obtenue en structurant d'abord l'algorithme panmictique puis en le parallélisant. C'est pourquoi nous distinguons entre structurer des populations et faire des implémentations parallèles, car le même AG structurée peut admettre de nombreuses implémentations parallèles différentes.

IV.4.2 Décentralisation de la population

Il existe une longue tradition d'utilisation des populations structurées dans le calcul évolutif (EC), notamment associée aux implémentations parallèles. Parmi les types d'algorithmes génétiques (AG) structurés les plus connus, les d'algorithmes génétiques distribués (AGD) et les d'algorithmes génétiques cellulaires (AGC) (Luo & al. 2018).

La décentralisation d'une seule population peut être réalisée en la divisant en plusieurs sous-populations, où les AG sont exécutés en effectuant des échanges clairsemés (migrations) d'individus (AGD), ou sous forme d'îles (AGC). Dans les AGD, des paramètres supplémentaires contrôlant le moment de la migration et la manière dont les migrants sont sélectionnés/incorporés depuis/vers l'île source/cible sont nécessaires (Tanese 1989 ; Belding 1995). Dans les AGC, l'existence de petites îles superposées permet d'explorer l'espace de recherche (Lance 2019). Ces deux types d'AG semblent fournir un meilleur échantillonnage de

l'espace de recherche et améliorer le comportement et le temps d'exécution de l'algorithme dans de nombreux cas (Cooper & Hinde 2003 ; Lee 2018 ; Ying 2008).

Dans les AGC, la boucle de reproduction est réalisée à l'intérieur de chacun des nombreux pools d'individus. Un individu donné à son propre pool de partenaires potentiels, défini par ses voisins .Cette structure avec des îles superposées linéaires ou à deux dimensions est utilisée pour fournir une diffusion en douceur de bonnes solutions à travers la grille. Un AGC peut être implémenté dans une plateforme MIMD (Multiple Instruction Multiple Data) à mémoire distribuée (de Melo & al.2020) (cluster, multi-nœuds,…), bien que son implémentation plus directe soit sur une plateforme SIMD (Single Instruction Multiple Data) (FPGA ou GPU).

Un AGD est un modèle multi-pools effectuant des échanges clairsemés d'individus entre les populations élémentaires. Ce modèle peut être facilement implémenté dans les plateformes MIMD à mémoire distribuée, C'est l'une des principales raisons de sa popularité au cours de la dernière décennie. Une politique de migration contrôle le type d'AGD utilisé. La politique de migration doit définir la topologie lors de la migration, quels individus sont échangés, la synchronisation entre les sous-populations et le type d'intégration des individus échangés au sein des sous-populations cibles. Le temps d'exécution et le nombre d'évaluations sont potentiellement réduits grâce à sa recherche séparée dans plusieurs régions.

IV.5 Algorithmes génétiques parallèles

L'objectif est de présenter une vision structurée des modèles parallèles et des implémentations parallèles des AG. Par conséquent, cette partie est consacrée à la description du modèle parallèle utilisé pour paralléliser un AG, ainsi qu'à la classification des implémentations parallèles.

IV.5.1 Modèles parallèles

Les algorithmes génétiques parallèles (AGP) sont des implémentations parallèles d'AG qui peuvent fournir des gains considérables en termes de performances et d'évolutivité. Les AGP peuvent implémentés sur des réseaux d'ordinateurs hétérogènes ou sur des mainframes parallèles. Nous passons en revue une brève description des principaux modèles conceptuels et des paradigmes d'AG parallèles qui sont mis en œuvre dans la littérature.

IV.5.2 Modèle indépendant

Ce modèle peut être inclus par d'autres pour l'utilisation d'un pool de processeurs afin d'accélérer l'exécution d'un algorithme séquentiel, simplement parce que des exécutions indépendantes peuvent être effectuées plus rapidement en utilisant plusieurs processeurs qu'en utilisant un seul. Dans ce cas, aucune interaction n'existe entre les analyses indépendantes. Cette méthode extrêmement simple de travail simultané peut être très utile. Par exemple, il peut être utilisé pour exécuter plusieurs versions du même problème avec des conditions initiales différentes, permettant ainsi de recueillir des statistiques sur le problème en peu de temps. Étant

donné que les AG sont de nature stochastique, la disponibilité de ce type de statistiques est très importante.

IV.5.3 Modèle maître-esclave

Il consiste à répartir les évaluations des fonctions objectifs entre plusieurs processeurs esclaves tandis que la boucle principale de l'AG est exécutée dans un processeur maître. Ce paradigme parallèle est assez simple à mettre en œuvre et son exploration de l'espace de recherche est conceptuellement identique à celle d'un AG s'exécutant sur un processeur série (Singh 2018). En d'autres termes, le nombre de processeurs utilisés est indépendant des solutions évaluées, à l'exception du temps. Ce paradigme est illustré sur la figure IV.3, où le processeur maître envoie des paramètres (ceux nécessaires à l'évaluation de la fonction objectif) aux esclaves ; les valeurs des fonctions objectifs sont ensuite renvoyées après calcul.

Ce modèle a tendance à présenter un goulot d'étranglement pour quelques dizaines de processeurs, empêchant ainsi généralement l'évolutivité. Le processeur maître contrôle la parallélisation des tâches d'évaluation de la fonction objectif (et éventuellement l'affectation et la transformation de la condition physique) effectuées par les esclaves. Il est généralement moins efficace car l'évaluation devient plus coûteuse à calculer en raison de la surcharge de communication par rapport au temps d'évaluation.

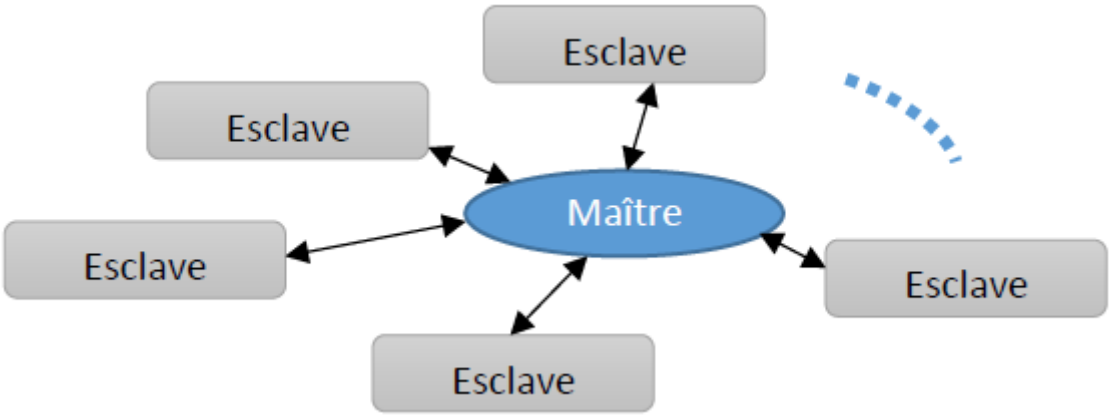


Figure IV.3 Modèle maître-esclave

IV.5.4 Modèle distribué

Dans les AG distribués, la population est structurée en sous-populations plus petites relativement isolées les unes des autres. Les AG parallèles basées sur ce paradigme sont parfois appelées AG multi-population. La principale caractéristique de ce type d'algorithme est que des individus au sein d'une sous-population peuvent parfois migrer vers une autre. Ce paradigme est illustré sur la figure IV.4. Notez que les canaux de communication indiqués sont logiques ; des affectations spécifiques sont effectuées dans le cadre de la stratégie de migration de l'AG et sont mappées à un réseau physique. Sur le plan conceptuel, la population globale d'AG est divisée en un certain nombre de sous-populations indépendantes et séparées. Un autre point de vue est celui de plusieurs algorithmes distincts s'exécutant simultanément. Les individus

migrent entre une île en particulier et ses voisins, bien que ces îles évoluent généralement de manière isolée pour la majorité du temps d'exécution de l'AG. Ici, les opérateurs génétiques (sélection, mutation et croisement) ont lieu au sein de chaque île, ce qui signifie que chaque île peut rechercher dans des régions très différentes de tout l'espace de recherche par rapport aux autres. Comme indiqué précédemment, chaque sous-population peut également avoir des valeurs de paramètres différentes (Wang & al. 2018). Le modèle distribué nécessite l'identification d'une politique de migration appropriée. Les principaux paramètres de la politique de migration sont les suivants :

- **Écart de migration** : Puisqu'un AGD fait généralement des échanges d'individus clairsemés entre les sous-populations, il faut définir l'écart de migration, c'est le nombre d'étapes dans chaque sous-population entre deux échanges successifs (étapes d'évolution isolée). Il peut être activé dans chaque sous-population soit périodiquement, soit en utilisant une probabilité donnée pour décider à chaque étape si la migration aura lieu ou non.
- **Sélection/remplacement** : Ce paramètre décide comment sélectionner les solutions d'émigrants et quelles solutions doivent être remplacées par les individus. Il est très courant dans les AG distribuées en parallèle d'utiliser les mêmes opérateurs de sélection / remplacement.
- **Topologie** : Ce paramètre définit le voisin de chaque île, c'est-à-dire les îles qu'une sous-population concrète peut envoyer (ou recevoir) à des individus. La nomenclature traditionnelle divise les AG parallèles en stepping-stone modèle et island (île) modele, selon que les individus peuvent migrer librement vers n'importe quelle sous-population ou s'ils sont limités à migrer vers des îles géographiquement proches, respectivement.
- **Taux de migration** : Ce paramètre détermine le nombre d'individus qui subissent une migration dans chaque échange. Sa valeur peut être donnée en pourcentage de la taille de la population ou bien en valeur absolue.

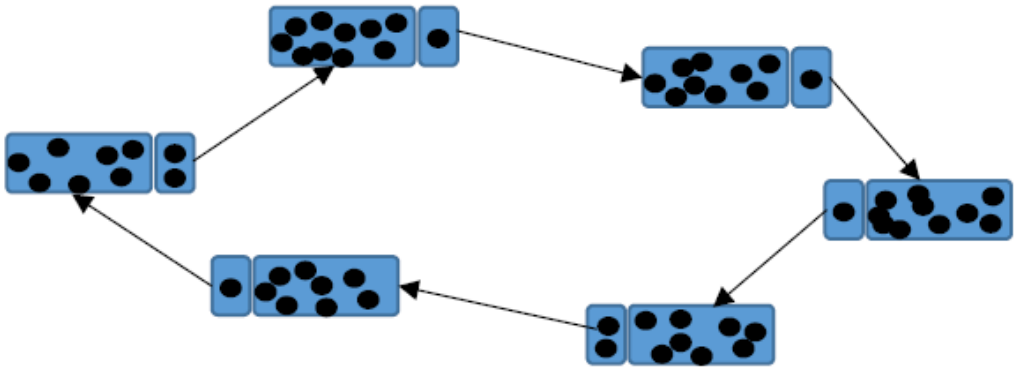


Figure IV.4 Modèle distribué

IV.5.5 Modèle cellulaire

Comme le modèle maître-esclave, le paradigme d'AG cellulaire parallèle traite une population conceptuelle unique, où chaque unité de traitement ne contient que quelques individus (deux au maximum). C'est la raison pour laquelle ce modèle est parfois appelé parallélisme à grain fin (fine grained). La principale caractéristique de ce modèle est la structuration de la population en pool de voisinages, où les individus ne peuvent interagir

qu'avec leurs voisins. Ainsi, les bonnes solutions émergent dans topologie globale (Zhao & al. 2016), elles sont lentement diffusées dans l'ensemble de la population. Ce modèle est illustré par la figure IV.5.

Les AG cellulaires ont été initialement conçus pour fonctionner dans des machines massivement parallèles, bien que le modèle lui-même ait été adopté également pour les systèmes distribués (Maruyama et al.1993) et les machines à monoprocesseur (Cheng et Gen 2019). Ce problème peut être clairement énoncé, car de nombreux chercheurs ont toujours en tête, la relation entre les AG massivement parallèles et les AG cellulaires. Les unités de traitement graphique (GPU) et le réseau de portes programmable sur site (FPGA) représentent de nouveaux prise en charge matérielle pour exécuter réellement des AGC sur des plateformes massivement parallèles (Cheng & Gen 2019 ; Thirer 2020).

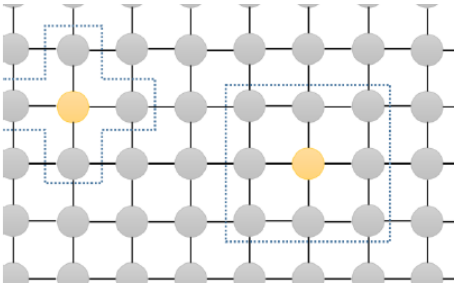


Figure IV.5 Modèle cellulaire

Un AGC peut être vu comme un automate cellulaire (AC) avec des règles probabilistes réinscriptibles, où l'alphabet de l'AC est équivalent à l'ensemble potentiel de chromosomes dans l'espace de recherche, c'est-à-dire au potentiel nombre de solutions au problème. Par conséquent, si nous considérons les AGC comme une sorte d'AC, il est possible d'importer des outils analytiques et des modèles et propositions existants dans le domaine des AC vers des AGC afin d'améliorer leurs performances.

La littérature présente deux types différents d'AGC selon la façon dont le cycle de reproduction est appliqué aux individus.

- D'une part, si le cycle est appliqué à tous les individus simultanément, l'AGC est dit **synchrone**, car les individus de la population de la génération suivante sont formellement créés en même temps, de manière concurrente.

- D'un autre côté, si nous mettons à jour séquentiellement les nouveaux individus de la population avec une politique d'ordre particulière, nous avons un AGC **asynchrone**.

Le comportement d'un algorithme est vraiment affecté par la politique de mise à jour des individus, ainsi que par d'autres paramètres comme la taille de l'île. De plus, l'ordre de visite aux individus pour mise à jour dans le cas asynchrone est également un problème capital dans le comportement de l'algorithme. Nous présentons les principales politiques de mise à jour des individus dans les AGC asynchrones :

•**Balayage de ligne (BL)** : C'est la méthode la plus simple. Il s'agit de mettre à jour séquentiellement les individus de la population ligne par ligne.

•**Balayage aléatoire (BA)** : Dans ce cas, la cellule suivante à mettre à jour est sélectionnée avec une probabilité uniforme sans remplacement (c'est-à-dire qu'il n'est pas possible de visiter une cellule deux fois en une génération).

•**Nouveau balayage aléatoire (NBA)** : Similaire à BA, avec la différence d'utiliser une nouvelle permutation aléatoire de cellules dans chaque génération.

•**Balayage aléatoire uniforme (BAU)** : Dans cette dernière méthode, le prochain individu à visiter est choisi au hasard avec une probabilité uniforme parmi toutes les composantes de la population avec remplacement (il est donc possible de visiter une cellule plus d'une fois dans la même génération).

Une génération est définie comme la mise à jour séquentielle de n individus, ce qui correspond à la mise à jour de tous les individus de la population dans les cas des politiques BL, BA et NBA, et éventuellement moins de n individus différents pour la méthode BAU (car certaines cellules peuvent être mises à jour plusieurs fois dans la même génération). Il est intéressant de remarquer qu'à l'exception de BL, les autres politiques de mise à jour asynchrone sont stochastiques, ce qui représente une source supplémentaire de non déterminisme.

IV.6 Nouvelles tendances dans les algorithmes génétiques parallèles

Nous nous référerons dans cette section aux directions de recherche d'algorithmes génétiques parallèles sur certaines des lignes les plus prometteuses où les réalisations futures doivent être notées.

1. **Résoudre les problèmes d'optimisation des fonctions dynamiques (OFD)** : les AGP auront un rôle important dans l'optimisation des fonctions complexes dont les optima varient dans le temps. De tels problèmes consistent à optimiser un ensemble successif de fonctions de fitness, chacune étant généralement une perturbation de la précédente. Les processus industriels, comme la planification de tâches réelles, et les tâches de la vie quotidienne telles que le contrôle des files d'attente dynamiques peuvent être traités avec des modèles dynamiques. Certains AGP, comme les AGC et les AGD, peuvent affronter de tels environnements OFD avec succès grâce à leurs améliorations de la diversité naturelle et à leurs caractéristiques (Runkler & Bezdek 2019).
2. **Développer des enjeux théoriques** : l'amélioration des explications formelles sur l'influence des paramètres sur la convergence et la recherche d'AGP dotera la communauté de recherche d'outils permettant d'analyser, de comprendre et de personnaliser un AG pour n'importe quel problème donné (Ferrucci & al. 2018).
3. **Exécution d'AGP sur des clusters géographiquement séparés** : Cela permettra à l'utilisateur d'utiliser des ressources de calcul peu localisées de manière méta-informatique afin de résoudre son problème d'optimisation. Un exemple distingué d'un tel système est d'utiliser le Web comme pool de processeurs pour exécuter des algorithmes afin de résoudre le même problème. En particulier, le Grid-computing et le Peer-to-Peer (Helal & al. 2017) sont devenus une véritable alternative au supercalculateur traditionnel pour le

développement d'applications parallèles qui exploitent des ressources informatiques massives. C'est un grand défi, car de nos jours, les cadres de grille et de P2P pour la méta-heuristique ne font qu'émerger.

4. **Nouvelles plates-formes parallèles** : exploiter de nouvelles architectures matérielles comme les GPU et les processeurs multi-cœurs en développant de nouvelles AGP spécialisées bénéficiant de ces architectures pour fonctionner plus efficacement (Abbasi & al. 2020).
5. **Benchmarking des techniques de soft computing** : à l'heure actuelle, il est clair qu'un ensemble de problèmes largement disponible, large et standard est nécessaire pour évaluer la qualité des AGP existants et nouveaux. Des exemples de problèmes de difficulté différente, spécialement ciblés pour tester le comportement des AGP et des techniques connexes, peuvent grandement aider les praticiens à choisir l'algorithme AGP ou hybride le plus approprié pour la tâche à accomplir.

IV.7 Déclaration des résultats dans les algorithmes génétiques parallèles

En général, l'objectif de la recherche scientifique est de présenter une nouvelle approche ou un nouvel algorithme qui fonctionne mieux dans un certain sens que les algorithmes existants. Cela nécessite des tests expérimentaux pour comparer le nouvel algorithme par rapport au reste. En général c'est difficile de faire des comparaisons équitables entre les algorithmes. La raison en est que nous devons garantir le même environnement expérimental (ressources, protocoles, systèmes, ...) et, en supposant que cela est correctement fait, que plus tard peut déduire des conclusions différentes à partir des mêmes résultats en fonction des mesures utilisées. Ceci est particulièrement important pour les méthodes non déterministes.

IV.7.1 Expérimentation

Le domaine et les instances du problème est le premier choix d'un chercheur qui doit faire pour tester son algorithme. Cette décision dépend des buts de l'expérimentation. On peut distinguer deux objectifs clairement différents : l'optimisation et l'apprentissage.

L'optimisation est la sélection du meilleur élément (par rapport à un critère) parmi un ensemble d'alternatives disponibles dans la conception d'une méta-heuristique qui bat les autres sur un problème donné ou un ensemble de problèmes. Ce type de recherche expérimentale finit par établir la supériorité d'une heuristique donnée sur les autres. Dans ce scénario, les chercheurs ne devraient pas se limiter à établir qu'une méta-heuristique est meilleure qu'une autre d'une manière ou d'une autre, mais aussi à rechercher pourquoi (Franke & Sarstedt 2019).Une décision importante est l'instance utilisée. L'ensemble d'instances doit être suffisamment complexe pour obtenir des résultats intéressants et doit avoir une variété suffisante pour permettre une certaine généralisation. Les générateurs de problèmes (Ng & al. 2014) sont particulièrement bons pour une analyse variée et large.

IV.8 Mesurer les performances

Une fois que nous avons choisi les instances que nous allons utiliser et les facteurs que nous allons analyser, nous devons sélectionner les mesures appropriées pour l'objectif de notre étude (Bowlin 1998). L'objectif d'une méta-heuristique est de trouver une bonne solution dans un délai raisonnable. Par conséquent, le choix des mesures de performance pour les expériences avec l'heuristique implique nécessairement à la fois la qualité de la solution et l'effort computationnel. En raison de la nature stochastique de la méta-heuristique, un certain nombre d'expériences indépendantes doivent être menées pour obtenir suffisamment de données expérimentales. Les mesures de performance de ces heuristiques sont basées sur une sorte de statistiques.

IV.8.1 Qualité des solutions

Il s'agit de l'un des paramètres les plus importants pour évaluer les performances d'un algorithme. Pour les cas où la solution optimale est connue, on peut facilement définir une mesure, le taux ou le nombre de tentatives de réussite. Cette mesure peut être définie comme le pourcentage d'exécutions se terminant avec succès (% hits). Mais cette métrique ne peut pas être utilisée dans tous les cas, car, il existe des problèmes où la solution optimale n'est pas connue du tout et où une limite n'est pas non plus disponible. Dans d'autres cas, bien que l'optimum soit connu, son calcul tarde trop et les exigences doivent être assouplies pour trouver une bonne approximation dans un délai raisonnable. C'est aussi une pratique courante en méta-heuristique que les expériences aient une limite spécifique d'effort de computationnel (un nombre donné de points d'espace de recherche visités ou un temps d'exécution maximum). Dans tous ces cas, lorsque l'optimum n'est pas connu ou localisé, des mesures statistiques sont également utilisées. Les mesures les plus courantes incluent la moyenne et la médiane des solutions les plus performantes, telles que fitness sur toutes les exécutions. Ces valeurs peuvent être calculées pour tout problème. Pour chaque série d'une méta-heuristique donnée, la meilleure fitness peut être définie comme meilleure solution à la fin. Pour les méta-heuristiques parallèles, elle est définie comme la meilleure solution globale trouvée par l'ensemble des exécutions. Dans un problème où l'optimum est connu, rien ne nous empêche d'utiliser à la fois le % de hits et la médiane/moyenne de la qualité/effort finale. De plus, toutes les combinaisons de valeurs faibles/élevées peuvent se produire pour ces mesures. Nous pouvons obtenir un faible nombre de tentatives contre une précision moyenne/médiane élevée ; cela, par exemple, indique une méthode robuste, qui atteint rarement la solution optimale. Une combinaison opposée est également possible mais elle n'est pas courante. Dans ce cas, l'algorithme atteint l'optimum dans plusieurs exécutions ou la majorité donne une très mauvaise fitness. En pratique, une simple comparaison entre deux moyennes ou médianes pourrait ne pas donner le même résultat qu'une comparaison entre deux Loi de probabilité. En général, il est nécessaire d'offrir des valeurs statistiques supplémentaires telles que la variance, et d'effectuer une analyse statistique globale pour s'assurer que les conclusions sont significatives et pas seulement de l’aléatoire.

IV.8.2 Effort computationnel

Le temps de calcul maximum reste toujours le jugement, malgré l'importance des algorithmes qui produisent des solutions plus précises. Au sein de la méta-heuristique, l'effort computationnel est généralement mesuré par le nombre d'évaluations de la fonction objectif et le temps d'exécution. En général, le nombre d'évaluations est défini en fonction du nombre de points de l'espace de recherche visité. De nombreux chercheurs préfèrent le nombre d'évaluations comme moyen de mesurer l'effort computationnel, car il élimine les effets d'implémentations, de logiciels et de matériel particuliers, rendant ainsi les comparaisons indépendantes de ces détails. Mais cette mesure peut être trompeuse dans plusieurs cas dans le domaine des méthodes parallèles. Par exemple, si certaines évaluations prennent plus de temps que d'autres (Crainic 2019) ou si une évaluation peut être effectuée très rapidement, le nombre d'évaluations ne reflète pas correctement le temps d'exécution de l'algorithme. Dans certains cas, le concept d'évaluation pourrait même ne pas exister. De plus, l'objectif du parallélisme n'est pas la réduction du nombre d'évaluations mais la réduction du temps. Par conséquent, on doit signaler les deux paramètres pour mesurer l'effort computationnel. Il est très courant d'utiliser les moyens évaluations/temps pour une solution, définis au cours de ces exécutions qui aboutissent à une solution. Cette pratique présente des inconvénients lorsque l'on trouve des solutions différentes, l'utilisation du temps/effort total d'exécution pour comparer les algorithmes devient difficile à interpréter du point de vue parallélisme. Pour mieux rapproché les comparaisons, un temps/effort prédéfini peut fournir une métrique intéressante pour donner une réflexion sur la qualité de solution des algorithmes.

IV.8.3 Analyses statistiques

La comparaison entre deux valeurs moyennes peut être différente de la comparaison entre deux distributions. Par conséquent, des méthodes statistiques devraient être employées dans la mesure du possible pour indiquer la force des relations entre les différents facteurs et les mesures de performance, ce qui conduit à un test pour une analyse de variance pour garantir la signification statistique des résultats, c'est-à-dire déterminer si un effet observé est probablement dû à des erreurs d'échantillonnage. Plusieurs méthodes statistiques et les conditions pour les applications sont présentées dans la figure IV.6. Premièrement, nous devons décider entre les tests non paramétriques et paramétriques ; lorsque les données suivent une loi normale, utilisez des méthodes non paramétriques sinon un test paramétrique peut être utilisé. Le test de Kolmogorov-Smirnov est une méthode puissante, précise et à faible coût pour vérifier la normalité des données. Pour les données qui ne suivent pas une distribution normale, un large éventail de méthodes a été proposé dans la littérature (Sarkar 2018), Toutes ces méthodes supposent plusieurs hypothèses pour obtenir un comportement approprié. La notion sous-jacente d'ANOVA est de supposer que chaque variation non aléatoire des observations expérimentales est due à des différences de performance moyenne à différents niveaux des facteurs expérimentaux. Les deux analyses, ne peuvent être appliquées que si la distribution de la source est normale. En méta-heuristique, la distribution pourrait facilement être non normale. Pour ce cas, on utilise le théorème de la limite centrale indique que la somme de nombreuses variables aléatoires distribuées de manière identique tend vers une gaussienne. Ainsi, la

moyenne de tout l'ensemble d'échantillons tend vers une distribution normale. Mais dans plusieurs cas, le théorème central limite n'est pas utile et trop général. Dans ces cas, il existe une multitude de techniques non paramétriques (comme Test des rangs signés de Wilcoxon) qui peuvent être utilisés pour promouvoir les arguments, même si les résultats n'ont aucune différence statistique entre la qualité des solutions produites par les méta-heuristiques (Suresh & Sakthi 2019).

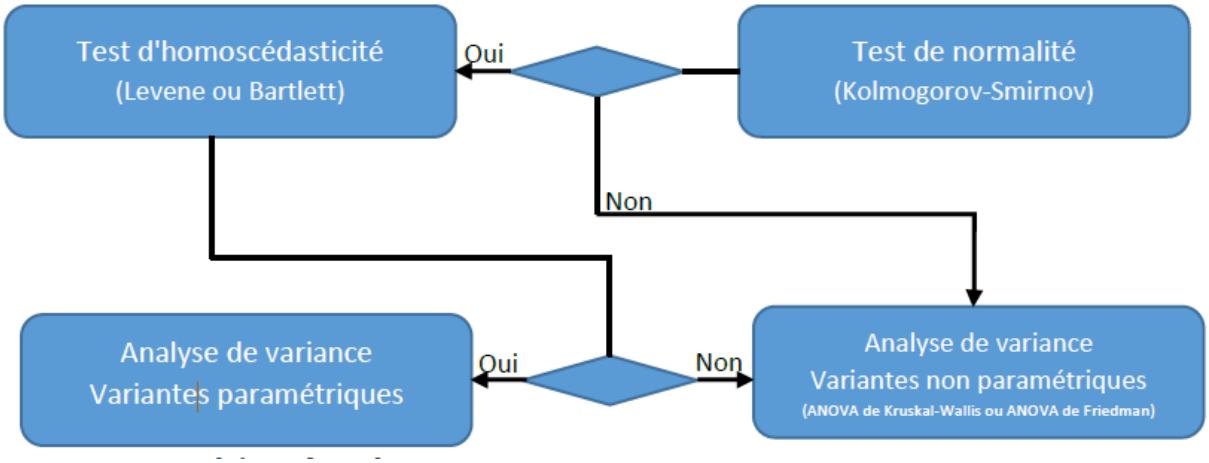


Figure IV.6 Schéma d'analyse statistique

IV.9 Approche proposée

Plusieurs ensembles d'instances sont utilisés pour comparer les résultats des calculs d'algorithmes sur le Job-Shop. Dans les recherches que nous avons effectuées, nous avons utilisé les recherches de limites les plus connues pour ces cas, car un bon nombre des limites signalées n'étaient pas les plus connues. Parfois, seules les limites des tableaux ont été utilisées, et non celles indiquées dans les textes des articles cités. Pour permettre une comparaison avec les résultats obtenus, nous avons donc réalisé une étude bibliographique sur les bornes les plus connues pour certains ensembles d'instances de référence. Les ensembles étudiés sont les ensembles de référence bien connus :

dmu (Demirkol & al. 1998), orb (Applegate & Cook 1991), ft (Fisher & Thompson 1963), la (Lawrence 1984), abz (Adams & al. 1988), swv (Storer & al. 1992), yn (Yamada & Nakano 1992), ta (Taillard 1993).

Nous présentons dans ce qui suit la représentation génétique proposée ainsi que les opérateurs génétiques et de la fonction fitness, puis dans le dernier chapitre nous approfondirons davantage sur la conception proposée ainsi que sur l'implémentation des deux architectures proposées dans des unités de traitement graphique.

IV.9.1 Représentations génétiques proposée

En tenant compte des contraintes de précédence, une représentation naturelle n'est pas aussi simple que le problème du voyageur de commerce (TSP). Pour certains problèmes d'optimisation combinatoire, il est généralement simple de réparer un chromosome irréalisable ou illégal et la procédure de réparation surpasse les différentes techniques telles que la stratégie

de pénalisation ou de rejet, mais dans tous les cas, il y a un temps de calcul pour arriver à une solution irréalisable, ou pour la réparer. Pour obtenir une représentation optimale, les chromosomes générés doivent produire des calendriers réalisables à tous les niveaux. Dans cette représentation, chaque gène représente une opération et une solution a été codé comme une séquence d'opérations, un chromosome contient n × m gènes, m fois est l'apparition de chaque chromosome (Figure IV.7.).

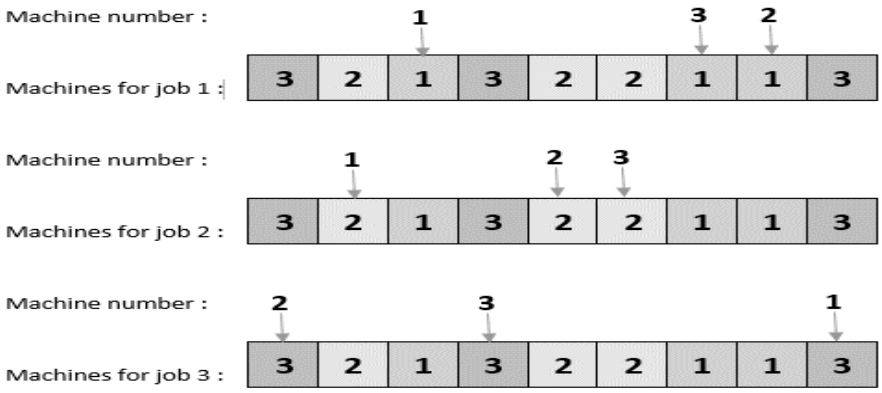


Figure IV.7 Représentation génétique pour m niveau (m=3).

IV.9.2 Opérateurs génétiques

Le croisement à deux points dans notre cas nécessite la sélection de deux points sur les parents, et tout entre les deux points est échangé entre eux, ce qui rend deux offspring. Le problème est que le résultat peut donner des chromosomes irréalisables, nous proposons d'échanger des cellules en double entre la partie migrante et les premières cellules de la partie restante (Figure IV.8.).La mutation est un processus de changement entre deux gènes choisis au hasard d'un individu choisi.

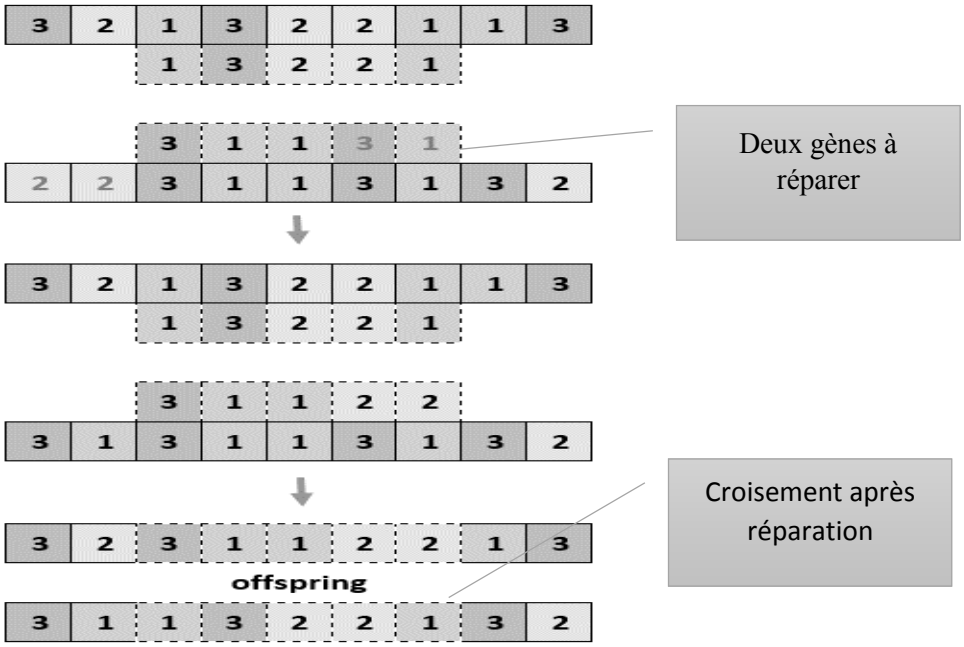


Figure IV.8 Exemple de croisement.

La fonction de fitness c'est la probabilité de continuité d'un individu, nous utilisons la formule F (x) = CmaxP-CmaxX où CmaxX est le Makespan de l'individu X et CmaxP est la valeur de Makespan la plus élevée de la population.

IV.9.3 Fitness et voisinages

Le voisinage de Moore et de Von-Neumann (Chopard 2012) sont les deux structures communes principalement considérées, la nécessité d'obtenir un résultat à court terme nous a conduits à utiliser le voisinage de Von-Neumann qui est défini par l'ensemble des quatre cellules qui sont situés en direction axiale (Figure IV.9.). Le choix de voisinage peut être formulé comme :

Voisinage du thread *i* :
Thread *i* – 1 (si *dim* % *i* <> 0)
Thread *i* + 1 (si *dim* % *i* <> 0)
Thread *i* – *dim*(si *i* – *dim* > 0)
Thread *i* + *dim* (si *i* + *dim* <= *dim2*)

Les opérateurs génétiques fonctionnant en parallèle, les mises à jour des individus doivent être asynchrones en fonction de l'état initial du voisinage. Nous choisissons l'Offspring pour chaque recombinaison avec une solution voisine par un mécanisme de sélection proportionné basé sur la fitness des solutions (c'est-à-dire que la probabilité de choisir une solution voisine est donnée par sa valeur de fitness sur la somme de la fitness de tous les voisins). Par conséquent, le parent c sera choisi avec probabilité :

$$Psc = \frac{Fitness\ (S_c)}{\sum_{i=0}^{n} Fitness(S_i)}$$

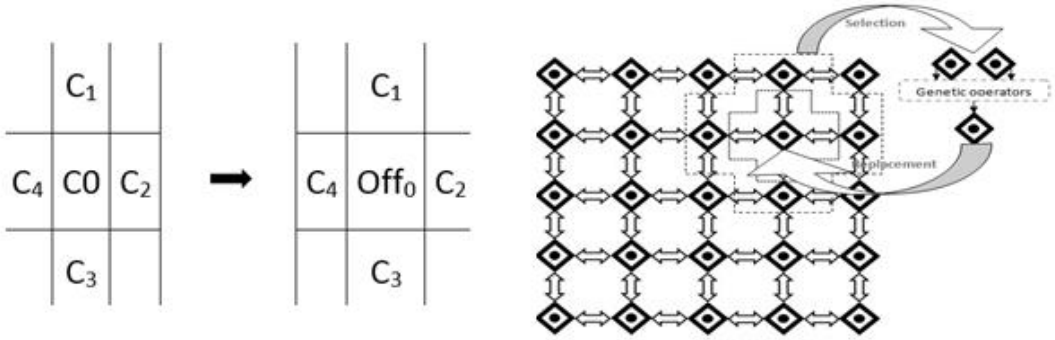


Figure IV.9 Mise à jour du voisinage

IV.9.4 Algorithme de résolution

Nous avons fait l'hypothèse implicite que le matériel génétique, ainsi que les conditions évolutives, telles que les méthodes de sélection et de recombinaison, étaient les mêmes pour tous les individus et toutes les populations d'un AG structurée (homogènes).

L'algorithme proposé passera par 4 phases (Figure IV.10) :

1. **Paramètres**

   Cette phase définit le nombre d'individus, le nombre de sous-populations et l'intervalle de migration.

2. **Initialisation**

   Les individus sont initialisés à une valeur aléatoire et font une évaluation préliminaire.

3. **Opérateur génétique**

   En plus des opérations génétiques, une étape de synchronisation est obligatoire dans cette phase car l'exécution est en concurrence.

4. **Migration**

   C'est là que se trouve l'hybridation entre les deux modèles (modèle cellulaire et distribué), dans cette phase chaque sous-population envoie et reçoit des individus et effectue le remplacement.

Paramètre :
S - Ensemble de sous-populations
Ind - Ensemble d'individus
M - Intervalle de migration
i - Séquence individuelle
j - Séquence de sous-population
N - Nombre d'individus migrés
Sortie : représentation basée sur les opérations
Initialisation :
Pour chaque Ind ϵ *S*, Initialiser Ind_i aléatoirement ;
Pour chaque Ind ϵ *S* Evaluer_Fitness
Tant que ! Condition_arrêt
Debut
Pour chaque S_j
Exécuter_en_Parallèle
    Sélectionner les individus
    Recombiner les individus sélectionnés
    Muter des individus
    Synchronisation
    Evaluer_Fitness
Pour chaque *M*
Envoyez *N* les meilleurs individus de la sous-population voisine
Recevez *N* les meilleurs individus de la sous-population voisine
Remplacer *N* individus dans la sous-population
Fin

Figure IV.10 Pseudocode du modèle parallèle proposé

Une nouvelle conception hybride (AGC/AGD) hautement parallèle pour le problème d'ordonnancement des tâches a été proposée dans le but de trouver des solutions optimisées dans un délai raisonnable.

Deux niveaux architecturaux ont été illustrés sur la figure IV.11 :

- Le premier consiste en une hybridation plate où un certain nombre de sous-populations adjacentes suivent le même principe de migration. Les individus pour chaque île soumis aux contraintes min / max selon les objectifs et la disponibilité de l'infrastructure, et
- Une hybridation parallélépipédique où les sous-populations sont placées sous la forme d'un cube entièrement maillé, cette représentation comprend la première ou l'hybridation plate est déployée pour chaque surface, à condition que pour tous les individus d'îles différentes puissent avoir les quatre voisins. La description de l'implémentation des deux architectures sur les unités de traitement graphique (GPU) qui gèrent sans pénalité un nombre dé-normalisé en matériel sera détaillée au chapitre quatre.

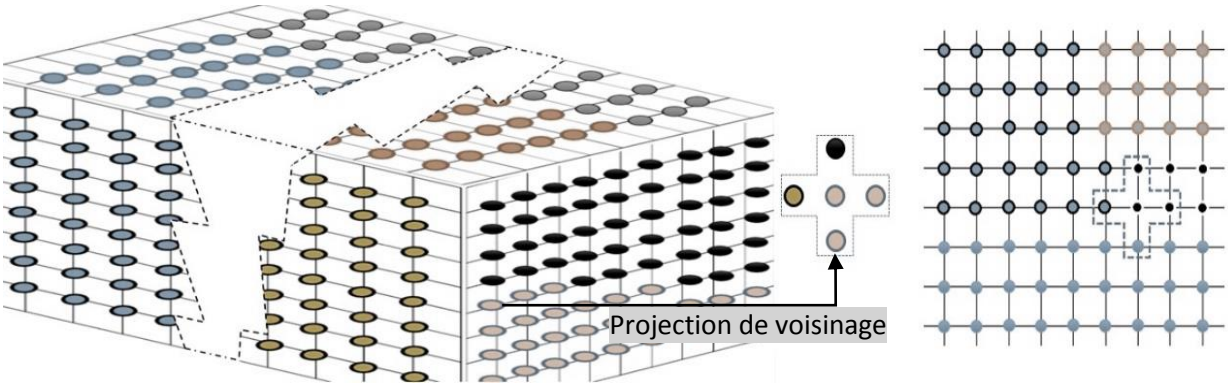


Figure IV.11 Modèles de conception.

IV.10 Conclusion

Les méta-heuristiques parallèles et principalement les algorithmes génétiques parallèles se sont révélés utiles dans la pratique pour résoudre un grand nombre d'applications. De nombreux problèmes réels peuvent nécessiter des jours ou des semaines de temps de calcul pour être résolus sur des machines série. C'est le scénario habituel lorsque l'on considère des problèmes difficiles tels que des problèmes combinatoires complexes avec de grands espaces de recherche, des problèmes multi-objectifs avec de nombreuses fonctions objectifs difficiles à évaluer ou des problèmes d'optimisation dynamique. Bien que la complexité intrinsèque temporelle d'un problème ne puisse pas être réduite en utilisant une quantité finie de ressources informatiques, les techniques de calcul parallèle permettent souvent de réduire les temps de résolution à des niveaux raisonnables. Il s'agit d'un avantage très important dans un contexte industriel ou commercial, où le temps de calculer une solution est déterminant pour la prise de décision et la compétitivité. En tenant compte de tous ces paramètres, on obtient une fonction exponentielle à chaque fois que l'on multiplie deux axes d'optimisation (cout, temps de réponse, optimalité), à cet effet nous verrons dans le dernier chapitre les avantages de l'utilisation des unités de traitement graphique ainsi que l'implémentation et les adaptations de l'algorithme

génétique parallèle sur les GPU, et nous finalisons avec une proposition d'une approche parallèle pour résoudre le problème Job-Shop déjà mentionné.

V. Chapitre V Optimisation de la gestion des ressources par le calcul à usage général sur les GPU

V.1 Introduction

Lors de l'étude d'un algorithme parallèle, il est important de prendre en compte sur quelle plateforme informatique il a été implémenté, car l'architecture matérielle impacte notamment le temps nécessaire pour effectuer les calculs, les communications, les synchronisations et le partage des données. Jusqu'à la dernière décennie, les propositions classiques de méta-heuristique parallèle étaient axées sur les superordinateurs traditionnels et les grappes de postes de travail. Actuellement, les nouvelles architectures de calcul parallèle émergentes telles que les processeurs multi-cœurs, les unités de traitement graphique (GPU) ou les environnements de grille, offrent de nouvelles opportunités pour développer des techniques de calcul parallèle afin d'améliorer la résolution des problèmes et de réduire les temps de calcul requis. Le but de cette partie est de fournir une introduction douce au monde du calcul sur les unités de traitement graphique, ou "GPGPU" comme on l'appelle, qui est un domaine actif nouveau et très respecté de la recherche graphique et des systèmes. La section résume également certains problèmes importants concernant les outils logiciels pour la mise en œuvre de méta-heuristiques parallèles.

V.2 Parallélisation partielle et loi d'Amdahl

Un moyen simple d'exploiter une machine parallèle avec un algorithme évolutif consiste à paralléliser l'étape d'évaluation. Tout d'abord, cela garantit que l'algorithme parallélisé est strictement identique à l'algorithme séquentiel. Deuxièmement, cette méthode est très simple à mettre en œuvre et est également très efficace, à condition que le temps d'évaluation soit long par rapport à toutes les autres étapes de la boucle évolutive, ce qui est heureusement assez souvent le cas. Mais Gene Amdahl sort une loi (loi d'Amdahl) qui stipule que si P est la proportion d'un algorithme qui peut être parallélisé et (1-P) est la proportion séquentielle du même algorithme, alors l'accélération maximale réalisable avec N cœurs est :

$$S(N) = 1 / ((1 - P) + (P / N))$$

V.1

Si N tend vers l'infini, l'accélération maximale est = 1 / (1 - P)

En d'autres termes, si un programme a 10% de code séquentiel et 90% de code parfaitement parallélisable, et si ce programme est exécuté sur une machine parallèle avec un nombre infini de cœurs, seule la partie séquentielle à 10% restera, ce qui signifie que le maximum d'accélération qui peut être obtenue en utilisant une machine parallèle n'est que de 10x. Afin d'obtenir un speedup 100x avec une machine parallèle avec un nombre infiniment grand de cœurs, il faut exécuter un algorithme dont la partie parallèle représente 99,9% de l'algorithme. Cela signifie que seuls les programmes hautement parallélisables peuvent atteindre une telle accélération.

V.3 Parallélisation complète pour une accélération maximale

Heureusement, les algorithmes évolutifs sont étrangement parallèles, ce qui signifie qu'il devrait être possible de les paralléliser complètement. En effet, en supposant que l'on considère

la boucle évolutive générique exécutant un AG sur une carte GPGPU à n cœurs avec une population de n individus situé dans la mémoire globale :

1. Afin d'initialiser la population, on peut demander à chacun des n noyaux de créer au hasard et indépendamment un individu. Comme la procédure d'initialisation sera identique pour tous les individus, il est probable que toutes les tâches seront compatibles avec une exécution SIMD presque parfaite sur les multiprocesseurs de la carte.
2. La prochaine étape consiste à évaluer tous les individus de la population. Une fois de plus, chacun des n cœurs peut évaluer de manière indépendante et principalement SIMD (les choses sont légèrement différentes pour la programmation génétique, mais l'évaluation de différents individus peut également être effectuée très efficacement en utilisant les architectures parallèles SPMD et SIMD).
3. Ensuite, chaque cœur peut vérifier si l'individu qu'il a évalué répond à un critère d'arrêt particulier ou si le nombre de générations prévu (ou la durée d'exécution maximale autorisée) a été atteint. Un ou plusieurs noyaux peuvent arrêter la boucle évolutive et, bien sûr, le meilleur de tous les individus de la population sera le résultat de l'algorithme évolutif. Cette dernière étape n'est pas parfaitement parallèle, mais elle se produit une fois dans tout le programme d'évolution artificielle, et vérifier quel est le meilleur individu n'est pas une tâche particulièrement longue.
4. Si aucun critère d'arrêt n'a été satisfait, on peut demander à chaque noyau de sélectionner n parents parmi les meilleurs et effectuer des opérateurs de variation sur les parents pour créer un offspring et répéter l'opération. La création des offsprings peut être effectuée indépendamment par chaque noyau sans nécessiter aucune communication entre les noyaux. La procédure de création des offsprings étant identique pour tous les cœurs, cette étape sera également majoritairement SIMD et donc très efficacement parallélisée sur une puce GPU.
5. Une fois que tous les noyaux ont créé leurs offsprings, il est nécessaire de les évaluer. Cela peut être fait en parallèle comme à l'étape 2.

V.4 Les types de parallélisme prédominants

Au cours des dernières décennies, de nombreux systèmes informatiques parallèles sont apparus sur le marché. Premièrement, ils ont été vendus en tant que supercalculateurs dédiés à la résolution de problèmes scientifiques spécifiques. Mais le parallélisme s'est répandu tout le long du marché grand public et de toutes sortes d'appareils portables. Diverses solutions parallèles ont progressivement évolué vers des systèmes parallèles modernes qui présentent au moins l'un des trois types de parallélisme dominants (Figure V.1) :

- Premièrement, les systèmes de mémoire partagée, c'est-à-dire les systèmes avec plusieurs unités de traitement attachées à une seule mémoire.
- Deuxièmement, les systèmes distribués, c'est-à-dire des systèmes constitués de nombreuses unités informatiques, chacune avec sa propre unité de traitement et sa mémoire physique, qui sont connectées à des réseaux d'interconnexion rapide.
- Troisièmement, les unités de processeur graphique utilisées comme coprocesseurs pour résoudre des problèmes à usage général intensifs en nombre.

V.5 L'impact de la communication

Nous avons vu que les différents modèle utilisent explicitement des réseaux d'interconnexion pour acheminer les demandes de mémoire vers les mémoires non locales. Des expériences récentes ont montré que les temps d'exécution de la plupart des applications parallèles du monde réel dépendent de plus en plus du temps de communication plutôt que du temps de calcul. Ainsi, à mesure que le nombre d'unités de traitement ou d'ordinateurs coopérants augmente, les performances des réseaux d'interconnexion deviennent plus importantes que les performances de l'unité de traitement. Plus précisément, le réseau d'interconnexion a un impact important sur l'efficacité et l'évolutivité d'un ordinateur parallèle sur la plupart des applications parallèles du monde réel. En d'autres termes, les performances élevées d'un réseau d'interconnexion peuvent finalement se traduire par des accélérations plus élevées, car un tel réseau d'interconnexion peut raccourcir le temps global d'exécution parallèle et augmenter le nombre d'unités de traitement qui peuvent être efficacement exploitées.

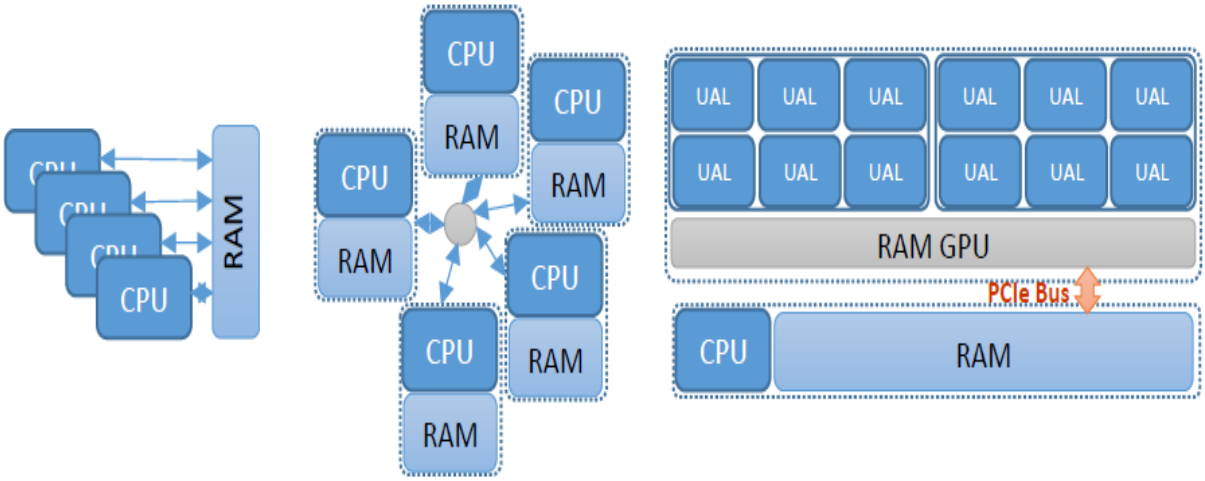


Figure V.1 Les différents types de système parallèle

V.6 Modèle de programmation

Le but est de fournir suffisamment de contexte sur la manière dont les GPU sont programmés pour le calcul non graphique. Les GPU modernes utilisent du matériel SIMD large pour exploiter le parallèle au niveau des données dans les applications.

Des contraintes ont été utilisées pour concevoir les GPU et un certain nombre de spécificités ont été utilisées pour simplifier l'architecture de ces puces qui ont fini par nous permettre de mettre en œuvre des cœurs de calcul plus génériques afin de maximiser la puissance de calcul.

Tout d'abord, il faut comprendre qu'une carte graphique est un ordinateur massivement parallèle à part entière, totalement indépendant de l'hôte. Cela signifie qu'une mémoire est partagée entre la carte graphique et le système hôte. Pour utiliser la carte, il faut préparer du code objet compilé pour la puce GPGPU (en utilisant un compilateur nvcc, par exemple, si l'on utilise une carte NVIDIA) ainsi que des données, puis transférer tout sur la carte

graphique et démarrer l'exécution à partir de l'hôte. Lorsque le calcul massivement parallèle est terminé, il faut alors re-transférer toutes les données traitées vers l'hôte.

En règle générale, des algorithmes identiques doivent être exécutés sur des pixels indépendants, donc plutôt que d'implémenter des cœurs indépendants dans le GPU comme c'est le cas dans les processeurs multi-cœurs, il est possible d'exploiter des données multiples à instruction unique (SIMD) où plusieurs cœurs exécutent la même instruction en même temps, mais sur des données différentes. Si tous les cœurs exécutent le même algorithme sur de nombreux pixels, tous les cœurs pourraient fonctionner en mode SIMD.

Au lieu d'exposer ce matériel SIMD directement au programmeur, les API de calcul GPU, telles que la plateforme CUDA et OpenCL, présentent un modèle de programmation de type MIMD qui permet au programmeur de lancer un large éventail de threads scalaires sur le GPU. Chacun de ces threads scalaires peut suivre son chemin d'exécution unique et accéder à des emplacements mémoire arbitraires. Les centres de données modernes regroupent un grand nombre de systèmes hétérogènes à grande échelle. Outre les processeurs à usage général, les nombreux cœurs et autres périphériques spécialisés prennent en charge l'exécution rapide de grandes applications distribuées. Les fournisseurs de cloud et les centres HPC à adopter des dispositifs hétérogènes adaptés aux applications d'IA. Dans ce qui suit, nous résumons les technologies largement adoptées qui sont à la base de ce processus de transformation du Datacenter.

V.6.1 Modèles de programmation basés sur des directives

La programmation directive est basée sur l'utilisation d'annotations dans le code source, également appelées pragmas. Les directives sont utilisées pour annoter le code source séquentiel existant sans qu'il soit nécessaire de modifier le code d'origine. Cette approche permet aux utilisateurs d'exécuter le programme séquentiellement en ignorant simplement les directives, ou d'utiliser un compilateur compatible qui est capable d'interpréter les directives et éventuellement de créer une version parallèle du code séquentiel d'origine.

Open Multi-Processing (OpenMP) est la norme dominante pour la programmation de systèmes multicœurs utilisant l'approche directive. Suite au succès d'OpenMP, le standard OpenACC (Open Accelerators) a été proposé lors de la conférence Super-Computing en 2011. Ce nouveau standard permet aux programmeurs d'implémenter des applications pour s'exécuter sur des architectures hétérogènes, telles que des processeurs multi-cœurs et des GPU, en utilisant l'approche directive. Suite au succès d'OpenACC pour la programmation hétérogène sur GPU, OpenMP a également inclus le support des appareils hétérogènes depuis le standard OpenMP 4.0. Cette version de la norme comprend un ensemble de nouvelles directives pour programmer les GPU ainsi que les processeurs multi-cœurs. En effet, OpenMP est complètement explicite sur la façon d'utiliser le GPU à partir du niveau de la directive du programme. Un accélérateur peut être n'importe quel périphérique, tel qu'un CPU ou un GPU. Ensuite, il spécifie comment les données doivent être vues par l'hôte (thread OpenMP principal sur le CPU). OpenMP peut devenir très verbeux, ne donnant aucun contrôle de l'exécution au programmeur. En outre, les programmeurs contrôlent la manière dont les données sont

partagées, allouées et transférées, ainsi que la manière dont l'exécution est effectuée et planifiée sur le GPU.

V.6.2 Architecture de la plateforme CUDA

Dans la programmation parallèle, l'unité d'exécution de base est le thread. Dans une CPU, les threads sont des sous-programmes d'un principal programme pour exécuter un ensemble d'instructions personnalisé pouvant inclure des accès mémoire à des ressources locales ou partagées. Si nécessaire, les threads peuvent communiquer entre eux à l'aide d'une ressource globale ou de la mémoire, mais une attention particulière est requise si les threads en cours d'exécution exécutent des opérations d'écriture dans la même adresse mémoire.

CUDA a introduit une plate-forme de calcul parallèle à usage général et un modèle de programmation capable de combiner des langages de programmation bien établis avec une architecture hautement parallèle qui est un GPU. La création d'un programme CUDA-C fonctionnel dans un GPU est un processus en trois étapes. Tout d'abord, l'environnement d'exécution doit être défini. Cet environnement consiste en un noyau où un développeur formalise la routine à exécuter dans le GPU et comment elle doit être exécutée. La définition du noyau à trois arguments associés, le nombre de blocs, le nombre de threads et la taille de la mémoire partagée. La façon dont un noyau est défini reflète la façon dont le problème est organisé spatialement, par exemple, la réduction de somme parallèle sur un tableau peut être représentée avec un noyau 1D et une multiplication entre deux matrices avec un noyau 2D. Une fois le noyau déclaré, la deuxième étape commence. Le programme est compilé via le pilote de compilateur de NVIDIA, NVCC, qui génère un ensemble de binaires qui incluent le code d'assemblage GPU, contenant le plan d'exécution pour un thread donné. Chaque thread se voit attribuer un identifiant unique à trois éléments (coordonnées x, y, z). Sur la base de plusieurs des indicateurs de compilation disponibles, NVCC peut effectuer certaines optimisations qui peuvent augmenter les performances d'un noyau.

V.7 Algorithmes génétiques et les processeurs graphiques

Les algorithmes évolutifs sont une solution puissante à divers problèmes d'optimisation. Des algorithmes évolutifs aident à trouver une solution appropriée au cas où les méthodes exactes seraient difficiles à incarner. L'une des variantes les plus populaires des algorithmes évolutifs est les algorithmes génétiques. Parfois, si un problème est complexe et que les individus sont lourds, il peut ne pas être possible de mettre en œuvre un algorithme génétique efficace, car les calculs prennent trop de temps et il n'est pas possible de stocker toutes les données nécessaires en mémoire. Afin de surmonter le deuxième problème, il est possible de les stocker sur disque et de les charger en mémoire une fois qu'ils sont nécessaires. Cela ne résout pas les problèmes de performances et rend l'ensemble du système encore plus lent. De plus, tous les individus ont été créés à l'aide des mêmes opérateurs génétiques et peuvent donc avoir trop de points communs, de sorte qu'un algorithme contournera un optima local. D'où l'idée d'utiliser un algorithme génétique parallèle qui utilise plusieurs algorithmes génétiques pour résoudre la même tâche et après avoir terminé, le meilleur individu de chaque algorithme est sélectionné, puis le meilleur d'entre eux est sélectionné, et c'est la solution pour un problème.

Dans cette approche, les populations sont isolées les unes des autres. Ces algorithmes génétiques ne dépendent pas les uns des autres, ils peuvent donc fonctionner en parallèle, en tirant parti d'un processeur multi-cœur. Chaque algorithme a son propre ensemble d'individus, de sorte que ces individus peuvent différer des individus d'un autre algorithme, car ils ont des antécédents de mutation / croisement différentes. Pour accélérer la convergence, il faut surmonter le problème de l'isolement avec une approche hybride qui repose principalement sur le modèle des îles et un mécanisme de migration entre sous-populations. Reste le problème de la surcharge de la RAM, l'algorithme adopté est orienté calcul, en prenant l'exemple du codage qui consomme plus d'UAL pour le décodage et moins d'espace pour stocker. Enfin, l'infrastructure GPU est la meilleure solution qui combine tous ces paramètres dont le but est d'aboutir à une solution très proche de l'optimum (ou optimal), sans consommer de ressources et dans un délai orienté reconfiguration.

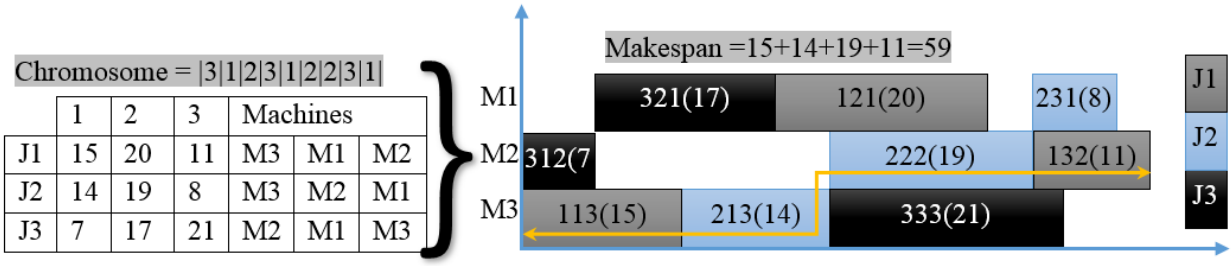
V.7.1 Représentation de la population

La première chose que nous devons faire est de représenter les données de population dans un format accessible par le GPU. L'idée générale est de stocker les populations dans la mémoire partagée et d'exploiter la puissance parallèle pour implémenter des opérateurs génétiques.

Dans notre représentation, le codage est donné par un chromosome pour chaque individu qui représente l'ordonnancement. L'objectif principal est de générer des programmations réalisables, minimiser les réparations d'individus, et d'appliquer facilement les opérateurs génétiques. Nous avons utilisé la représentation basée sur les opérations, ce qui permet de coder l'ordonnancement comme une séquence d'opérations.

Pour un problème donné de jobs et de machines, un chromosome est une permutation avec des répétitions de jobs. Les opérations sont représentées par le numéro de chaque job, et elles apparaissent plusieurs fois dans le chromosome. Chaque apparition d'un même numéro indique une opération dans la séquence de programmation pour le job donné. Ce type de représentation génère toujours des solutions réalisables. Par exemple, un chromosome |3|1|2|3|1|2|2|3|1| (3*3) est donné (Figure V.2), où 1, 2 et 3 correspondent à des jobs. Chaque tâche est répété autant de fois que le nombre de machines que le problème possède et ils représentent les opérations de chaque job.

Le chromosome de chaque individu est séquentiellement représenté en plusieurs segments, chaque segment contient *n* gènes forment une codification de taille *n* x *m*. Une autre carte de texture 1D est utilisée pour conserver les scores de fitness de chaque individu de la population. La représentation proposée permet le calcul efficace des opérateurs génétiques. Il présente plusieurs avantages : il conserve naturellement la topologie de la population décrite, il maintient l'empreinte mémoire de chaque individu dans la mémoire cache pour les utiliser efficacement, le regroupement des chromosomes utilise les vastes calculs SIMD effectués par le GPU, ou les opérations sont faites simultanément.



	1	2	3	Machines		
J1	15	20	11	M3	M1	M2
J2	14	19	8	M3	M2	M1
J3	7	17	21	M2	M1	M3

Figure V.2 Exemple décodage, représentation et calcul makespan

V.7.2 Évaluation de la fonction de fitness

Il est important de souligner que notre but est de résoudre les problèmes dont la fonction fitness peut être entièrement implémentée en GPU. Ce n'est que dans ce cas que nous pouvons éviter le goulot d'étranglement de la lecture des données de population du matériel graphique vers la mémoire système à chaque itération de GA. D'un autre côté, l'exécution de l'évaluation de la fonction fitness sur le GPU peut tirer parti des capacités de traitement parallèle du GPU.

Une fois la fonction responsable de l'évaluation de la fonction fitness exécuté, les valeurs sont stockées dans la texture dédiée (Figure V.2). Cette texture est ensuite réaffichée lors du passage suivant, et un autre programme est exécuté pour effectuer des opérateurs génétiques.

```
Paramètre :
x : individu
n, m : dimensions
pr[n][2] : programme
ord[n][m] : ordonnancement
job[n] : compteur jobs
makespan
Sortie : ordonnancement
Initialisation :
Job :=1
Makespan :=0
Debut
Pour i = 0 a i < n × m
        ord[Job[x[i]]][pr[x[i]][2]]=x[i]. Job[x[i]]. pr[x[i]][2]
        Si (pr[x[i]][2]<> pr[x[i-1]][2])
                Si (pr[x[i]][1]= pr[x[i-1]][1])
                makespan := makespan + pr[x[i]][1]
                        Sinon makespan := makespan - pr[x[i-1]][1] + pr[x[i]][1]
        Sinon makespan := makespan + pr[x[i]][1]
        Job[x[i]]++ ;
Fin
```

Figure V.3 Pseudocode décodage et calcul Makespan pour chaque thread

V.7.3 Opérateurs génétiques

Les opérateurs de sélection, de croisement et de mutation peuvent être facilement mappés à une solution parallèle. Le programme nécessite des textures de population de recherche, une texture de fitness et une texture pour les paramètres du système tels que les probabilités de mutation et les probabilités de croisement, etc. Le résultat est écrit dans une nouvelle texture de population. Dans notre implémentation, pour une population représentée par *x* textures de population, *x* passes doivent être effectuées à chaque génération de AG par sous-population. Dans chaque passage tous les gènes de chaque chromosome sont traités en parallèle. Cela est possible car l'opérateur de croisement et l'opérateur de mutation que nous avons utilisés peuvent être effectués dans le même block de threads. L'opérateur de croisement définit la procédure de génération d'un off-spring à partir de deux parents. Pour chaque individu, le meilleur individu de son quartier sera choisi comme l'un des parents, tandis que l'autre est lui-même. L'opération de mutation est la dernière étape de l'opération génétique. Le rôle de la mutation dans l'AG est de restaurer les solutions perdu ou inattendu dans une population pour empêcher la convergence prématurée de l'AG vers un résultat local, dans notre cas, le nombre de sous-populations est important, ce qui justifie une faible probabilité de mutation (0,001). La description de la phase de recombinaison dans la même sous-population est illustrée dans la figure V.7.

```
Paramètre :
x : individu
dim : Nbr de colonne de la matrice (sub-pop)
Sortie : mise à jour de la sous-population
Initialisation :
bs = x
Debut
Pour chaque x
Exécuter_en_Parallèle
        Evaluation (x)
        Synchronisation ()
        Recombinaison(x, x+1, x+dim)
        bs = meilleur_solution ()
        If (x mod dim == 0)
                x = bs
Fin
```

Figure V.4 Pseudocode recombinaison dans une sous-population

V.7.4 Synchronisation dans les processeurs graphiques

La synchronisation implique que nous allons en quelque sorte attendre que tous nos threads atteignent un certain point avant de continuer. Sur un GPU où vous faites constamment tourner de nombreux nouveaux threads, cela se traduit en fait par quelque chose comme "attendez que tous les threads d'une phase se terminent avant que les threads d'une deuxième phase commencent à s'exécuter". Le cas commun où nous devons le faire est celui où un envoi doit lire les résultats qui ont été écrits par un autre envoi. Par exemple, disons que nous exécutons *n* threads dans la phase A qui écrivent collectivement leurs résultats dans *n* éléments d'un tampon (Device Memory). Une fois la partie A terminée, nous voulons exécuter *n* threads de la phase B, qui lira ensuite ces *n* éléments dans le tampon de sortie d'origine et les utilisera pour calculer de nouveaux résultats écrits dans un autre tampon. Dans des algorithmes parallèles efficaces, les threads coopèrent et partagent des données pour effectuer des calculs collectifs. Pour partager des données, les threads doivent se synchroniser. La granularité du partage varie d'un algorithme à l'autre, dans notre cas la synchronisation des threads doit être flexible, pour cela nous avons utilisé les groupes coopératifs de la plateforme CUDA (Le modèle de programmation des groupes coopératifs décrit le modèle de synchronisation à l'intérieur des blocs de threads comme le montre la Figure V.5) qui visent à satisfaire ces besoins en étendant le modèle de programmation pour permettre aux noyaux d'organiser dynamiquement des groupes de threads. Dans le même bloc, tous les cellules sont des candidats possibles à la mise à jour et fonctionnent donc de manière synchrone ; c'est-à-dire qu'il y a une coordination entre eux dans le temps. Il est donc nécessaire de stocker le vecteur d'état courant, le comportement est maintenant déterministe ; étant donné n'importe quel état, une transition d'état se produit vers un état suivant bien défini, sans comportement probabiliste. Pour l'échange entre les blocs cela ne se fait pas de manière purement asynchrone, juste que les mises à jour n'attendent pas une synchronisation mais utilisent l'état actuel qui est représenté dans la mémoire Host même si les threads du bloc source ne sont pas encore synchronisés (voir figure V.8). L'avantage d'utiliser un modèle où la plupart des phases se font de manière synchrone est que le nombre de générations peut être prédéfini alors qu'un exemple de 100 threads asynchrones peut générer en moyenne 50 itérations dans la même génération et le principe de continuer les calculs avec plus de 50 répétitions semblent mauvais puisque le but n'est pas d'atteindre la convergence totale. Et d'un autre point de vue architectural les GPU s'orientent vers des calculs légers et massivement parallèles et les algorithmes génétiques appartiennent à la famille des algorithmes évolutifs donc les adaptations du modèle empêchent l'utilisation asynchrone qui grignotera les ressources.

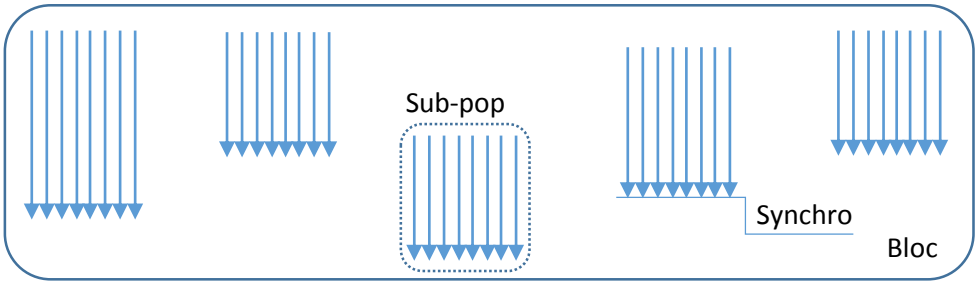


Figure V.5 Le modèle de programmation des groupes coopératifs

V.7.5 Implémentation d'algorithmes génétiques sur les plateformes de processeurs graphiques

Commençons par la forme la plus simple de l'algorithme cellulaire, c'est l'implémentation unidimensionnelle. La conception unidimensionnelle consiste en une ligne de rangées de cellules (chromosomes). Chaque cellule peut être active en fonction de sa probabilité d'être sélectionnée. La figure (V.5) montre comment choisir les deux voisins en fonction du voisinage de Von-Neumann, par exemple si l'on considère que nous avons une cellule de position *n* les deux voisins sélectionnés sont *n + 1* et *n + nl* ou *nl* est le nombre de cellules par ligne de sous-population. Les dernières cellules de chaque ligne nous projetons le 1D vers 2D qui est codé par *n mod nl == 0* ainsi que les cellules de l'intervalle [*dim-nl, dim* [sont réservées pour une migration vers une autre sous-population (Amrane, Debbat & Yahyaoui 2020) . Les cellules changent d'état et évoluent en fonction de la probabilité de sélection et après l'application des deux opérateurs génétiques.

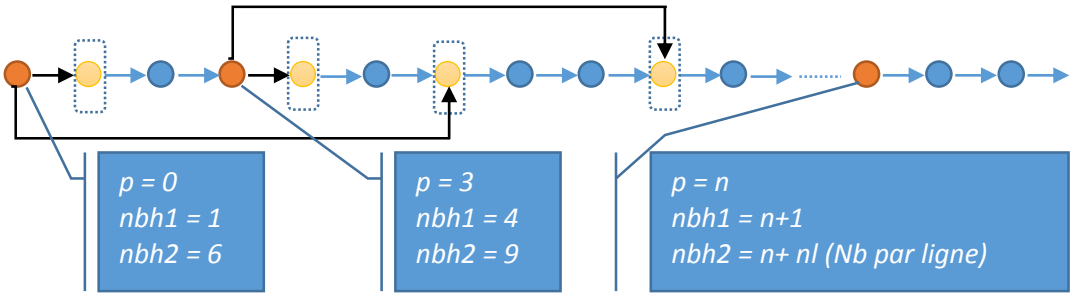


Figure V.6 Voisinage unidimensionnel

Les règles s'appliquent à chaque cellule en parallèle mais après synchronisation des threads pour chaque génération. Pour la figure (V.6), une projection bidimensionnelle pour mieux voir les étapes de l'algorithme. La première phase est l'application de l'opération de sélection où les cellules en jaune sont les chromosomes susceptible d'être choisi, la deuxième phase c'est le remplacement de la nouvelle cellule lors de l'application des opérateurs génétiques et les cellules en rouge sont les cellules après changement. La dernière phase c'est la phase de préparation à une migration vers d'autres sous-populations où les meilleures solutions évoluent vers la dernière lignée et la dernière colonne ou ces deux non pas concernés par les étapes précédentes, vu que la mise en œuvre est non pas circulaire et s'implémente sur une seule dimension et dans la même sous-population. L'approche unidimensionnelle est mieux adaptée dans les GPU pour une amélioration de temps de calcul car chaque ligne représente un groupe et les opérations s'appliquent de la même façon pour chaque position pour tous les groupes.

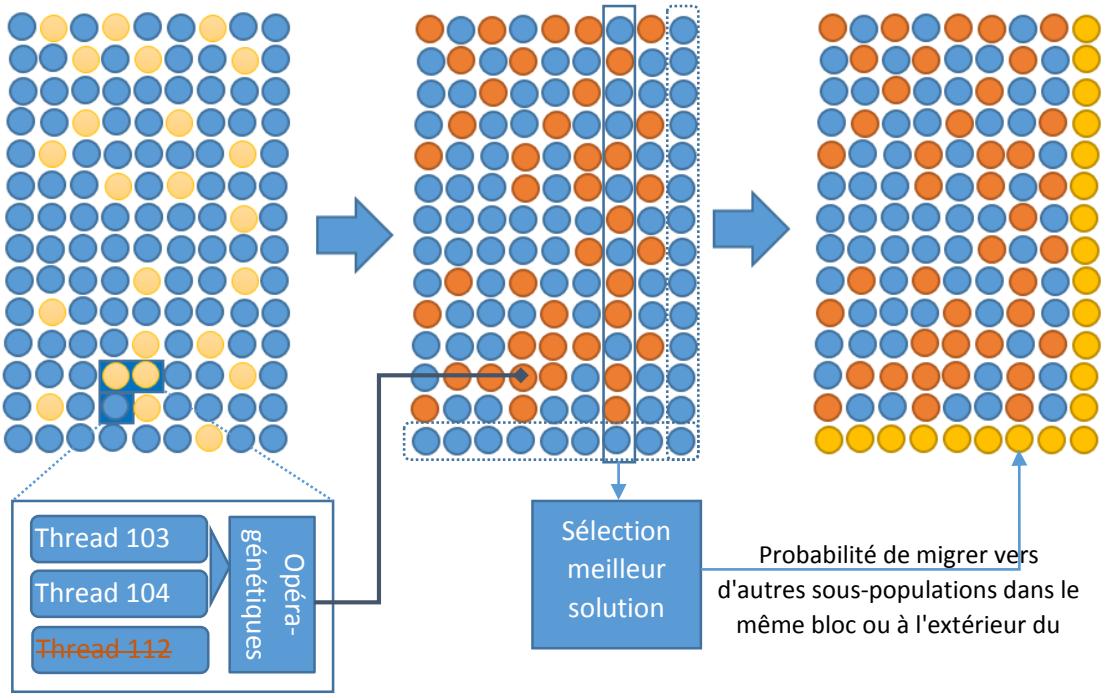


Figure V.7 Mécanisme de sélection, de recombinaison et de préparation à la migration.

V.8 Conception sur les processeurs graphiques

Deux niveaux de représentation sont proposés :

L1 : La première représentation est une configuration a un seul bloc qui doit être divisé en plusieurs sous-populations ; le nombre de threads est égal au nombre d'individus car chaque individu de la sous-population est mappé à un thread GPU (Figure V.8). En prenant par exemple le bloc x, qui est subdivisé en m sous-population, chaque sous-population contient n chromosomes (threads) T_{x0}. . T_{xn}; au hasard et séquentiellement on initialise tous les threads, puis tout ce qui suit se fait en parallèle, (le calcul du makespan et l'évaluation) .Comme notre solution est basée sur un algorithme parallèle, les phases de synchronisation sont obligatoires pour assurer la migration entre sous-populations. Si la solution est atteinte, nous migrons vers l'hôte, sinon les meilleures solutions sont choisies parmi les voisins et la migration vers d'autres sous-populations est effectuée, la phase d'initialisation est remplacée par la phase de migration. Même dans la même sous-population, des opérateurs génétiques sont appliqués. La procédure mentionnée restera en boucle jusqu'à ce que les conditions d'arrêt soient atteintes.

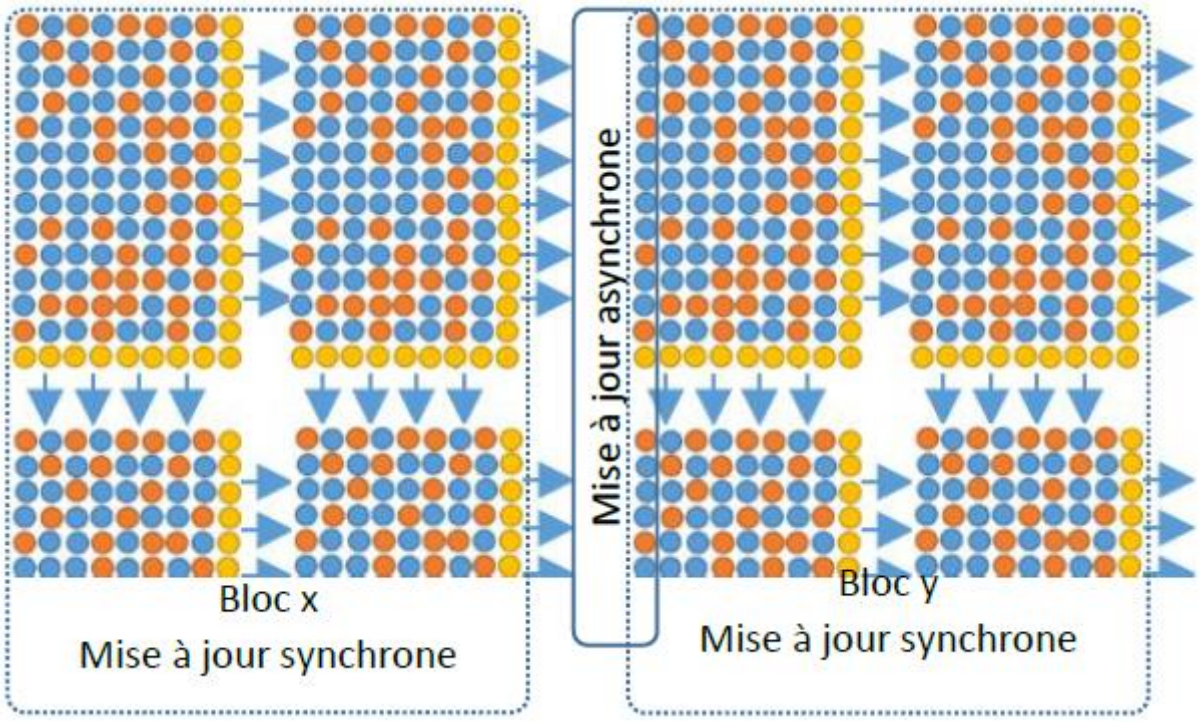


Figure V.8 Mise à jour entre/inter blocs

L2 : La représentation est programmée en plusieurs blocs, et chaque îlot est mappé à un bloc de threads GPU, également le nombre de threads est égal au nombre d'individus. (Figure V.9).Le deuxième niveau est une projection du premier niveau avec deux différences majeures, la première est la synchronisation autonome pour chaque sous-population, et la seconde est la migration effectuée indépendamment. Pour assurer une migration cohérente, une hauteur prédéfinie de sous-populations doit être respectée pour toutes les îles (Amrane, Debbat & Yahyaoui 2020).

L'algorithme proposé exploite les points forts du calcul graphique et gère très bien les points faibles. Compte tenu de la taille limitée du cache, une codification simplifiée a été approuvée pour minimiser la consommation de ressources et exploiter la puissance de calcul maximale (UAL), ainsi que le codage réel "JxM" qui économise du temps de calcul dans la phase de rétro-ingénierie. Pour une latence tolérable, un grand nombre de chromosomes (Threads) est déployé et les instructions plus lourdes comme les évaluations et les opérations génétiques sont effectuées en parallèle.

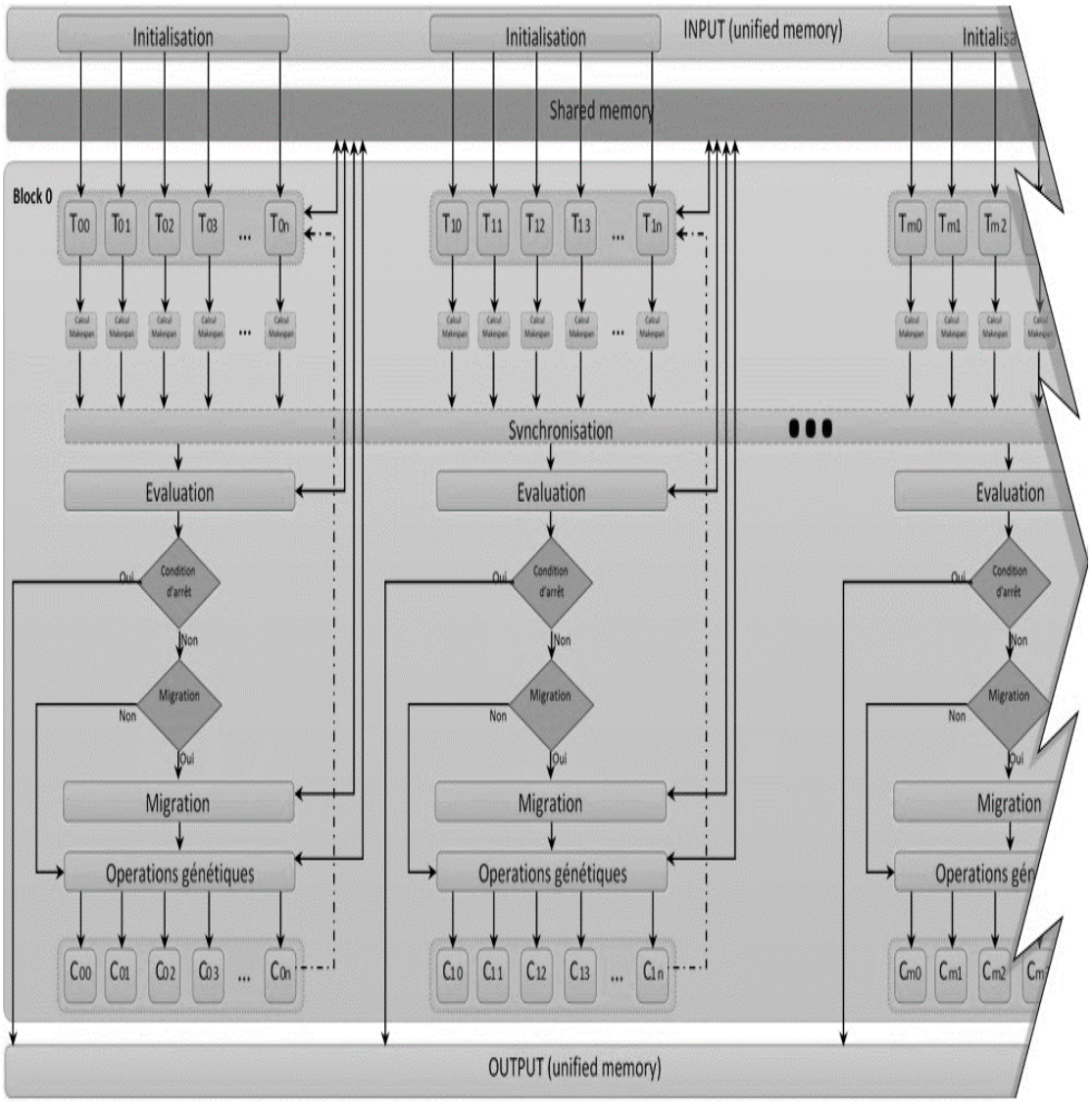


Figure V.9 Conception de l'algorithme inter-bloc

Une autonomie en termes de ressources est accordée au thread afin d'éviter un accès simultané à une ressource critique provoquant une sérialisation. Pour la phase de migration, la mémoire partagée utilisée pour interconnecter le même bloc, idéale pour les problèmes de taille moyenne, nous avons besoin de la mémoire globale lors du transfert d'informations entre l'hôte et le périphérique pour les problèmes a grand échelle.

L'architecture est subdivisée en plusieurs phases selon la conception de l'algorithme ou selon la nature et les spécificités de l'infrastructure parallèle basée sur GPU où la grande majorité est orientée vers une implémentation parallèle, autrement dit, il n'y a pas de dépendance entre les chromosomes d'une population pour le processus d'évaluation, de la fonction de fitness et les opérations génétiques (Amrane, Debbat & Yahyaoui 2020). Ainsi, toute la population peut être exploitée en parallèle en une génération.

La première population de solutions est initialisée aléatoirement et au niveau de l'hôte, c'est la seule phase jusqu'à ce que les itérations soient terminées qui a été déployée séquentiellement. Le processus d'évaluation proposé repose sur une méthode entièrement

parallèle où tous les threads exécutent le même code et le préfixe «__shared__» (CUDA macro), qui est utilisée pour déclarer une variable, un vecteur ou une matrice partagée par tous les threads de chaque bloc pour préparer les entrées des prochaines étapes de voisinage et de migration. Le processus peut accéder au chromosome en tant que tableau 1D et réécrire les résultats dans la mémoire du GPU pour éviter les allers-retours entre l'hôte et le périphérique. La probabilité de sélection pour chacun des chromosomes est également calculée, pour les résultats de la fonction de fitness, un tri rapide basé sur le GPU fourni par la bibliothèque CUDPP est appliqué.

L'approche utilisée pour le croisement utilise plusieurs threads par chromosome, tous

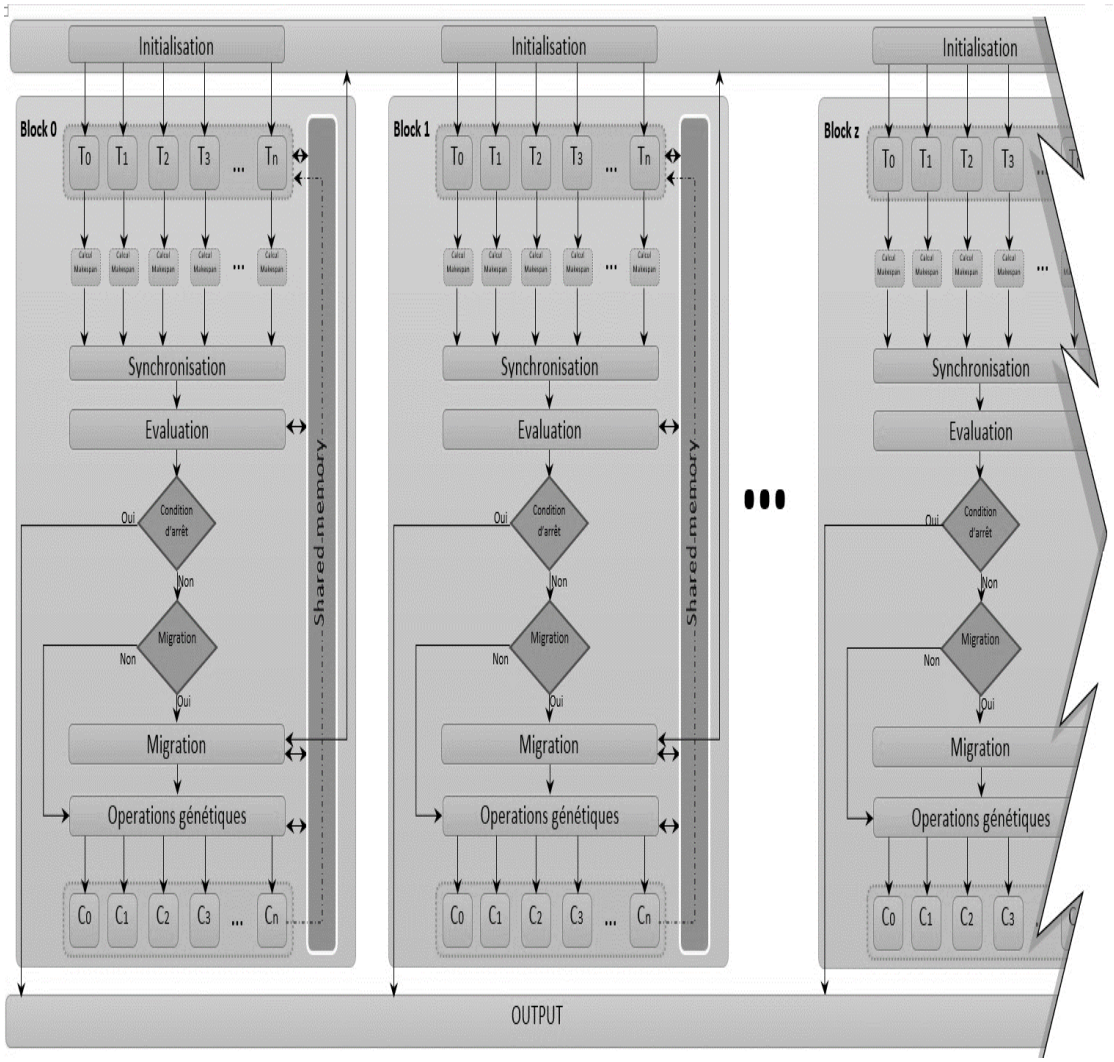


Figure V.10 Conception de l'algorithme entre blocs

les threads doivent connaître les valeurs des points de croisement. Sachant que la phase de synchronisation n'est plus un obstacle, chaque cœur génère lui-même ces valeurs de manière indépendante. En outre des avantages mentionnés, la conception autonome de l'algorithme permet d'utiliser le nombre maximum de threads par bloc et le nombre maximum de blocs. Les limites sont dues à l'efficacité de l'équipement, c'est-à-dire que l'on peut facilement améliorer les résultats en augmentant les performances de l'infrastructure (complexité O (log n)) et cela se traduit par une autonomie de chaque chromosome (thread) à faire toutes les opérations

associées et l'emplacement dans la grille (ID) détermine le lien entre les différentes cellules (voisinage, migration). Le modèle de thread SIMT nous permet de donner une implémentation étroite et efficace de l'algorithme pour utiliser la grande puissance parallèle du GPU. L'accélération de la communication inter-threads avec l'utilisation de la mémoire GPU réduit la complexité ainsi que le temps de rafraîchissement H / D.

V.9 Complexité

En supposant que les ressources informatiques sont infinies (le besoin du nombre de threads est toujours satisfaisant), les complexités des implémentations proposées sont de l'ordre de "O (log n)" dans la plupart des phases de l'algorithme à l'exception des implémentations du CPU les complexités sont de l'ordre de "O (n log n)". On peut voir que la complexité est considérablement réduite par rapport à l'implémentation du CPU sauf pour l'algorithme d'initialisation qui a le plus haut ordre de complexité, mais cet algorithme n'est pas critique dans le temps car il n'est exécuté qu'une seule fois.

V.10 Validation (Évaluation expérimentale)

V.10.1 Plate-forme

L'algorithme PGA proposé pour résoudre JSSP est déployé sous une plateforme Linux Ubuntu 16.04 comme système d'exploitation, Nsight Eclipse Edition à l'aide de de la plateforme CUDA 9.0 comme IDE et C ++ (boîte à outils GPU) comme langage de programmation. L'infrastructure utilisée est une station de calcul dotée d'un CPU Intel i7-4510U et 16 GB de RAM avec une GPU NVIDIA GM108 de capacité de calcul 5.0 Architecture Maxwell. Nous avons analysé l'accélération de la solution proposée aux algorithmes séquentiels et parallèles pour différents instances. Après avoir évalué le résultat, l'algorithme proposé PGA-GPU donne de meilleures performances pour les problèmes à grande échelle, la convergence des deux algorithmes montrera également la stabilité du PGA-GPU même après une augmentation de la taille de la population.

V.10.2 Résultats

Généralement les conceptions trouvées dans la littérature sont liés à la mémoire, ou principalement liés au calcul, dans notre cas (Amrane, Debbat & Yahyaoui 2020) un équilibre entre calcul et espace est très bien géré, on peut voir sur la figure V.10 qu'au début le temps d'exécution est en faveur de l'algorithme séquentiel, mais quand on augmente le nombre d'individus, l'efficacité de l'algorithme parallèle devient importante, et montre l'exponentialité de l'algorithme. Une accélération de l'algorithme en cas d'augmentation du nombre d'individus jusqu'au nombre maximum de cœurs sur GPU, avec saturation ultérieure. En raison de la faible dépendance aux informations, la taille du bloc de threads n'a pas d'effet critique. Lors de la réalisation de cet ensemble d'expériences, la charge sur le GPU était de 15 à 85% (en fonction du nombre de threads).Sur la figure V.11, la stabilité de l'algorithme proposé est claire même en augmentant le nombre de générations ainsi que la taille de la population, ce qui donne une

convergence linéaire. Notez que l'algorithme n'a jamais atteint les limites du périphérique, ce qui justifie les résultats obtenus.

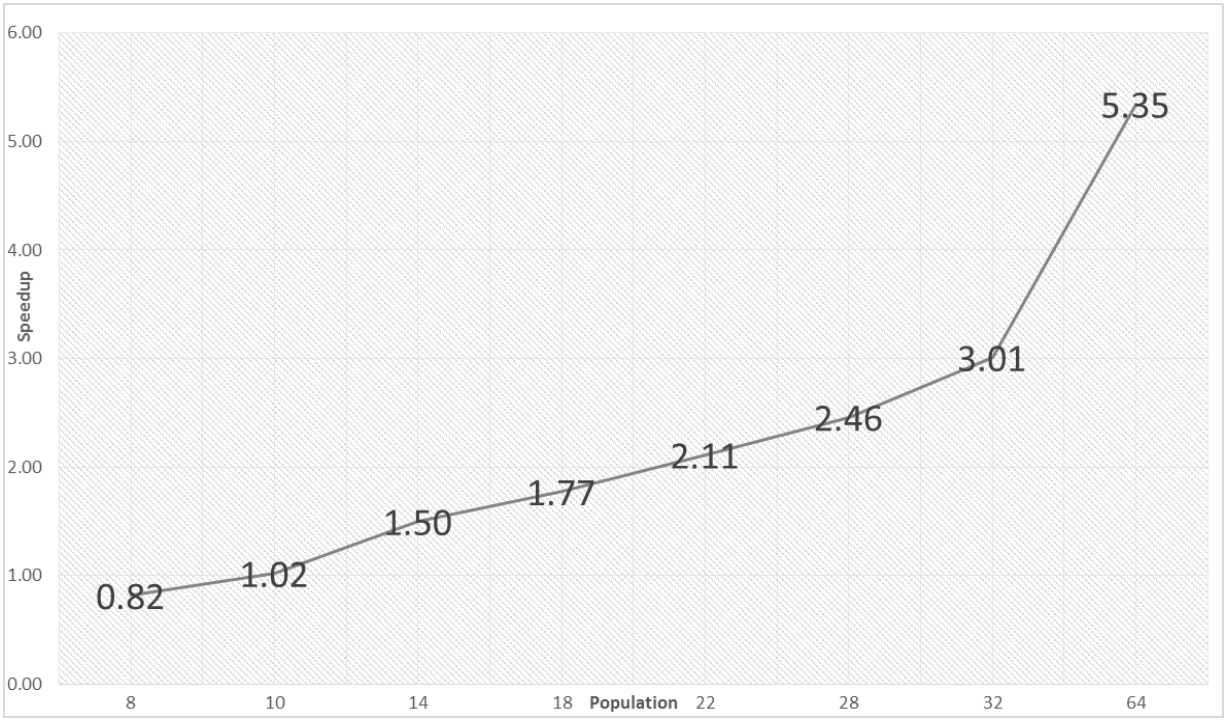


Figure V.11 GPU Speedup

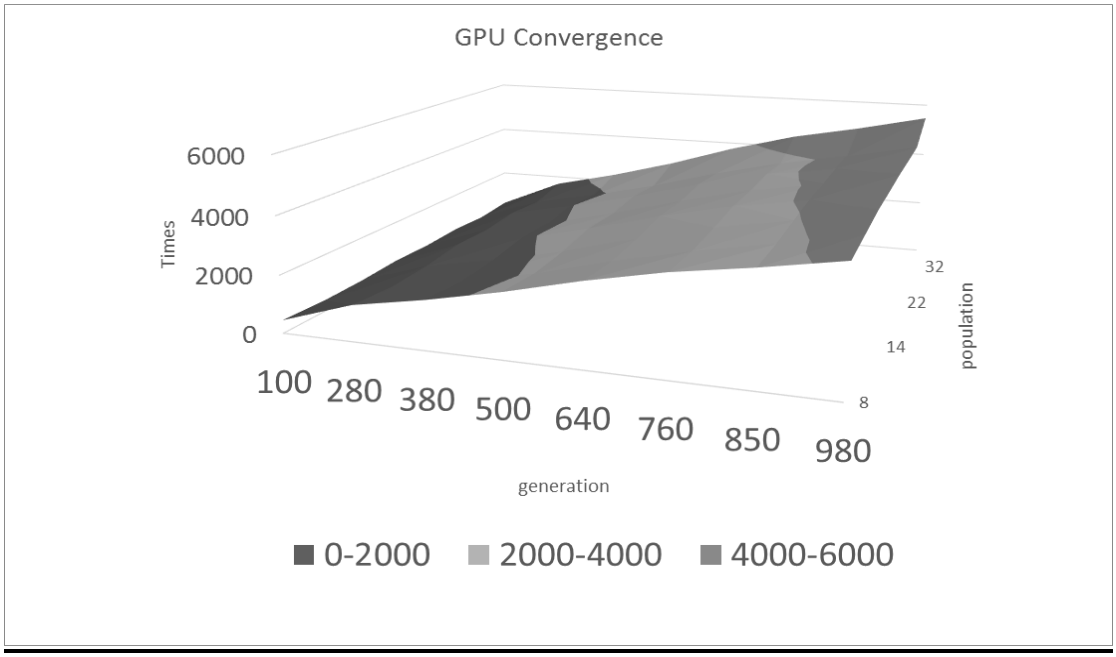


Figure V.12 GPU Convergence (Pop size)

Dans le tableau V.1, LB PGA-GPU signifie la borne inférieure de la solution proposée, et le LB PBA & LB PSO est respectivement la borne inférieure de l'algorithme de chauve-souris Parallel (Dao, Pan, Nguyen, & Pan , 2015) et l'algorithme hybride basé sur PSO et AIS pour optimiser le makespan dans les problèmes d'ordonnancement Job-Shop. Deux classes sont choisies dans la littérature FT 06.10.20 et LA01-39 (MRP II, 1998) pour évaluer l'efficacité de la solution proposée. Les résultats expérimentaux montrent que la méthode proposée nous conduit à obtenir de meilleurs résultats et surtout sur les cas où la valeur optimale n'est pas atteinte.

Instances	**JxM**	**LB PGA-GPU**	**Average**	**LB PBA**	**LB PSO**
ft06	6x6	55	55	55	55
ft10	10x10	930	930	930	930
ft20	20x5	1165	1165	1165	1165
la01	10x5	666	668.64	666	666
la02	10x5	655	657.69	655	655
la03	10x5	597	598.76	597	597
la04	10x5	590	593.58	590	590
la05	10x5	593	595.75	593	593
la06	15x5	926	927.5	926	926
la07	15x5	890	890	890	890
la08	15x5	863	863	863	863
la09	15x5	951	953.32	951	951
la10	15x5	958	959.65	958	958
la11	20x5	1222	1223.05	1222	1222
la12	20x5	1039	1039	1039	1039
la13	20x5	1150	1151.41	1150	1150
la14	20x5	1292	1295.59	1292	1292
la15	20x5	1207	1209.07	1207	1207
la16	10x10	945	945	945	945
la17	10x10	784	787.37	784	784
la18	10x10	848	848	848	848
la19	10x10	842	845.74	842	842
la20	10x10	902	905.18	902	902
la21	15x10	1046	1046	1046	1046

la22	15x10	927	930.35	933	932
la23	15x10	1032	1034.64	1032	1032
la24	15x10	935	936.55	941	950
la25	15x10	977	977	977	979
la26	20x10	1218	1220.64	1218	1218
la27	20x10	1235	1238.61	1247	1256
la28	20x10	1216	1219.53	1216	1227
la29	20x10	1152	1154.37	1179	1184
la30	20x10	1355	1356.67	1355	1355
la31	30x10	1784	1785.41	1784	1784
la32	30x10	1850	1852.29	1850	1850
la33	30x10	1719	1721.15	1719	1719
la34	30x10	1721	1724.62	1724	1723
la35	30x10	1888	1890.09	1889	1888
la36	15x15	1268	1270.69	1279	1281
la37	15x15	1397	1400.73	1411	1415
la38	15x15	1196	1198.13	1208	1213
la39	15x15	1233	1233	1236	1246

Tableau V.1 Résultats numériques trouvés et comparés à d'autres approches

Les résultats des instances SWV (Storer, Wu, & Vaccari, 1992) et YN (Fogel, 1993) sont présentés dans le tableau V.2. Les algorithmes GESTS (Nasiri & Kianfar, 2012) et TSSA (Zhang, Guan et Rao, 2007) fournissent les 10 premiers résultats SWV01-10 également le YN1-4. La colonne " JxM " correspond au nombre de Jobs x Machines, LB est le résultat obtenu par l'algorithme, Average est la moyenne des résultats de dix (10) lancements, CPU Av est le temps d'exécution moyen du CPU, la moyenne du temps d'exécution du GPU est représenté par GPU Av, Gen Min est le nombre minimum de génération pour obtenir le résultat. Pour avoir une meilleure solution pour un temps d'exécution minimal, le nombre maximum de génération proposé est 980, nous pouvons voir que la moyenne est d'environ 961 et toutes les instances ont réalisé les meilleurs résultats dans un temps raisonnable. La solution proposée (Amrane, Debbat & Yahyaoui ISIA'2018) présente de meilleures performances sur les deux axes, que ce soit en termes de temps d'exécution ou de makespan. Le temps d'exécution du GPU montre une diminution moyenne de 12x de l'algorithme parallèle proposé par rapport aux GUESTS.

		PGA-GPU (Approches adoptées)					GESTS			TSSA		
Instances	JxM	LB	Average	CPU Av (S)	GPU Av (S)	Gen Min	LB	Average	CPU Av (S)	LB	Average	CPU Av (S)
SWV01	20x10	**1407**	1408.57	42.84	5.23	934	1412	1417.5	53.2	1412	1423.7	142.1
SWV02	20x10	1475	1475	43.54	5.13	971	1475	1477.2	45.1	1475	1480.3	119.7
SWV03	20x10	1398	1398	27.34	3.18	969	1398	1398.4	40.1	1398	1417.5	139.1
SWV04	20x10	**1464**	1466.13	30.7	3.97	954	1471	1476.3	50.3	1470	1483.7	143.9
SWV05	20x10	1426	1426	25.52	3.45	980	1426	1436.5	48.2	1425	1443.8	146.7
SWV06	20x15	**1630**	1631.27	70.68	8.23	937	1677	1678.4	81.7	1679	1700.1	192.5
SWV07	20x15	**1513**	1514.56	53.34	6.1	958	1595	1622.6	74.9	1603	1631.3	190.2
SWV08	20x15	**1671**	1673.54	53.78	6.08	947	1766	1785.7	78.5	1756	1786.9	190
SWV09	20x15	1660	1660	58.22	7.16	969	1660	1681.8	72	1661	1689.2	193.8
SWV10	20x15	**1663**	1666.11	43.84	5.45	942	1760	1775.5	79.6	1754	1783.7	184.6
YN1	20x20	**854**	855.66	26.44	2.93	977	884	885.3	40.8	884	891.3	106.3
YN2	20x20	**870**	870	24	2.35	973	905	906.6	32.2	907	911.2	110.4
YN3	20x20	892	892	29.04	3.2	979	892	893.3	43.5	892	895.5	110.8
YN4	20x20	**929**	930.77	19.74	2.04	973	969	970.3	38.2	969	972.6	108.7

Tableau V.2 Résultats numériques trouvés et comparés à d'autres approches

Dans le tableau V.1, la meilleure solution entre les méthodes PGA-GPU (Proposé), PBA (Dao, Pan, Nguyen, & Pan, 2015) et PSO (Ge, Sun, Liang, & Qian, 2008) est indiquée en gras, la méthode PGA-GPU (Amrane, Debbat & Yahyaoui CITIM'2018) surpasse la méthode PBA dans 8 instances et surpasse la méthode PSO dans 11 instances, lorsque la complexité du problème augmente, l'amélioration devient particulièrement évidente. Dans le tableau V.2, pour la même classe d'instances, un temps de calcul plus petit est nécessaire pour le PGA-GPU ainsi que la moyenne de dix lancements. En exécution séquentielle, le temps CPU augmente avec la taille et la complexité du problème, mais dans la méthode proposée (PGA-GPU), le temps d'exécution parallèle reste stable, même en cas des problèmes à grande échelle comparant à la fois les algorithmes GESTS et TSSA. Les différences des moyennes entre PGA-GPU et GESTS (1,04%, 1,06%, 1,07%) montrent les performances de l'algorithme proposé. La figure V.12 illustre la convergence des deux algorithmes. L'algorithme parallèle est plus efficace car il trouve les meilleures solutions avec moins d'itérations. Nous pouvons voir que le long du graphique, la ligne de l'algorithme parallèle est toujours en dessous de la convergence PBA et commencera à 1,08% de moins, et la stabilité de l'algorithme commence aux cent quarantièmes itérations. L'étude des résultats obtenus avec l'algorithme proposé et sa comparaison avec les méthodes précédentes montre que la méthode proposée est meilleure et en particulier pour les problèmes à grande échelle. La raison principale de ce résultat est la légèreté du programme ainsi que l'orientation vers un parallèle complet.

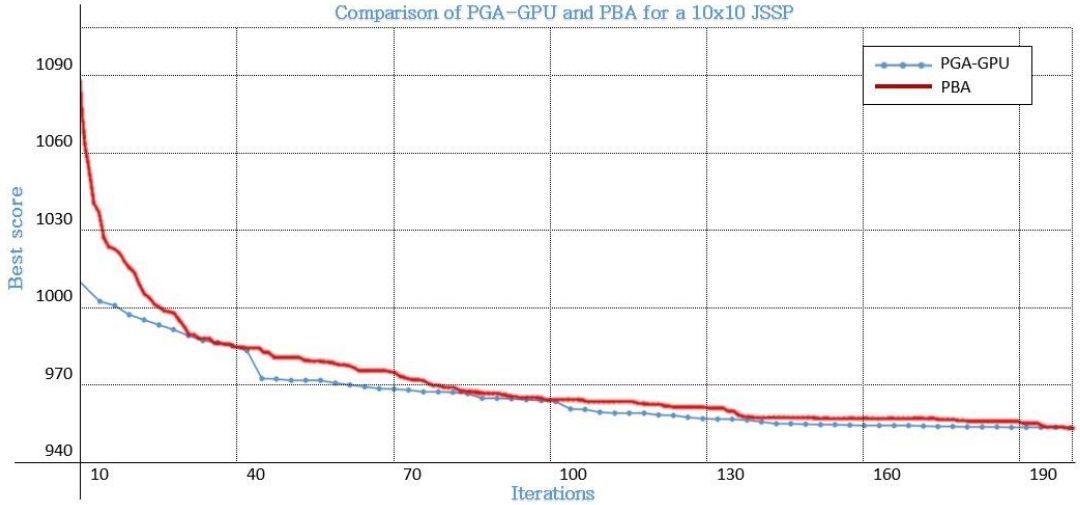


Figure V.13 Convergence PGA-GPU par rapport à PBA

V.11 Conclusion

Ce travail a proposé un algorithme génétique hybride parallèle entre une approche complexe et une approche fine pour résoudre le JSSP basé sur une architecture GPU utilisant la plateforme CUDA, les résultats attestent de l'efficacité de la méta-heuristique parallèle pour résoudre les problèmes NP-Hard. La mise en œuvre d'opérateurs génétiques au niveau Device, la fiabilité et la légèreté de l'algorithme, ainsi que la représentation génétique appropriée pour le traitement GPU, ont permis à l'algorithme d'atteindre des accélérations presque 18x pour des problèmes à grande échelle tout en consommant moins de 80% de la Mémoire partagée. Pour éviter les temps morts lors de la phase de croisement les offsprings donnent toujours une solution réalisable. La meilleure valeur a été atteinte avant que les itérations ne soient épuisées, ce qui prouve la convergence de l'algorithme. Le PGA proposé peut s'adapter à des extensions d'infrastructure et la solution peut être adoptée pour une recherche future afin de résoudre un autre type de problème comme le Job-Shop flexible et l'hybridation avec d'autres méta-heuristiques parallèles peut améliorer la convergence de l'algorithme.

Conclusion générale

Les travaux de cette thèse s'inscrivent dans le domaine de l'optimisation des systèmes de fabrication, principalement des méthodes permettant de minimiser le temps total écoulé pour les Jobs exécutés passant par des machines différentes. La recherche se concentre sur les problèmes de type Job-Shop compte tenu de la large couverture qui se caractérise par leur projection dans le monde réel ainsi que de sa complexité qui nécessite des méthodes très avancées comme les méta-heuristiques pour surmonter ces contraintes

La recherche expérimentale avec des méta-heuristiques parallèles et hybrides est le principal axe étudié dans cette thèse, nous avons examiné les différentes taxonomies de ces techniques, couvrant à la fois les aspects algorithmiques et informatiques pour répondre au besoin de la résolution du problème posé. Les principaux problèmes d'une conception expérimentale sont mis en évidence. Nous nous sommes concentrés sur des mesures de performance parallèles qui permettent de comparer des approches parallèles avec d'autres techniques de la littérature. Par ailleurs, nous avons montré l'importance de l'analyse statistique pour étayer nos conclusions.

Les travaux de cette thèse peuvent être classés en deux axes principaux, le premier axe c'est le passage de l' algorithme génétique (AG) séquentiel à l'application des algorithmes génétique hybride et parallèles sur le problème d'ordonnancement Job-Shop qui a pris beaucoup de recherches sur la partie conception en tenant compte de l'infrastructure utilisée pour surmonter la lenteur des méta-heuristiques basées sur la population. Le deuxième axe c'est l'architecture hétérogène CPU-GPU qu'elle est devenue la plate-forme informatique grand public qui accélère les applications, ce qui a conduit à un changement de paradigme fondamental dans la programmation parallèle. La différence entre les AG parallèles sur CPU et les AG parallèles sur GPU est clairement distinguée. Conceptuellement, un AG parallèle sur CPU exécute plusieurs instances séquentielles sur plusieurs processeurs, tandis qu'un AG parallèle sur GPU organise des threads massifs de centaines de milliers pour fonctionner véritablement en parallèle pour exercer la puissance du calcul GPU. Notre approche adoptée a permis de développer un système de fabrication plus intelligent avec une flexibilité et une transparence accrue. La plupart des tâches d'optimisation dans les différentes phases sont traités de manière parallèle vue qu'elles imposent plusieurs contraintes qui ne permettent généralement pas l'utilisation de méthodes exactes. La complexité de ce problème et les ressources limitées disponibles pour les résoudre (temps, mémoire) ont fait le développement d'un mécanisme de codification qui fournissent des solutions réalisables optimales ou sous-optimales dans un délai raisonnable. L'approche proposée permet non seulement pour réduire le temps de résolution, mais également pour améliorer la qualité des solutions. Contrairement aux méthodes exactes, où l'efficacité du temps est une mesure principale du succès. Le modèle parallèle utilisé ainsi que les approches de conception et les stratégies de mise en œuvre des AG sont examinés du point de vue de l'architecture de calcul GPU. La conception de l'AG parallèle sur GPU dépend à la fois du problème et de l'architecture, une implémentation efficace, généralisation du parallélisme et l'optimisation de l'accès à la mémoire. La plate-forme CUDA utiliser permet d'avoir plus de contrôle sur les threads massivement GPU afin d'exploiter pleinement la puissance de calcul des GPU. L'hybridation du modèle permet d'aborder des

applications de calcul à très grande échelle, maintenir la diversité des solutions entre les îles et renforce la capacité de recherche locale sur la recherche globale et stochastique au sein de chaque île.

Le travail a montré la puissance parallélisme pour réduire le temps de calcul des méta-heuristiques et / ou améliorer la qualité des solutions fournies. Les résultats rapportés démontrent que l'efficacité de ces modèles dépend à la fois du type de méta-heuristique en question et des caractéristiques du problème traité. Ainsi que l'échange d'informations asynchrone garantit un gain de temps de calcul mais souvent moins efficace que l'échange synchrone avec une efficacité plus grande s'il y a beaucoup de voisins candidats à évaluer. L'application des méta- heuristiques parallèles sur le problème générale de type Job-Shop montre la flexibilité de ces techniques par rapport à d'autres solutions d'optimisation classiques, ce qui en fait d'excellents candidats pour s'attaquer aux tâches d'optimisation les plus difficiles.

Comme perspectives nous envisagerons :

- Vue la portabilité de l'algorithme, nous pourrons l'utiliser dans les travaux futurs comme noyau pour des systèmes plus robustes et flexibles.
- Le déploiement dans d'autres architectures GPU plus complexes avec des versions de capacités de calculs plus avancées.
- Amélioration du modèle de recherche pour qu'il soit plus robuste afin de traiter plus de cas réels avec plus de flexibilité en termes d'utilisation de différentes infrastructures et/ou différentes architectures sous d'autres contraintes.
- Développement de nouveaux modèles parallèles de recherche éventuellement basés sur des architectures hétérogènes pour garantir des solutions meilleures pour les problèmes multi-objectifs.

Références bibliographiques

(Abdelhafez & al. 2020) Abdelhafez, A., Luque, G., & Alba, E. (2020). Parallel execution combinatorics with metaheuristics: Comparative study. Swarm&Evolutionary Computation, 55, 100692. doi:10.1016/j.swevo.2020.100692

(Adams, Balas & Zawack 1987) Adams, J., Balas, E., & Zawack, D. (1987). The shifting bottleneck procedure for job shop scheduling, Internatioanl Journal of Flexible Maunfacturing Systems, 34(3), 391–401.

(Adams, Balas&Zawack 1988) Adams, J., Balas, E.&Zawack, D., (1988). procedure for The shifting bottleneck job shop scheduling, Management Science, 34(3), 391–401.

(Alba & Troya 2000) Alba, E., & Troya, J. M. (2000, September). Cellular evolutionary algorithms: Evaluating the influence of ratio. In International Conference on Parallel Problem Solving from Nature (pp. 29-38). Springer, Berlin, Heidelberg.

(Alba 2002) Alba, E. (2002). Parallel evolutionary algorithms can achieve super-linear performance. Information Processing Letters, 82(1), 7-13. doi:10.1016/s0020-0190(01)00281-2

(Alba&Dorronsoro 2008) Alba, E., Dorronsoro, B., 2008. Cellular Genetic Algorithms, Operations Research/Computer Science Interfaces, Vol. 42. Springer-Verlag, Heidelberg.

(Amrane, Debbat & Yahyaoui 2020) AMRANE, A., Debbat, F., YAHYAOUI, K. GPU based Hybrid Cellular Genetic Algorithm for Job-Shop Scheduling Problem. International Journal of Applied Metaheuristic Computing (IJAMC), 12(3). (Indexed In: SCOPUS, Web of Science Emerging Sources Citation Index (ESCI), INSPEC)

(Amrane, Debbat & Yahyaoui CITIM'2018) AMRANE, A., Debbat, F., YAHYAOUI, K. Cellular Genetic Algorithm for solving Job-Shop Scheduling Problem Using Graphic Processing Unit.In Proceedings of the third International Conference on Multimedia Information Processing (CITIM'2018). October 09-10, 2018, Mascara, Algeria

(Amrane, Debbat & Yahyaoui ISIA'2018) AMRANE, A., Debbat, F., YAHYAOUI, K. An efficient Fine-Grained Parallel Genetic Algorithm for solving Job-Shop Scheduling Problem Using Graphic Processing Unit. In Proceedings of the third International Symposium on Informatics and its Applications (ISIA'2018). November 6-7, 2018, M'sila, Algeria

(Aiex, Binato & Resende 2003) Aiex, R. M. , Binato, S. & Resende, M. G. C. (2003). Parallel GRASP with path-relinking for job shop scheduling, Parallel computing in numerical optimization, 29(4), 393–430.

(Alba & Troya 2000) Alba, E., & Troya, J. M. (2000, September). Cellular evolutionary algorithms: Evaluating the influence of ratio. In International Conference on Parallel Problem Solving from Nature (pp. 29-38). Springer, Berlin, Heidelberg.

(Alba 2002) Alba, E. (2002). Parallel evolutionary algorithms can achieve super-linear performance. Information Processing Letters, 82(1), 7-13. doi:10.1016/s0020-0190(01)00281-2

(Alba&Dorronsoro 2008) Alba, E., Dorronsoro, B., 2008. Cellular Genetic Algorithms, Operations Research/Computer Science Interfaces, Vol. 42. Springer-Verlag, Heidelberg.

(Alba&Tomassini 2002) Alba, E., Tomassini,M., 2002. Parallelism&evolutionary algorithms. IEEE Transactions on Evolutionary Computation 6, 5, 443–462.

(Applegate & Cook 1991)	Applegate, D. & Cook,W. (1991). A computational study of the job shop scheduling problem, ORSA Journal of Computing, 3(2), 149–156.
(Baker & Scudder 1990)	Baker, k. & Scudder, G. (1990). Sequencing with earliness & trainess penalties: A review, Operations Research, 38, 22–36.
(Baker 1974)	Baker, K. (1974). Introduction to sequencing & scheduling, NewYork, John Wiley & Sons.
(Baker 1974)	Baker, K.R., (1974). Introduction to Sequencing&Scheduling, Wiley, New York.
(Balas 1969)	Balas, E., (1969). Machine Scheduling via disjunctive graphs : An implicit enumeration algorithm, Operation Research, 17, 941–957.
(Balas&Vazacopoulos 1998)	Balas, E.&Vazacopoulos, A., (1998). Guided local search with shifting bottleneck for job shop scheduling. Management Science, 44(2), 262–275.
(Baptiste & al. 2001)	Baptiste, P., Pape, C. L., & Nuijten, W. (2001). Cumulative Scheduling Problems. Constraint-Based Scheduling International Series in Operations Research & Management Science, 149-158. doi:10.1007/978-1-4615 1479-4_7
(Barr & Hickman 1993)	Barr, R. S., & Hickman, B. L. (1993). Feature Article—Reporting Computational Experiments with Parallel Algorithms: Issues, Measures,&Experts' Opinions. ORSA Journal on Computing, 5(1), 2-18. doi:10.1287/ijoc.5.1.2
(Bean 1994)	Bean, J. (1994). Genetic algorithms & random keys for sequencing & optimizaion, ORSA Journal on Computing, 6(2), 154–160.
(Bean 1994)	Bean, J. C. (1994). Genetics & random keys for sequencing & optimization, ORSA Journal on Computing, 6, 154–160.
(Belding 1995)	Belding, T. C. (1995). The distributed genetic algorithm revisited. arXiv preprint adap-org/9504007.
(Belegundu & Chandrupatla 2019)	Belegundu, A. D., & Chandrupatla, T. R. (2019). Optimization Concepts&Applications in Engineering. doi:10.1017/9781108347976
(Belegundu & Rajan 1988)	Belegundu, A., & Rajan, S. (1988). A shape optimization approach based on natural design variables&shape functions. Computer Methods in Applied Mechanics&Engineering, 66(1), 87-106. doi:10.1016/00457825(88)90061-8
(Belegundu 2011)	Belegundu, A. D., & Chandrupatla, T. R. (2011). Optimization Concepts&Applications in Engineering. doi:10.1017/cbo9780511975905
(Bellman 1954)	Bellman, R. (1954). Some Problems in the Theory of Dynamic Programming. Econometrica, 22(1), 37. doi:10.2307/1909830
(Benderbal 2018)	Benderbal HH (2018) Développement d'une nouvelle famille d'indicateurs de performance pour la conception d'un système manufacturier reconfigurable (RMS) approches évolutionnaires multicritères. Ph.D. dissertation, University of Lorraine
(Bennett & Hancock 1920)	Bennett, A. A., & Hancock, H. (1920). Theory of Maxima&Minima. The American Mathematical Monthly, 27(6), 266. doi:10.2307/2972316
(Bennett & Hancock 1920)	Bennett, A. A., & Hancock, H. (1920). Theory of Maxima&Minima. The American Mathematical Monthly, 27(6), 266. doi:10.2307/2972316
(Bermudez 1998)	Bermudez, J. (1998). Advanced Planning&Scheduling: Is it as good as it sounds? The report on Supply Chain Management, March, 3–18.
(Berretta&Rodrigues 2004)	Berretta, R.&Rodrigues, L. F., (2004). A Genetic algorithm for a multistage capacitated lot-sizing problem, International Journal of Production Economics, 87(1), 67–81.

(Biggs & Wilson 1986)	Biggs, N., Lloyd, E. K., & Wilson, R. J. (1986). Graph Theory, 1736-1936. Oxford University Press.
(Binato, Hery, Loewenstern & Resende 2002)	Binato, S., Hery, W.J., Loewenstern, D.M., & Resende, M.G.C., (2002). A GRASP for job shop scheduling. In: Ribeiro, C.C., Hansen, P. (Eds.), Essays&Surveys in Metaheuristics. Kluwer Academic Publishers.
(Blackstone, Phillips & Hogg 1982)	Blackstone, J., Phillips, D., & Hogg, G. (1982). A state of the art survey of dispatching rules for maunfacturing job shop operations, International Journal of Production Research, 20,26–45.
(Blazewicz,Ecker, Schmidt,&Weglarz 1994)	Blazewicz, J., Ecker, K. H., Schmidt, G. andWeglarz, J., (1994). Scheduling in Computer&Manufacturing Systems, Springer.
(Borchers 2009)	Borchers, B. (2009). Review of practical optimization: Algorithms&engineering applications by Andreas Antoniou&Wu-Sheng Lu (Springer Verlag, 2007). ACM SIGACT News, 40(1), 20-22. Borchers, B. (2009). Review of practical optimization: Algorithms&engineering applications by Andreas Antoniou&Wu-Sheng Lu (Springer Verlag, 2007). ACM SIGACT News, 40(1), 20-22. doi:10.1145/1515698.1515704no. 1, 2009, pp. 20–22., doi:10.1145/1515698.1515704.
(Bowlin 1998)	Bowlin, W. F. (1998). Measuring performance: An introduction to data envelopment analysis (DEA). The Journal of Cost Analysis, 15(2), 3-27.
(Brandimarte 1993)	Brandimarte, P., (1993). Routing&scheduling in a flexible job shop by tabu search, Annals of Operations Research, 41, 157–183.
(Brucker & Knust 2011)	Brucker, P., & Knust, S. (2011). Resource-Constrained Project Scheduling. Complex Scheduling, 117-238. doi:10.1007/978-3-642-23929-8_3
(Brucker 1998)	Brucker, P., (1998). Scheduling Algorithms, Springer.
(Bruker&Schlie 1990)	Bruker, P.&Schlie, R., (1990). Job-shop scheduling with multi-purpose machines. Computing, 45, 369–375.
(Butt 2020)	Butt, J. (2020). A Strategic Roadmap for the Manufacturing Industry to Implement Industry 4.0. Designs, 4(2), 11. doi:10.3390/designs4020011
(Buttazzo 2011)	Buttazzo, G. C. (2011). Hard real-time computing systems: predictable scheduling algorithms&applications (Vol. 24). Springer Science & Business Media.
(Capawa 2020)	Fotsoh, E. C., Mebarki, N., Castagna, P., & Berruet, P. (2020). A Classification for Reconfigurable Manufacturing Systems. Springer Series in Advanced Manufacturing Reconfigurable Manufacturing Systems: From Design to Implementation, 11-28. doi:10.1007/978-3-030-28782-5_2
(Chambers 1996)	Chambers, J. B., (1996). Classical&Flexible Job Shop Scheduling by Tabu Search. PhD thesis, University of Texas at Austin, Austin, U.S.A.
(Charon 1996)	Charon, I., (1996). Germinated, A.&Hudry, O., Methodes d'Optimization Combinatoires, Paris, France: Masson.
(Cheng & Gen 2019)	Cheng, J. R., & Gen, M. (2019). Accelerating genetic algorithms with GPU computing: A selective overview. Computers & Industrial Engineering, 128, 514-525.
(Cheng, Gen & Sasaki 1995)	Cheng, R., Gen, M. & Sasaki, M. (1995). Film-copy deliverer problem using genetic algorithms, Computers&Industrial Engineering , 29(1–4), 549–553.
(Cheng, Gen&Tsujimura 1996)	Cheng, R., Gen, M.&Tsujimura, Y. (1996). A Tutorial Sur-vey of Job-Shop Scheduling Problems Using Genetic Algorithm, Part I: Representation. Computers&Industrial Engineering, 30(4), 983–997,

(Cheng, Gen&Tsujimura 1999)	Cheng, R., Gen, M.&Tsujimura, Y. (1999). A Tutorial Survey of Job-shop Scheduling Problems using Genetic Algorithms, Part II: Hybrid Genetic Search Strategies, Computers&Industrial Engineering, 36(2), 343–364.
(Conway, Maxwell&Miller 1967)	Conway, R.W., Maxwell, W.L.,&Miller, L.W., (1967). Theory of Scheduling, Addison- Wesley, Reading, MA.
(Cooper & Hinde 2003)	Cooper, J., & Hinde, C. (2003). Improving Genetic Algorithms' Efficiency Using Intelligent Fitness Functions. Developments in Applied Artificial Intelligence Lecture Notes in Computer Science, 636-643. doi:10.1007/3-540-45034-3_64
(Crainic 2019)	Crainic, T. (2019). Parallel metaheuristics&cooperative search. In Handbook of Metaheuristics (pp. 419-451). Springer, Cham.
(Croce, Tadei & Volta 1995)	Croce, F., Tadei, R., & Volta, G. (1995). A genetic algorithm for the job shop problem, Computer & Operations Research, 22, 15–24.
(Dahi & Alba 2019)	Dahi, Z. A., & Alba, E. (2019). The grid-to-neighbourhood relationship in cellular GAs: From design to solving complex problems. Soft Computing, 24(5), 3569-3589. doi:10.1007/s00500-019-04125-w
(Dauzere-Pers & Lasserre 1993)	Dauzere-Pers, S. & Lasserre, J. (1993). A modified shifting bottleneck procedure for job-shop scheduling, International Journal of Production Researches, 31, 923–932.
(Davis 1985)	Davis, L. (1985). Job shop scheduling with genetic algorithms, Proceedings of the First International Conference on Genetic Algorithms, 136–140.
(De Jong 1994)	De Jong, K. (1994). Genetic algorithms: a 25 year perspective, Computational Intelligence: Imitating Life, 125–134.
(de Melo & al.2020)	de Melo, V. V., Fazenda, Á. L., Sotto, L. F. D. P., & Iacca, G. (2020, April). A MIMD Interpreter for Genetic Programming. In International Conference on the Applications of Evolutionary Computation (Part of EvoStar) (pp. 645-658). Springer, Cham.
(Dellaert, Jeunet&Jornard 2000)	Dellaert, N., Jeunet, J.&Jornard, N., (2000). A genetic algorithm to solve the general multi-level lot-sizing problem with time-varying costs, International Journal of Production Economy, 68, 241–257.
(Diaz & al. 2018)	Diaz, J. E., Handl, J., & Xu, D. (2018). Integrating meta-heuristics, simulation&exact techniques for production planning of a failure-prone manufacturing system. European Journal of Operational Research, 266(3), 976-989. doi:10.1016/j.ejor.2017.10.062
(Dobbs & Nelson 1976)	Dobbs, M. W., & Nelson, R. B. (1976). Application of optimality criteria to automated structural design. AIAA Journal, 14(10), 1436-1443. doi:10.2514/3.7232
(Dorndorf & al. 2000)	Dorndorf, U., Pesch, E., & Phan-Huy, T. (2000). A Time-Oriented Branch-and-Bound Algorithm for Resource-Constrained Project Scheduling with Generalised Precedence Constraints. Management Science,46(10),1365-1384. doi:10.1287/mnsc.46.10.1365.12272
(Dorndorf & Pesch 1995)	Dorndorf, W. & Pesch, E. (1995). Evolution based learning in a job shop scheduling enviorment, Computer & Operations Research, 22, 25–40.
(Dorronsoro&al. 2013)	Dorronsoro, B., Danoy, G., Nebro, A. J., & Bouvry, P. (2013). Achieving super-linear performance in parallel multi-objective evolutionary algorithms by means of cooperative coevolution. Computers & Operations Research, 40(6), 1552-1563. doi:10.1016/j.cor.2011.11.014
(Eck 2003)	Eck, M. (2003). Advanced Planning&Scheduling, BWI paper: http://obp.math.vu.nl/logistics/papers/vaneck.doc
(Eguia & al.2017)	Eguia I, Villa G, Lozano S (2017) Efficiency assessment of reconfigurable manufacturing systems. Proc Manuf 11:1027–1034

(ewajinda&Chongstitvatana 2009) ewajinda, Y., & Chongstitvatana, P. (2009). Hardware architecture&FPGA implementation of a parallel elitism-based compact genetic algorihm. TENCON 2009 - 2009 IEEE Region 10 Conference. doi:10.1109/tencon.2009.5396138

(Fang, Ross & Corne 1993) Fang, H., Ross, P. & Corne, D. (1993). A promising genetic algorithm approach to jobshop scheduling re-scheduling&open-shop scheduling problems, Proceedings of the fifth International Conference on Genetic Algorithms, Morgan Kaufmann Publishers, San Mateo, CA, 375–382.

(Ferrucci & al. 2018) Ferrucci, F., Salza, P., & Sarro, F. (2018). Using hadoop MapReduce for parallel genetic algorithms: a comparison of the global, grid&island models. Evolutionary computation, 26(4), 535-567.

(Fisher & Thompson 1963) Fisher, H. & Thompson, G. (1963). Probabilistic learning combinations of job-shop scheduling rules, Industrial Scheduling, 15, 1225–1251.

(Fisher&Thompson 1963) Fisher, H.&Thompson,G. L., (1963). Probabilistic learning combinations of local job-shop scheduling rules, In J.F. Muth&G.L. Thompson, Industrial Scheduling, Prentice Hall, 225-251.

(Fletcher 2000) Fletcher, R. (2000). Practical Methods of Optimization. doi:10.1002/9781118723203

(Fralkenauer & Bouffoix 1991) Fralkenauer, E. & Bouffoix, S. (1991). A genetic algorithm for job shop, Proceedings of IEEE International Conference on Robotics & Automation, 824–829.

(Franke & Sarstedt 2019) Franke, G., & Sarstedt, M. (2019). Heuristics versus statistics in discriminant validity testing: a comparison of four procedures. Internet Research.

(French 1982) French, S., (1982). Sequencing&Scheduling. Mathematics&its applications, Ellis Horwood Limited.

(Garey&Johnson 1978) Garey, M. R.&Johnson,D. S., (1978). Strong NP-Completeness Results: Motivation, Examples&Implications, Journal of the ACM, 25, 499–508.

(Garey, Johnson&Sethi 1976) Garey, M. R., Johnson, D. S.&Sethi, R., (1976). The complexity of flowshop&jobshop scheduling, textitMathematics of Operations Research, 1(2), 117–129.

(Gen &Cheng 1997) Gen, M.&R., Cheng, (1997). Genetic Algorithms&Engineering Design, John Wiley Sons, New York.

(Gen &Cheng 2002) Gen, M.&Cheng, R., (2002). Genetic Algorithms&Engineering Optimization, New York: John Wiley & Sons.

(Gen & al.2016) Gen, M., Hao, X., & Zhang, W. (2016). Advances in Hybrid Metaheuristics for Stochastic Manufacturing Scheduling: Part I Models&Methods. Advances in Intelligent Systems&Computing Proceedings of the Tenth International Conference on Management Science&Engineering Management, 1063-1077. doi:10.1007/978-981-10-1837-4_88

(Gen, Tsujimura & Kubota 1994) Gen, M.,Tsujimura, Y., & Kubota, E. (1994). Solving job-shop scheduling problem using genetic algorithms, Proceedings of the 16th International Conference on Computers & Industrial Engineering, 576–579, Ashikaga, Japan.

(Geyer 2009) Geyer, P. (2009). Component-oriented decomposition for multidisciplinary design optimization in building design. Advanced Engineering Informatics, 23(1), 12-31. doi:10.1016/j.aei.2008.06.008

(Giffler & Thompson 1960) Giffler, B. & Thompson, G. (1960). Algorithms for solving production scheduling problem,Operations Research, 8(4), 487–503.

(Goldberg 1989)	Drews, G. N., & Goldberg, R. B. (1989). Genetic control of flower development. Trends in Genetics, 5, 256-261. doi:10.1016/0168-9525(89)90098-x
(Gomory & Baumol 1960)	Gomory, R. E., & Baumol, W. J. (1960). Integer Programming&Pricing. Econometrica, 28(3), 521. doi:10.2307/1910130
(Goncalves & Beirao 1999)	Gonc¸alves, J.F., & Beir˜ao, N.C., (1999). Um Algoritmo Gen`etico Baseado em Chaves Aleat`orias para Sequenciamento de Operac¸ ˜oes. Revista Associac¸ ao Portuguesa de Desenvolvimento e Investigac¸ ao Operacional, 19, 123–137 (in Portuguese).
(Goncalves, Magalhaes Mendes&Resende 2005)	Gonc¸alves, J. F., Magalh˜aes Mendes, J. J.&Resende, M. G. C., (2005). A hybrid genetic algorithm for the job shop scheduling problem, European Journal of Operational Research, 167(1), 77–95.
(Grabot&Geneste 1994)	Grabot, B.&Geneste, L., (1994). Dispatching rules in scheduling: A fuzzy approach, International Journal of Production Research, 32(4), 903-915.
(Gravetter & Forzano 2018)	Gravetter, F. J., & Forzano, L. A. B. (2018). Research methods for the behavioral sciences. Cengage Learning.
(Griva 2008)	Griva, I. (2008). Linear&nonlinear optimization, second edition. Society for Industrial&Applied Mathematics.
(Hasting & Yeh 1992)	Hastings, N. A. J. & Yeh, C. -H. (1992). Bill of manufacture, Production&Inventory Management Journal, 4th Quarter, 27–31.
(Haupt 1989)	Haupt, R. (1989). A survey of priority-rule based scheduling problem, OR Spectrum, 11, 3–16.
(Helal & al. 2017)	Helal, M. H., Fan, C. T., Liu, D. Y., & Yuan, S. M. (2017, November). Peer-to-Peer Based Parallel Genetic Algorithm. In 2017 International Conference on Information, Communication&Engineering (ICICE) (pp. 535 538). IEEE.
(Holland 1975)	Holland, J., (1975). Adaptation in Natural&Artificial Systems, University of Michigan Press, Ann Arbor.
(Huang & al.2018)	Huang S, Wang G, Yan Y (2018) Delayed reconfigurable manufacturing system. Int J Prod Res 57(8):2372–2391
(Ida&Osawa 2005)	Ida, k.&Osawa, A., (2005). proposal of algorithm for shortening idel time on job-shop scheduling problem&its numerical experiments, J Jpn Ind manage Assoc, 56(4), 294–301.
(Jain &Meeran 1998)	Jain A. S.&Meeran S. (1998). A State-of -the-art Review of Job-shop Scheduling Techniques, Technical Report, Department of Applied Physics, Electronics&Mechanical Engineering, University of Dundee, Scotland.
(Kacem, Hammadi&Borne 2002)	Kacem, I., Hammadi, S.&Borne, P., (2002). Approach by localization&multiobjective evolutionary optimization for flexible job-shop scheduling problems. IEEE Trans. Systems,Man,&Cybernetics-Part C, 32(1), 1-13.
(Kallrath 2002)	Kallrath, J. (2002). Planning&scheduling in the process industry. OR spectrum, 24(3), 219-250.
(Kanso 2010)	Kanso M (2010) Contribution à la construction des configurations des systèmes manufacturiers: une approche basée sur la décision multi-critères. Ph.D. dissertation, University of South Brittany
(Kantorovich 1960)	Kantorovich, L. V. (1960). Mathematical Methods of Organizing&Planning Production. Management Science, 6(4), 366-422. doi:10.1287/mnsc.6.4.366

(Karp & Flatt 1990) Karp, A. H., & Flatt, H. P. (1990). Measuring parallel processor performance. Communications of the ACM, 33(5), 539-543. doi:10.1145/78607.78614

(Kenneth & Dan 2018) Kenneth R. Baker Dan Trietsch (2018) Resource-Constrained Project Scheduling. Principles of Sequencing&Scheduling, 483-509. doi:10.1002/9781119262602.ch17

(Ketfi 2005) Ketfi A (2005) Une Approche Générique pour la Reconfiguration Dynamique des Applications à base de Composants Logiciels. Ph.D. dissertation, University of Grenoble I

(Kim, Yamazaki & Gen 2004) Kim, K., Yamazaki, G., Lin, L. & Gen, M. (2004). Network-based hybrid genetic algorithm for scheduling in FMS environments, Artificial Life & Robotics, 8, 67–76.

(Kincaid & Cheney 1992) I., E., Kincaid, D., & Cheney, W. (1992). Numerical Analysis--Mathematics of Scientific Computing. Mathematics of Computation, 59(199), 297. doi:10.2307/2152998

(Kirk & al. 2016) Kirk, D. B., & Wen-Mei, W. H. (2016). Programming massively parallel processors: a hands-on approach. Morgan kaufmann.

(Knowles&Corne 2000) Knowles, J. D.&Corne, D. W., (2000). M-PAES: A Genetic Algorithm for Multiobjective Optimization, Proc. of the Congress on Evolutionary Computation, 1, 325–332.

(Koren & al. 1999) Koren Y, Heisel U, Jovane F, Moriwaki T, Pritschow G, Ulsoy G, Van Brussel H (1999) Reconfigurable manufacturing systems: introduction. CIRP Ann 48(2):527–540

(Koren & Shpitalni 2010) Koren Y, Shpitalni M (2010) Design of reconfigurable manufacturing systems. J Manuf Syst 29(4):130–141

(Krasnogor 2002) Krasnogor, N., (2002), Studies on the theory&design space of Genetic algorithms, Ph.D.dissertation, Univ. of the West of England, Bristol, U.K.

(Kubota 1995) Kubota, A. (1995). Study on optimal scheduling for manufacturing system by genetic algorithms, Master's thesis, Asikaga Institute of Technology, Ashikaga, Japan.

(Kucuksayacigil & Ulusoy 2020) Kucuksayacigil, F., & Ulusoy, G. (2020). Hybrid genetic algorithm for bi-objective resource-constrained project scheduling. Frontiers of Engineering Management. doi:10.1007/s42524-020-0100-x

(Kuhn & Tucker 2013) Kuhn, H. W., & Tucker, A. W. (2013). Nonlinear Programming. Traces&Emergence of Nonlinear Programming, 247-258. doi:10.1007/978-3-0348-0439-4_11

(Kuhn 2013) Kuhn, H. W. (2013). Nonlinear Programming: A Historical View. Traces&Emergence of Nonlinear Programming, 393-414. doi:10.1007/978-3-0348-0439-4_18

(Kusiak 2000) Kusiak, A., (2000). Computational Integrated in Design & Manufacturing, John Wiley & Sons, New York.

(Lafou 2016) Lafou M (2016) Contribution à la conception de systèmes d'assemblage automobile, performants, pérennes et innovants par des indicateurs technologiques et économiques répondant à la diversité croissante des produits. Ph.D. dissertation, University of Paris-Saclay

(Lameche & al.2017) Lameche K, Najid NM, Castagna P, Kouiss K (2017) Modularity in the design of reconfigurable manufacturing systems. IFAC-PapersOnLine 50(1):3511–3516

(Lameche 2018) Lameche K (2018) Proposition d'une méthodologie pour la conception des systèmes de production reconfigurable et d'un outil associé d'aide à la décision par simulation de flux. Ph.D. dissertation, University of Bretagne Loire

(Lance 2019)	Lance D. Chambers (2019).Structure&Performance of Fine-Grain Parallelism in GeneticSearch. Practical Handbook of Genetic Algorithms, 139-154. doi:10.1201/9780429128332-7
(Lee & Chen 2001)	Lee, C. Y. & Chen, Z. -L. (2001). Machine scheduling with transportation considerations,Journal of Scheduling, 4, 3–24.
(Lee 2018)	Lee, C. (2018). A review of applications of genetic algorithms in operations management. Engineering Applications of Artificial Intelligence, 76, 1-12. doi:10.1016/j.engappai.2018.08.011
(Levine 1996)	Levine, D. (1996). Users guide to the PGAPack parallel genetic algorithm library. doi:10.2172/366458
(Lopez&Ramirez 2005)	Lopez, O.&Ramirez, M., (2005).A STEP-based manufacturing information system to share flexible manufacturing resources data, Journal of Intelligent Manufacturing, 16(3), 287–301.
(Luo & al. 2018)	Luo, J., & Baz, D. E. (2018). A Survey on Parallel Genetic Algorithms for Shop Scheduling Problems. 2018 IEEE International Parallel&Distributed Processing Symposium Workshops (IPDPSW). doi:10.1109/ipdpsw.2018.00103
(Maganha & al.2018)	Maganha I, Silva C, Ferreira LMDF (2018) Understanding reconfigurability of manufacturing systems: an empirical analysis. J Manuf Syst 48:120–130
(Manaseer & Hwaitat 2018)	Manaseer, S., & Hwaitat, A. K. (2018). Measuring Parallel Performance of Sorting Algorithms. Modern Applied Science, 12(10), 23. doi:10.5539/mas.v12n10p23
(Maruyama & al. 1993)	Maruyama, T., Hirose, T., & Konagaya, A. (1993, June). A fine-grained parallel genetic algorithm for distributed parallel systems. In proceedings of the 5th International Conference on Genetic Algorithms (pp. 184 190).
(Mati, Rezg&Xie 2001)	Mati, Y., Rezg, N.&Xie, X., (2001). An Integrated Greedy Heuristic for a Flexible Job Shop Scheduling Problem, IEEE International Conference on Systems, Man,&Cybernetics, 4, 2534–2539.
(Mehrabi & al.2000)	MehrabiMG, UlsoyAG,KorenY(2000b) Reconfigurable manufacturing systems&their enabling technologies. Int J Manuf Technol Manage 1(1):114–131
(Menouer 2018)	Menouer, T. (2018). Solving combinatorial problems using a parallel framework. Journal of Parallel&Distributed Computing, 112, 140-153. doi:10.1016/j.jpdc.2017.05.019
(MESX Joint Working Group 2004)	MESX Joint Working Group. (2004). MESX White Paper, [Online]. Available:http://www.mstc.or.jp/faop/doc/informative/MESX-WP.pdf, (in Japanese).
(Montgomery 2017)	Montgomery, D. C. (2017). Design&analysis of experiments. John wiley & sons.
(Moon 2004)	Moon, C., (2004). Evolutionary System Approach for Advanced Planning in Multi-Plant Chain, PhD dissertation, Waseda University, Japan.
(Moon&Seo 2005)	Moon, C.&Seo, Y., (2005). Advanced planning for minimizing makespan with load balancing in multi-plant chain, International Journal of Production Research. 43(20), 4381–4396.
(Moon, Kim&Gen 2004)	Moon, C., Kim, J. S.&Gen, M., (2004). Advanced planning&scheduling based on precedence&resource constraints for e-plant chains, International Journal of Production Research, 42(15), 2941–2955.
(Moon, Lee&Gen 2004)	Moon, C., Lee, Y. H.&Gen,M., (2004). Evolutionary Algorithm for Process plan Selection with Multiple Objectives, Journal of Industrial Engineering&Management Systems, 3(2), 125–131.

(Moon, Seo&Lee 2002) Moon, C., Lee, M., Seo, Y.&Lee, Y. H., (2002). Integrated machine tool selection&operation sequencing with capacity&precedence constraints using genetic algorithm, Computers & Industrial Engineering, 43(3), 605–621.

(Morton & Pentico 1993) Morton, T. & Pentico, D. (1993). Heuristic scheduling systems-with applications to a production systems & project management, New York, John Wiley & Sons.

(Mudge 2003) Mudge, M. R. (2003). An introduction to numerical methods&analysis, by James F. Epperson. Pp. 556. £32.50. 2002. ISBN 0 471 3164744 (Wiley). The Mathematical Gazette, 87(509), 413-413. doi:10.1017/s002555720017353x

(N.Instruments 2011) Understanding Parallel Hardware: Multiprocessors, Hyperthreading, Dual-Core, Multicore&FPGAs - National Instruments. (2011). Retrieved from http://www.ni.com/tutorial/6097/en/

(Najid, Dauzere-Peres&Zaidat 2002) Najid, N.M., Dauzere-Peres, S.&Zaidat, A., (2002). A modified simulated annealing method for flexible job shop scheduling problem, IEEE International Conference on Systems, Man&Cybernetics, 5(6).

(Nakano & Yamada 1992) Nakano, R. & Yamada, T. (1992). Conventional genetic algorithms for job-shop problems, Proceedings of the fourth International Conference on Genetic Algorithms, Morgan Kaufmann Publishers, San Mateo, CA, 477–479.

(Ng & al. 2014) Ng, C. K., & Li, D. (2014). Test problem generator for unconstrained global optimization. Computers & operations research, 51, 338-349.

(Nishi&Konishi 2005) Nishi, T.&Konishi, M., (2005). An autonomous decentralized supply chain planning system for multi-stage production processes, Journal of Intelligent Manufacturing, 16(3), 259–275.

(Nishioka 1999) Nishioka, Y. (1999). Supply Chain Planning for Computer Optimized Manufacturing, Management Systems, Japan Industrial Management Association, 9(3), 132–136 (in Japanese).

(Nishioka 2002) Nishioka, Y. (2002). A New Turn of Manufacturing Enterprise Architecture with Advanced Planning&Scheduling, Management Systems, Japan Industrial Management Association, 12(1), 9–13 (in Japanese).

(Norman & Bean 1995) Norman, B. & Bean, J. (1995). Random keys genetic algorithm for job-shop scheduling: unabridged version, Technical report, University of Michigan, Ann Arbor.

(Norman & Bean 1995) Norman, B. & Bean, J. (1995). Random keys genetic algorithm for scheduling, Technical report, Universuty of Michigan, Ann Arbor.

(Nowicki & Smutnicki 1996) Nowicki, E. & Smutnicki, C. (1996). A fast taboo search algorithm for the job shop problem,Management Science, 42(6), 797–813.

(Nowicki&Smutnicki 1996) Nowicki, E.&Smutnicki, C., (1996). A fast taboo search algorithm for the job-shop problem, Management Science, 42(6), 797-813.

(Nowicki&Smutnicki 2005) Nowicki, E.&Smutnicki, C., (2005). An advanced tabu search algorithm for the job-shop problem, Journal of Scheduling, 8(2), 145–159.

(Okamoto, Gen & Sugawara 2005) Okamoto, A., Gen, M. & Sugawara, M. (2005). APS System based on Scheduler moGA & XML, Journal of the Society of Plant Engineers Japan, 17(2), 15–24 (in Japanese).

(Okamoto, Gen & Sugawara 2005) Okamoto, A., Gen, M. & Sugawara, M. (2005). Cooperation of Scheduling Agent & Transportation Agent in APS System, Proceedings of the J.S.L.S. Kyushu Division Conference,1–11 (in Japanese).

(Okamoto, Gen & Sugawara 2005) Okamoto, A., Gen, M. & Sugawara, M. (2005). APS System based on Scheduler moGA & XML, Journal of the Society of Plant Engineers Japan, 17(2), 15–24 (in Japanese).

(Okamoto, Gen & Sugawara 2006) Okamoto, A., Gen, M. & Sugawara, M. (2006). Integrated Data Structure&Scheduling Approach for Manufacturing & Transportation using Hybrid Multistage Operation-based Genetic Algorithm, Journal of Intelligent Manufacturing, 17, 411–421.

(Okamoto, Gen & Sugawara 2006) Okamoto, A., Gen, M. & Sugawara, M. (2006). Integrated Data Structure & Scheduling Approach for Manufacturing & Transportation using Hybrid Multistage Operation-based Genetic Algorithm, Journal of Intelligent Manufacturing, 17, 411–421.

(Okamoto, Gen & Sugawara 2006) Okamoto, A., Gen, M. & Sugawara, M. (2006). Integrated Scheduling Problem of Manufacturing & Transportation with Pickup & Delivery, International Journal of Logistics&SCM Systems, 1, 19–27.

(Orvosh & Davis 1994) Orvosh, D. & Davis, L. (1994). Using a genetic algorithm to optimize problems with Feasibility constrains, Proceedings of the First IEEE Conference on Evolutionary Computation, 548–552.

(Panwalkar & Iskander 1977) Panwalkar, S. & Iskander,W. (1977). A survey of scheduling rules, Operations Research, 25, 45–61.

(Paredis 1992) Paredis, J. (1992). Exploiting constraints as background knowledge for genetic algorithms: a case-study for scheduling, Parallel Problem Solving from Nature 2, PPSN-II, North-Holland. 231–240.

(Pravica 2011) Pravica, D. W., & Spurr, M. J. (2011). Mathematical modeling for the scientific method. Jones&Bartlett.

(Prins 2000) Prins, C., (2000). Competitive genetic algorithms for the open-shop scheduling problem, Mathematical Methods of Operations Research, 52, 389–411.

(PSLX Consortium 2003) PSLX Consortium (2003). PSLX Technical Specifications, Recommendation, Version 1.0, [Online]. Available:http://www.pslx.org/.

(Raa&Aghezzaf, 2005) Raa, B.&Aghezzaf, E. H., (2005). A robust dynamic planning strategy for lot-sizing problems with stochastic demands, Journal of Intelligent Manufacturing, 16(2), 207–213. 76. Su, P., Wu, N.&Yu, Z., (2003) Resource selection for distributed manufacturing in agile manufacturing, Proc. of IEEE International Conference on Systems, Man&Cybernetics, 2(1), 1578–1582.

(Roy&Sussmann 1964) Roy, B.&Sussmann, B. (1964). Les problems d'ordonnancement avec contraintes disjonctives, Note D.S. No. 9 bis, SEMA, Paris, France.

(Runkler & Bezdek 2019) Runkler, T. A., & Bezdek, J. C. (2019, April). Optimizing the C Index Using a Canonical Genetic Algorithm. In International Conference on the Applications of Evolutionary Computation (Part of EvoStar) (pp. 287-298). Springer, Cham.

(Sarkar 2018) Sarkar, T. (2018). What if your data is NOT Normal? Retrieved June 5, 2020, from https://towardsdatascience.com/what-if-your-data-is-not-normal-d7293f7b8f0

(Sarma & De Jong 1996) Sarma, J., & De Jong, K. (1996, September). An analysis of the effects of neighborhood size&shape on local selection algorithms. In International Conference on Parallel Problem Solving From Nature (pp. 236 244). Springer, Berlin, Heidelberg.

(Sehatbakhsh&al. 2013) Sehatbakhsh, N., Aliasgari, M., & Fakhraie, S. M. (2013). FPGA implementation of genetic algorithm for dynamic filter-bank-based multicarrier systems. 2013 8th International Conference on Design &

	Technology of Integrated Systems in Nanoscale Era (DTIS). doi:10.1109/dtis.2013.6527781
(Singh 2018)	Singh, R. (2018). Task Scheduling Techniques Based On Parallel Genetic Algorithm Approaches: A Review. International Journal of Computer Applications & Information Technology, 11(1), 229-241.
(Soukhal, Oulamara & Martineau 2005)	Soukhal, A., Oulamara, A. & Martineau, P. (2005). Complexity of flow shop scheduling problems with transportation constraints, European Journal of Operational Research, 161, 32–41.
(Spears & Dejong 1991)	Spears, W. M. & Dejong, K. A. (1991). On the virtues of parameterized uniform crossover, Proceedings of the Fourth International Conference on Genetic Algorithms, 230–236.
(Sprecher, Kolisch&Drexl 1995)	Sprecher, A., Kolisch R.&Drexl, A., (1995). Semi-active, active,&non-delay schedules for the resource-constrained project scheduling problem, European Journal of Operational Research, 80(1), 94–102.
(Storer, Wu & Vaccari 1992)	Storer, R., Wu, S., & Vaccari, R. (1992). New search spaces for sequencing problems with application to job shop scheduling, Management Science, 38(10), 1495–1510.
(Suresh & Sakthi 2019)	Suresh, K., & Sakthi, U. (2019). Analysis of heuristic-based multilevel thresholding methods for image segmentation using R programming. International Journal of Reasoning-based Intelligent Systems, 11(2), 151-160.
(Tan 2000)	Tan, W., (2000). Integration of process planning&scheduling-a review, Journal of Intelligent Manufacturing, 11(1), 51–63.
(Tan 2004)	Tan, W., (2004). A linearized polynomial mixed integer programming model for the integration of process planning&scheduling, Journal of Intelligent Manufacturing, 15(5), 593–605.
(Tanese 1989)	Tanese, R. (1989). Distributed genetic algorithms for function optimization. University of Michigan. Computer Science&Engineering Division.
(Tavakkoli-Moghaddam, Jolai, Vaziri, Ahmed,&Azaron 2005)	Tavakkoli-Moghaddam, R., Jolai, F., Vaziri, F., Ahmed, P.K.&Azaron, A., (2005). A hybrid method for solving stochastic job shop scheduling problems, Applied Mathematics&Computation, 170(1), 185–206.
(Thirer 2020)	Thirer, N. (2020, May). A FPGA implementation of a self-adaptive genetic algorithm. In Artificial Intelligence&Machine Learning for Multi-Domain Operations Applications II (Vol. 11413, p. 114131Y). International Society for Optics&Photonics.
(Tiwari & al. 2015)	Tiwari, N., Sarkar, S., Bellur, U., & Indrawan, M. (2015). Classification framework of MapReduce scheduling algorithms. ACM Computing Surveys (CSUR), 47(3), 1-38.
(Varghese & Raj 2016)	Varghese, B. M., & Raj, R. J. (2016). A survey on variants of genetic algorithm for scheduling workflow of tasks. 2016 Second International Conference on Science Technology Engineering&Management (ICONSTEM). doi:10.1109/iconstem.2016.7560870
(W3C 2003)	W3C (2003). Scalable Vector Graphics (SVG) 1.1 Specification, W3C Recommendation, [Online]. Available:http://www.w3.org/TR/2003/REC-SVG11-20030114/.
(W3C 2004)	W3C (2004). Extensible Markup Language (XML) 1.0 (Third Edition), W3C Recommendation, [Online]. Available:http://www.w3.org/TR/2004/REC-xml-20040204.
(Wächter 2017)	Wächter, A. (2017). Chapter 17: Nonlinear Optimization Algorithms. Advances&Trends in Optimization with Engineering Applications, 221-235. doi:10.1137/1.9781611974683.ch17

(Wang & al. 2007) Wang, D., Wang, G. G., & Naterer, G. F. (2007). Extended Collaboration Pursuing Method for Solver Larger Multidisciplinary Design Optimization Problems. AIAA Journal, 45(6), 1208-1221. doi:10.2514/1.21167

(Wang & al. 2018) Wang, S., Xing, J., & Li, J. (2018, September). A Fully Distributed Genetic Algorithm for Global Optimization of HVAC Systems. In International Conference on Smart City&Intelligent Building (pp. 609-620). Springer, Singapore.

(Wang, & Zheng 2001) Wang, L., & Zheng, D., (2001). An effective hybrid optimisation strategy for job-shop scheduling problems. Computers & Operations Research, 28, 585–596.

(Wiest 1964) Wiest, J. D., (1964). Some properties of schedules for large projects with limited resources, Operations Research, 12, 395–418.

(Wu&Weng 2005) Wu, Z.&Weng, M. X., (2005). Multiagent scheduling method with earliness&tardiness objectives in flexible job shops. IEEE Trans. System, Man,&Cybernetics-Part B, 35(2), 293–301.

(Xia&Wu 2005) Xia, W.&Wu, Z., (2005). An effective hybrid optimization approach for muti-objective flexible job-shop scheduling problem. Computers & Industrial Engineering, 48, 409–425.

(YahYaoui 2013) Khadidja, Y. (2013). Hybrid algorithm based on HBMO and GRASP for real-time task scheduling problem resolution. International Journal of Computer Science Issues (IJCSI), 9(4), 197.

(Yamada & Nakano 1992) Yamada, T.&Nakano, R. (1992). A genetic algorithm applicable to large-scale job-shop problems, Parallel Problem Solving from Nature: PPSN II, 281–290, Elsevier Science Publishers, North-Holland .

(Yang 2001) Yang, J. B. (2001). GA-based discrete dynamic programming approach for scheduling in FMS environments, IEEE Transactions on Systems, Man & Cybernetics, Part B, 31(5), 824–835.

(Yang 2001) Yang, J. B. (2001). GA-based discrete dynamic programming approach for scheduling in FMS environments, IEEE Transactions on Systems, Man & Cybernetics, Part B, 31(5), 824–835.

(Yang&Wang. 2000) Yang, S.&Wang, D. (2000). Constraint Satisfaction Adaptive Neural Network&Heuristics Combined Approaches for Generalized Job-Shop Scheduling, IEEE Trans. on Neural Networks, 11(2), 474–486.

(Ying &Liao 2004) Ying K. C.&Liao C. J. (2004). An Ant Colony Optimization for permutation flow-shop sequencing. Computers & Operations Research, 31(5), 791–801.

(Ying 2008) Ying, W. (2008). Improving the Computational Efficiency of Thermodynamical Genetic Algorithms. Journal of Software, 19(7), 1613-1622. doi:10.3724/sp.j.1001.2008.01613

(Yoshimura & Fujimi 2003) Yoshimura M, Izui K, Fujimi Y (2003) Optimizing the decision making process for largescale design problems according to criteria interrelationships. International journal of production research, 41(9):1987–2002

(Zhang&Gen 2005) Zhang, H.&Gen, M., (2005). Multistage-based genetic algorithm for flexible job-shop scheduling problem. Journal of Complexity International, 11, 223–232.

(Zhao & al. 2016) Zhao, Y., Chen, L., Xie, G., Zhao, J., & Ding, J. (2016). GPU implementation of a cellular genetic algorithm for scheduling dependent tasks of physical system simulation programs. Journal of Combinatorial Optimization, 35(1), 293-317. doi:10.1007/s10878-016-0007-y

(Özgüven, Özbakır & Yavuz 2010) Özgüven, C., Özbakır, L., & Yavuz, Y. (2010). Mathematical models for job-shop scheduling problems with routing and process plan flexibility. Applied Mathematical Modelling, 34(6), 1539-1548.

La liste des travaux scientifiques faisant partie de ce travail de thèse :

Revues scientifiques internationales

- AMRANE, A., Debbat, F., YAHYAOUI, K. GPU based Hybrid Cellular Genetic Algorithm for Job-Shop Scheduling Problem. *International Journal of Applied Metaheuristic Computing (IJAMC), 12*(3). (Indexed In: SCOPUS, Web of Science Emerging Sources Citation Index (ESCI), INSPEC)

Conférences internationales

- AMRANE, A., Debbat, F., YAHYAOUI, K. Cellular Genetic Algorithm for solving Job-Shop Scheduling Problem Using Graphic Processing Unit.In Proceedings of the third International Conference on Multimedia Information Processing (CITIM'2018). October 09-10, 2018, Mascara, Algeria
- AMRANE, A., Debbat, F., YAHYAOUI, K. An efficient Fine-Grained Parallel Genetic Algorithm for solving Job-Shop Scheduling Problem Using Graphic Processing Unit. In Proceedings of the third International Symposium on Informatics and its Applications (ISIA'2018). November 6-7, 2018, M'sila, Algeria