RÉPUBLIQUE ALGÉRIENNE DÉMOCRATIQUE ET POPULAIRE
MINISTÈRE DE L'ENSEIGNEMENT SUPÉRIEUR ET DE LA RECHERCHE SCIENTIFIQUE
UNIVERSITÉ SAAD DAHLAB BLIDA 1

FACULTÉ DES SCIENCES
DÉPARTEMENT DE MATHÉMATIQUES

Mémoire de projet de fin d'études

Présenté pour l'obtention du diplôme de master

Option : Recherche opérationnelle

Ordonnancement d'atelier multi-agents sur des machines dédiées

---

Présenté par :

CHABANE CHAOUCHE Nesserine

IHABDIOUNE Ghofrane

Devant le jury composé de :

Président de jury	*Professeur*	*CHELLALI Mustapha*	*(Université Blida 1)*
Examinateur	*MAA*	*DJAZOULI Adel*	*(ENP)*
Encadreur	*MCA*	*AMROUCHE Karim*	*(Université Alger 3)*

Blida, 2021-2022

Dédicace

Je dédie ce modeste travail :

Á la mémoire de mon oncle et mes grands parents,
Que dieu garde leurs âmes dans son vaste paradis.

Á mes chers parents qui m'ont toujours poussé
et motivé pour donner le meilleur de moi-même.

Á mes frères Tarek, Riad et Reda,à mes tantes,
et à ma chère cousine Ryma
qui n'a jamais cesser de me soutenir.

*** Nesrine ***

Dédicace

Avec joie, fierté et respect. Je dédie ce mémoire :

À mes très chers parents
Pour tout l'amour dont vous m'avez entouré,
pour tout ce que vous avez fait pour moi.

À mes très chers grands-parents.

À mon cher frère Djamel.

À ma chère petite soeur Melissa
source de vie, d'amour.

À ma chère cousine Amani.

À ma famille et mes cousines et cousins Nada et dania,
Karam, Ramy et Nazim.

À mes tantes et mon oncle
Qui ont toujours été la pour moi.

À mes amis Dounia, Incaf, Meriem et Yacine
pour leur soutien moral et encouragements
qu'ils m'ont toujours apportés.

À ma chere amie Nour el houda, source d'espoir et de motivation.

À ma chére binôme Nesrine
merci pour tous ces moments qu'on avait vécu ensemble.

***Ghofrane ***

Remerciements

En tout prmeier lieu, on remercie ALLAH le tout puissant de nous avoir donné la santé et la volonté d'entamer et de terminer ce travail.

Nous adressons nos sincères remerciements à notre encadrant M.Karim AMROUCHE pour avoir accepter de nous encadrer, on le remercie pour la qualité de son encadrement et ses conseils avisés.

Nos remerciements vont également à tous les enseignants du département de mathématique de l'université SAAD DAHLAB de Blida.

Nous tenons à remercier également tous les membres de jury d'avoir bien voulu participer à l'évaluation de ce travail.

ملخص

محور البحث الذي تتناوله هذه الأطروحة هو جدول الإنتاج الذي يمثل قضية رئيسية في الصناعة التحويلية، خلال دراستنا كنا مهتمين بمشكلة جدول ورشة العمل من نوع Flow-Shop مع عاملين مرتبطين بمجموعات فرعية منفصلة من المهام التي يتم تنفيذها حسب تدهور مهام العامل B، يشترك العاملان في نفس الموارد بحيث يهدف كل منهما إلى تقليل المعيار الخاص به، في هذه الحالة المعايير التي تم أخذها بعين الإعتبار هي: تقليل وقت المعالجة النهائي للعامل A و إستهلاك الطاقة للعامل B.

للإجابة على المشكل من الضروري العثور على أفضل تسلسل للمهام لتسوية بين معايير العاملين. في الجزء الأول قمنا بتطبيق نهج $\epsilon-contrainte$، إقترحنا نموذجين رياضيين و إستدلالين للبحث المحلي وهما La recherche tabou و Le recuit simulé بالإضافة إلى خوارزمية جينية NSGA-ii ، و قد تم إجراء تجارب لإثبات أدائهم.

الكلمات المفتاحية:

الجدولة، Flow-shop، وقت إنتهاء المعالجة، الطاقة، méta-heuristique، النمذجة الرياضية، طريقة تقريبية.

Résumé

L'axe de recherche traité dans ce mémoire est l'ordonnancement de la production qui représente un enjeu primordial dans l'industrie manufacturière, au cours de notre étude nous nous sommes intéréssées à un problème d'ordonnancement d'atelier de type flow shop avec deux agents associés à des sous ensembles de tâches disjoints en tenant compte de la contrainte de déterioration des tâches du deuxième agent, ces agents partagent les mêmes ressources et chacun d'eux vise à minimiser un critère propre à lui, dans notre cas les deux objectifs conflictuels considérés sont le temps de fin de traitement des tâches de l'agent A (C^A_{max}) et la consommation énergétique de l'agent B (TEC^B).
Pour répondre à la problématique il faut trouver les meilleurs séquences de compromis entre les critères des deux agents, dans une première partie nous avons appliqué l'approche $\epsilon - contrainte$, deux modèles mathématique sont proposés par la suite et deux métaheuristiques de recherche locale à savoir la recherche tabou et le recuit simulé, et un algorithme génétique NSGA-II, des expérimentations sont menées afin de montrer leurs performances.

Mots clés :

Ordonnancement, flow shop, makespan, énergie, méta-heuristiques, modélisation mathématique, méthodes approchées.

Abstract

The research axis treated in this thesis is the production scheduling which represents a primordial stake in the manufacturing industry, during our study we are interested in a Flow-Shop scheduling problem with two agents associated with disjiont sub-sets of jobs by taking into account the constraint of job's deterioration of the second agent, These agents share the same resources and each one of them aims to minimize a certain objective function, in our case the two conflicting objectives considered are the makespan of agent A (C^A_{max}) and the total energy consumption of agent B(TEC^B).
In order to answer the problem, it is necessary to find the best compromise sequences between the objective functions of the two agents, in a first part we applied the ϵ-constraint approach, two mathematical models are proposed thereafter and two local search metaheuristics namely the tabu search and the simulated annealing, and a genetic algorithm NSGA-II, experiments are carried out in order to show their performances.

Keywords :

Scheduling, flow shop, makespan, energy, Meta-heuristics, mathematical modeling, approximate methods.

TABLE DES MATIÈRES

TABLE DES FIGURES

LIST OF ALGORITHMS

LISTE DES TABLEAUX

INTRODUCTION

L'ordonnancement est un domaine très vaste il est lié à des secteurs d'activités très variés (entreprise industrielle, administration, informatique etc...), afin d'améliorer son niveau de comptétivité toute entité économique doit gérer efficacement la réalisation du produit ou service en tenant compte des contraintes temporelles et la disponibilité des ressources tout en visant la satisfaction du client, dans notre étude nous nous intéressons à l'ordonnancement dans les ateliers manufacturiers qui sont considérés comme une source intarissable de problèmes d'ordonnancement, ces problèmes font partie de la classe des problèmes combinatoires qui consistent à faire le choix d'une meilleure alternative parmi un nombre combinatoire d'alternatives.

Dans la littérature consacrée à la gestion de production, des centaines d'articles portant sur l'ordonnancement multicritères ont été publiés, la minimisation de la date de fin de traitement des tâches (Makespan) est l'un des critères les plus étudiés, de nos jours les ateliers de production font partie des secteurs énergivores donc dans le but de rester compétitifs et afin de minimiser l'impact environnemnetal ces entreprises industrielles doivent s'y mettre à maîtriser la facture énergétique qui est assez significative dans leurs comptes, pour cela de nombreux chercheurs s'intéressent de plus en plus aux problèmes d'ordonnancement qui traitent la gestion de la consommation énergétique.

Dans ce mémoire on traite un problème d'ordonnancement d'atelier de type Flow Shop à deux objectifs conflictuels avec deux agents en compétition où le temps d'exécution du deuxième agent varie avec le temps, les critères considérés sont la minimisation du Makespan du premier agent et la minimisation de l'énergie du deuxième, l'atelier est consitué de m machines et le nombre d'opérations de chaque tâche est égal à m, il y a des contraintes de précédence de type chaine entre les opérations d'une tâche : l'opération (i , j+1) ne peut commencer avant la fin de l'opération (i , j), on suppose que toutes les tâches sont disponibles à l'instant zéro et que leur préemption n'est pas autorisée.

Nous avons structuré la suite du manuscrit selon le plan suivant :

— Le **chapitre 1** sert à familiariser le lecteur avec les notions de base relatives à la théorie d'ordonnancement, où nous présentons aussi les approches utilisées pour la résolution des problèmes d'ordonnancement.

— Dans le **chapitre 2** des définitions des problèmes d'optimisation multicritères ont été présentées telles que la dominance, les approches Pareto et non Pareto.

— Le **chapitre 3** est consacré à la description de notre problème et à la définition de la notion de tâche déteriorable, nous rappelons aussi les différents scénarios des problèmes d'ordonnancement multi-agents.

— Le **chapitre 4** est dédié à des formulations mathématiques des problèmes obtenus suite à l'approche $\epsilon - contrainte$, et à la résolution du problème décrit au chapitre 3, on a proposé trois métaheuristiques : le recuit simulé et la recherche tabou qui sont des méthodes de recherche locale et un algorithme basé sur le principe de Pareto : NSGA-II, enfin pour clore ce chapitre une méthode d'aide à la décision multicritère est présentée qui a pour but de sélectionner la meilleure alternative en se basant sur la distance euclidienne.

— Le **chapitre 5** porte sur l'implémentation des approches citées dans le chapitre précédent et la présentation des résultats expérimentaux.

On termine notre étude par une conclusion générale qui récapitule l'ensemble des résultats obtenus dans ce mémoire.

CHAPITRE 1

PROBLÈMES D'ORDONNANCEMENT ET TECHNIQUES DE RÉSOLUTION

Dans ce chapitre introductif nous rappellons quelques notions de base de l'ordonnancement dans les ateliers , nous verrons également la représentation des problèmes d'ordonnancement et les différentes approches de résolution.

1.1 L'ordonnancement

L'ordonnancement représente l'un des domaines les plus étudiés en recherche opérationnelle depuis les années 50. En effet, les problèmes d'ordonnancement apparaissent dans de nombreux domaines d'application, notamment dans le secteur industriel (atelier, gestion de production), la construction, l'informatique et l'économie... ect.

Plusieurs définitions de l'ordonnancement ont été proposées dans la littérature, nous citons [7] :

Definition 1. Ordonnancer, c'est programmer l'exécution d'une réalisation en attribuant des ressources aux tâches et en fixant leur date d'exécution tout en cherchant à atteindre un objectif.

Definition 2. Un ordonnancement constitue une solution ou un calendrier au probléme à planifier, Il définit pour chaque tâche du problème les dates du début ou de fin de leur exécution et les ressources auxquelles elle sera affectée.

1.1.1 Classes d'ordonnancement

Definition 3. Un ordonnancement est semi-actif si aucune tâche ne peut être exécutée plus tôt sans changer l'ordre d'exécution.

Definition 4. Un ordonnancement actif est un ordonnancement où l'on ne peut pas avancer l'exécution d'une tâche sans retarder une autre tâche.

1.1.2 Éléments d'un problème d'ordonnancement

Dans un problème d'ordonnancement, quatre éléments fondamentaux sont nécessaires à la définition et la résolution du problème : les tâches, ressources, contraintes et objectifs.

1.1.2.1 Tâches :

Une tâche désigne une opération ou un ensemble d'opérations à réaliser.
Sa réalisation est caractérisée par :

- S_j la date de début d'exécution de la tâche J_j.
- P_j la durée opératoire de la tâche J_j.
- C_j la date de fin de traitement de la tâche J_j.
- L'exécution d'une tâche J_j peut commencer à la date r_j, et elle doit être achevée à la date d_j.

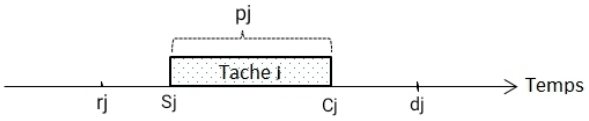


FIGURE 1.1 – Notations relatives à l'exécution d'une tâche

1.1.2.2 Ressources :

Une ressource est un moyen technique (machines ,outil ...ect) ou humain disponible en quantité limitée, destiné à être utilisé pour la réalisation d'une ou plusieurs tâches. Nous distinguons deux type de ressources pouvant être requises par les tâches :

- **Ressource consommable :** disponible en quantité limitée, cette quantité diminue au fur à mesure de son utilisation.
- **Ressource renouvlable :** la quantité disponible reste immuable durant la réalisation d'une ou plusieurs tâches.

Suivant la nature des ressources, une typologie des problèmes d'ordonnancement peut être présenté sur la Figure (1.2) :

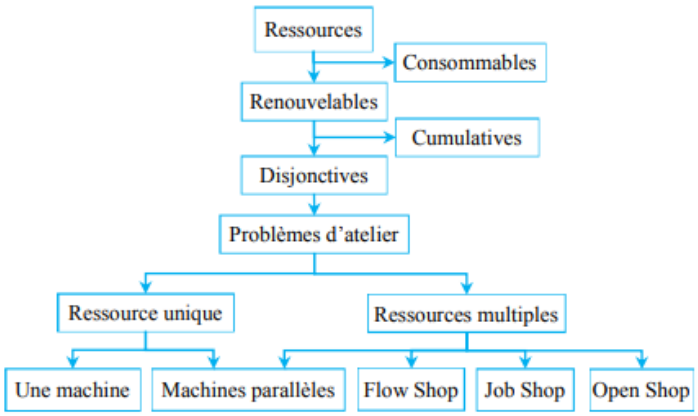


FIGURE 1.2 – Typologie des problèmes d'ordonnancement par ressource

1.1.2.3 Contraintes :

Les contraintes représentent les limites imposées par certaines variables (ressources). Dans les problèmes d'ordonnancement, deux types de contraintes sont distinguées :

— Les contraintes temporelles comprennent les contraintes de temps alloué, qui correspondent généralement aux impératifs liés aux tâches ou encore à la durée totale d'un ordonnancement.
— Les contraintes de ressources, quant à elles, traduisent la disponibilité des ressources et le fait qu'elle soit en quantité limitée.

1.1.2.4 Critères d'optimisation :

Dans la résolution des problèmes d'ordonnancement on cherche à optimiser un critère (ou fonction objectif) correspondant à une amélioration suivant au moins l'un des trois facteurs : coût, qualité ou délais, on résume les principaux critéres qui caractérise l'ordonnancement dans le tableau suivant :

Notation	Description
C_{max}	date d'achèvement de l'ordonnancement
$\sum C_j$	la somme des dates de fin de traitement
$\sum U_j$	le nombre de tâches en retard
$\sum W_j U_j$	la somme pondérée du nombre de tâches en retards
L_{max}	le décalage maximum
T_{max}	le retard maximum
$\sum_{j=1}^{n} w_j T_j$	la somme pondérée des retards

Table 1.1 – Critères d'optimisation

Afin de décrire les variables les plus couramment impliquées dans l'ordonnancement, nous présentons le tableau suivant :

Notation	Description
S_j	la date de début de la tâche J_j
C_j	la date de fin de traitement de la tâche J_j
r_j	la date de disponibilité de la tâche J_j
d_j	la date au plus tard de la tâche J_j
$L_j = c_j - d_j$	le retard algébrique de la tâche J_j
$T_j = Max\{c_j - d_j, 0\}$	Retard absolu de la tâche J_j
U_j	indicateur de retard de la tâche J_j

Table 1.2 – Paramètres d'ordonnancement

1.2 Les ateliers de production

En général, dans un problème d'ordonnancement d'atelier, nous avons un ensemble de n tâches $\{J_1, ..., J_n\}$ et un ensemble de m machines $\{M_1, ..., M_m\}$ et leur ordre d'utilisation pour fabriquer un produit, qui dépend de la nature de l'atelier considéré .
Alors un atelier est caractérisé par le nombre de machines qu'il contient ainsi que leurs types.
On distingue trois types de problème d'ordonnancement d'atelier :

1.2.1 Problème à une machine

Les problèmes d'ordonnancement à une machine se caractérisent par l'existence d'une seule ressource pour réaliser un ensemble de tâches, ces derniers sont constitués d'une seule opération qui

nécessite la même machine.

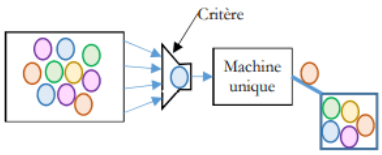


FIGURE 1.3 – problème à une machine

1.2.2 Probléme à machines parallèles

Dans ce cas, on dispose plusieurs ressources faisant la même fonction. On distingue trois types de machines parallèles :

- Machines identiques : Si toutes les machines ont la même vitesse d'exécution des tâches.
- Machines uniformes : Chaque machine a sa propre vitesse qui ne dépend pas des tâches à exécuter.
- Machines générales : Si les vitesses d'exécution des machines dépendent des tâches et sont différentes .

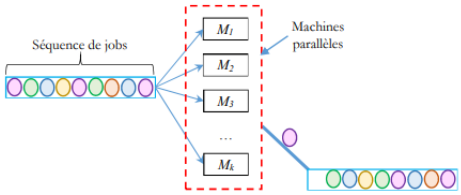


FIGURE 1.4 – problème à machines parallèles

1.2.3 Les problémes à machines dédiées

Dans le cas des machines dédiées, qui sont le plus souvent en série, les ressources sont spécialisées dans l'exécution de certaines opérations, on considére les trois types d'ateliers suivants :

1.2.3.1 Flow Shop :

L'atelier de type flow-shop, également appelé atelier à cheminement unique, les opérations élémentaires de toutes les tâches doivent être traitées par toutes les machines dans le même ordre .

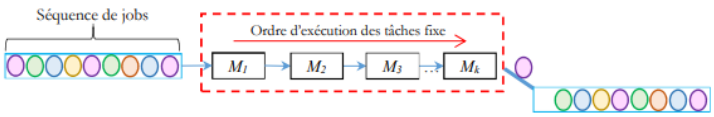


FIGURE 1.5 – problème à machines dédiées (Flow-Shop)

— **L'algorithme de Johnson** dévelopé dans le cadre de la recherche d'une séquence de durée minimale d'un atelier de type Flow-Shop[14], le déroulement de l'algorithme de Johnson peut être résumer dans le pseudo code de l'algorithme 1.

Algorithm 1: Algorithme de Johnson pour le problème de flow-shop à deux machines

Begin
While la liste des jobs non vide faire :
Placer dans l'ensemble U les jobs pour lesquels $P_{1i} \leq P_{2i}$;
Placer dans l'ensemble V les jobs pour lesquels $P_{1i} \geq P_{2i}$;
Ordonnancer les jobs de l'ensemble U dans l'ordre croissant des P_{1i} ;
Ordonnancer les jobs de l'ensemble V dans l'ordre décroissant des P_{2i} ;
On fusionne les deux ensembles U et V ;
On exécute la permutation obtenue sur les deux machines tout en respectant le même ordre.
End

Remarque : Nous avons utilisé l'algorithme de Johnson au chapitre 4.

1.2.3.2 Job Shop :

L'atelier de type job-Shop, également appelé atelier à cheminement multiple, chaque opération doit être traité par les machines dans un ordre bien déterminé .

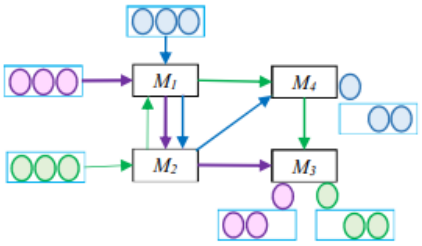


FIGURE 1.6 – problème à machines dédiées (Job-Shop)

1.2.3.3 Open Shop :

L'atelier de type Open-shop, également appelé atelier à cheminement libre, les opérations d'une tâche peuvent être exécutées dans un ordre quelconque.

La Figure (1.7) ci-dessous décrit les différents types d'atelier :

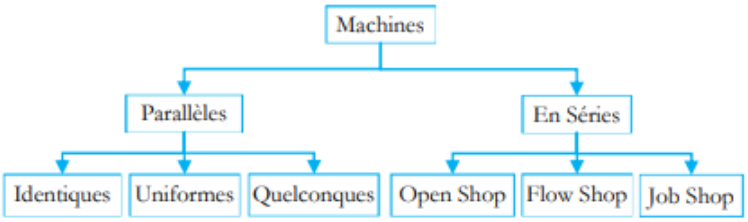


FIGURE 1.7 – les différents types d'atelier

1.3 Classification des problèmes d'ordonnancement

Vu la variété des problèmes d'ordonnancement, une notation composée de trois champs α, β, γ est proposée en 1979 par GRAHAM et E.L. LAWLER [12], cette notation permet de classifier ces problèmes et les caractérisent d'une manière précise.

Le champ α : Ce champ décrit l'environnement des machines, et il contient deux sous champs α_1 qui désigne le type de machines utilisées et α_2 indique le nombre de machines.

$\alpha_1 \in$ {1,P,Q,R,F,J,O,FH}, la description de ces notations est décrite dans le tableau ci-dessous :

Notation	Description
1	problème à une seule machine
P	problème à machines parallèles identiques
Q	problème à machines parallèles uniformes
R	problème à machines parallèles indépendantes
F	problème Flow shop
J	problème Job shop
O	problème Open shop
FH	problème Flow shop hybride

TABLE 1.3 – Valeurs du champ α_1

Le champ β : Ce champ repésente l'ensemble des contraintes imposés sur l'exécution des tâches il peut être vide et il peut aussi contenir des contraintes comme l'indique le tableau ci-dessous.

Notation	Signification
pmtn	possibilité de la préemption de la tâche
prec,tree,chains	contraintes de précédence entre les tâches
res	l'emploi des ressources additionnelles
r_j	les temps au plus tôt diffèrent par tâche
p_j=p	temps d'exécution des tâches sont égaux à p
no-wait	les capacités d'attente sont nulles

TABLE 1.4 – Valeurs du champ β

Le champ γ : Ce champ signifie le critère à optimiser.
$\gamma \in \{C_{max}\,;\, \sum_{j=1}^{n} C_j\,;\, \sum_{j=1}^{n} U_j\,;\, \sum_{j=1}^{n} w_j C_j\,;\, L_{max}\,;\, ...\}$

Exemples :

- O3|pmtn,r_j|L_{max} signifie un ordonnancement préemptif dans un open shop à trois machines, les tâches arrivent à des instants différents et l'objectif est de minimiser le décalage maximum.

- F3|pmtn,prec |$\sum_{j=1}^{n} T_j$ signifie un ordonnancement préemptif dans un flow shop à trois machines ou il existe des contraintes de précédence entre les tâches, ce problème a pour objectif la minimisation de la somme des retards.

1.4 Diagramme de GANTT

Ce type de diagramme a été mis au point en 1910 par Henry Gantt, il est formé de deux axes orthogonaux tel que l'axe des abscisses représente le temps et l'axe des ordonnées représente les machines.
À titre d'illustration, une solution réalisable du problème F2||C_{max} est représentée dans la figure

ci-dessous.
Les durées d'exécution des tâches sont données dans le tableau suivant :

	J_1	J_2	J_3	J_4	J_5
M_1	1	2	3	2	1
M_2	2	1	4	3	2

TABLE 1.5 – Durées d'exécution du problème m=2 et n=5

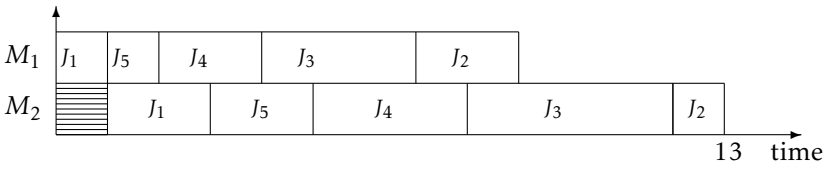


FIGURE 1.8 – Exemple d'un diagramme de Gantt

1.5 Ordonnancement multi-agents

Après les problèmes d'ordonnancement multi-objectifs traités jusqu'ici dans lesquels un seul preneur de décision est pris en considération, un nouvel air de recherche dans la théorie d'ordonnancement a été mis à l'étude : les problèmes d'ordonnancement multi-agents où plusieurs agents sont en compétition puisqu'ils partagent les mêmes ressources et chaque agent vise à minimiser un critère ne dépendant que de l'exécution de ses propres tâches.

Cette notion d'agent est déjà considérée dans l'intelligence artficielle où un agent est considéré comme une entité logicielle ou physique capable d'accomplir une mission qui lui est attribuée en coopération avec d'autres agents[6], le terme agent recouvre beaucoup de sens différents suivant les domaines d'applications, dans notre étude la notion d'agent que nous considérons est plus proche de celle utilisé en « théorie des jeux » où des agents « joueurs » prennent des décisions en choisissant des stratégies qui peuvent les mener au meilleur des gains, ces agents peuvent être un objet, un programme ou un humain.
Nous définissons dans la suite un agent par une entité autonome munie de certaines aptitudes telle que la capacité de poursuivre un but et à négocier avec d'autres agents l'utilisation des ressources partagées afin d'atteindre son objectif.

En 2003-2004 Baker et Smith [4] et Allesandro Agnetis et al [2] ont introduit pour la première fois les problèmes d'ordonnancement à deux agents, ces problèmes attirent l'attention de plusieurs chercheurs de nos jours surtout dans les ateliers manufacturiers où l'on trouve un travail récent qui traite un problème d'ordonnancement d'atelier de type Flow Shop [3] .

1.5.1 Typologie des problèmes d'ordonnancement multi-agents

Dans cette partie nous allons définir une classification des problèmes multi-agents basée sur la relation entre les sous-ensembles J^k en décrivons brièvement différents scénarios de la classification.[1]

Agents en compétition : Dans ce cas, les agents n'ont pas de tâche en commun, c'est-à-dire tous les travaux de chacun des K ensembles de tâches sont attribués seulement à un agent. Cela signifie que pour deux agents A et B on a $J^A \cap J^B = \emptyset$ et chacun d'eux a un critère f^A et f^B à minimiser, la notation de ce scénario dans le champ β est **CO** « Competing Agents ».

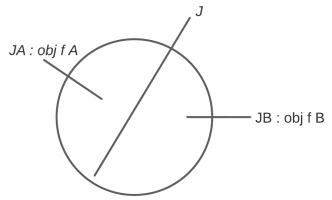


FIGURE 1.9 – Agents en compétition

Agents interférants : Dans ce cas, le K^{ime} agent dit agent global est associé à l'ensemble totale des tâches J et les $K-1$ agents restants sont appelés agents locaux et ils sont en compétition, la relation entre les sous ensembles de tâches attribués à chaque agent est donné par :

$$\mathrm{J} \supseteq \mathrm{J}^1 \supseteq \mathrm{J}^2 \supseteq ... \supseteq \mathrm{J}^{K-1}$$

La notation de ce scénario dans le champ β est **IN** « Interfering Agents ».

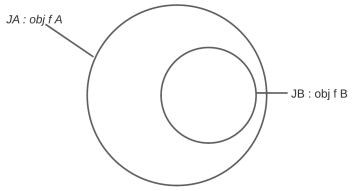


FIGURE 1.10 – Agents interférants

Multi-critère : C'est le cas classique de l'ordonnancement multicritère, c'est-à-dire dans lequel $\mathrm{J} = J^1 = J^2 = ... = J^K$

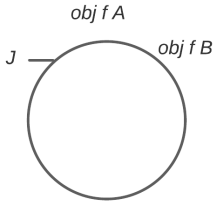


FIGURE 1.11 – Multi-critère

Non-Disjoints : Il s'agit du cas le plus général, dans lequel deux ensembles de tâches peuvent ou non se croiser donc la notation J_j^k indique un travail J_j qui n'appartient qu'à J^k.
Dans le cas de K = 2 agents, on suppose qu'un emploi dans $J^A \cap J^B$ n'a qu'un seul temps de traitement possible, quel que soit l'agent auquel il appartient cependant concernant les autres paramètres cela dépend de l'agent en question.

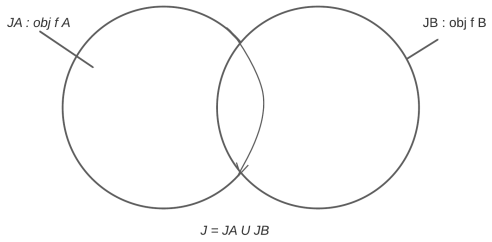


FIGURE 1.12 – Non-Disjoints

1.6 Méthodes de résolution des problèmes d'ordonnancement

Les problèmes d'ordonnancement font partie des problèmes d'optimisation combinatoire, la diversité de ces problèmes a conduit à une grande variété de techniques de résolution, dans cette partie nous passons en revue les méthodes les plus connues en les classifiant en deux grandes familles de classes : méthodes exactes et méthodes approchées.

Dans la suite et pour chaque méthode un aperçu est présenté, plus de détails ne sont donnés que pour les approches que nous utilisons dans notre étude.

1.6.1 Méthodes exactes

Ces méthodes appelées aussi complètes vue qu'elles garantissent la complétude de la résolution en donnant la solution optimale exacte.

L'inconvénient majeur de ses méthodes est l'explosion combinatoire : le temps de calcul nécessaire d'une telle méthode croit exponentiellement avec la taille du problème à résoudre alors l'efficacité de ces algorithmes n'est prometteuse que pour les problèmes de petites et moyennes tailles.

1.6.2 Méthodes approchées

Face aux difficultés rencontrées par les méthodes exactes pour résoudre différents problèmes d'optimisation de grande taille en un temps raisonnable, les méthodes approchées ont fait leur apparition, ces techniques ne fournissent pas une solution optimale mais elles permettent de trouver la meilleure approximation possible de la solution optimale en un temps de calcul raisonnable. Ces méthodes peuvent être divisées en deux catégories :

◇ **Heuristiques** ◇ **Métaheuristiques**.

1.6.2.1 Heuristiques

Le mot heuristique dérive du grec ancien eurisko qui signifie ("je trouve") et qualfié tout ce qui sert à la découverte, l'invention et à la recherche. Ces méthodes trouvent une solution approchée de qualité "raisonnable" à un problème NP-complet en un temps polynomial. Typiquement, elles sont spécifiques à un type de problème [9]. Parmis ces méthodes nous citons, les heuristiques constructives et les méthodes d'amélioration.

1.6.2.2 Métaheuristiques

Le terme métaheuristique vient des mots grecs méta qui signifie ("au delà") et de terme heuristique évoqué dans la partie précédente. Elles sont des stratégies ou des méthodes itératives permettant de guider la recherche d'une solution optimale de problèmes combinatoires NP-difficiles

[17]. Leur fonctionnement, contrairement aux heuristiques, est indépendant du problème traité. Ces méthodes sont classées en fonction du nombre de solutions qu'elles manipulent :

1. **Méthode à base de solution courante :** ont plusieurs noms comme méthodes de recherche locale ou méthodes de trajectoire. Leur mécanisme consiste à faire évoluer interactivement une solution dans l'espace de recherche vers un optimum global[16]. Parmis les méthodes de trajectoire les plus utilisées, nous trouvons :

   – **Recuit simulé** c'est une méthode basée sur la recherche locale inspirée du recuit simulé physique utilisé en métallurgie, lui-même reposant sur les lois de thermodynamique énoncées par Boltzmann. Cette méthode consiste à alterner des cycles de chauffe et de refroidissement lent à métal solide, jusqu'à atteindre son équilibre thermodynamique.

   – **La recherche tabou** La méthode de recherche tabou est une procédure itérative formalisée en 1986 par F.Glover[11], la premiére mention du terme "métaheuristique" est faite lors de la conception de cette méthode. Le principe de base de la recherche tabou est simple, à partir d'une solution initial quelconque S_0, cet algorithme engendre une succession de solutions et poursuit la recherche de solutions même lorsqu'un optimum local est rencontré en permettant des déplacements qui n'améliorent pas la solution courante. Afin d'éviter le problème de revenir sur des solutions déjà rencontrées la recherche tabou fait intervenir une liste tabou (TL), cette liste permet de mémoriser temporairement les dernières solutions visitées et d'interdire tout mouvement menant à ces solutions.
   l'algorithme suivant décrit les étapes de la recherche tabou :

Algorithm 2: La recherche tabou

Begin
Construire une solution initial S_0 ;
S $\longleftarrow S_0$;
Initialiser une liste tabou vide TL=$\emptyset$;
Repeat
(1) Générer un ensemble $N^{'} \subseteq N(s)$ de solution voisines de s ;
(2) Choisir la meilleur solution $\in N^{'}$ telle que $S^{'} \notin$ TL ;
(3) Mettre à jour la liste TL des solutions taboues ;
(4) S $\longleftarrow S^{'}$;
Until la satisfaction de la condition d'arrêt.
Retourne la meilleure solution.
End

2. **Méthode à base de population :** elles peuvet être applées aussi méthodes de recherche globale. Contrairement aux approches à solution unique, ces méthodes font évoluer simultanément un ensemble de solution dans l'espace de recherche[8]. Citons par exemple :

   – Les algorithmes génétiques qui tentent de reproduire l'évolution naturelle des individus en respectant la loi de survie énoncée par Darwin. Ces algorithmes ont été rapidement appliqués pour résoudre et avec succés les problèmes d'optimisation combinatoires en recherche opérationnelle et les problèmes d'apprentissage dans le domaine de l'intelligence artificielle, l'algorithme suivant résume les différentes étapes d'un algorithme génétique standard :

Algorithm 3: Algorithme génétique

Begin

Génération d'une population initiale de K individus ;

Repeat

(1) Évaluation de la fonction objectif de chaque individus ;

(2) Sélection des meilleurs individus ;

(3) Croisement entre 2 individus : génération 2 enfants issus de deux parents sélectionnés ;

(4) Mutation des génes de certains individus de la population ;

Until la satisfaction de la condition d'arrêt.

Retourner la meilleure solution.

End

La figure (Figure 1.9) illustre le principes de base des ces algorithmes :

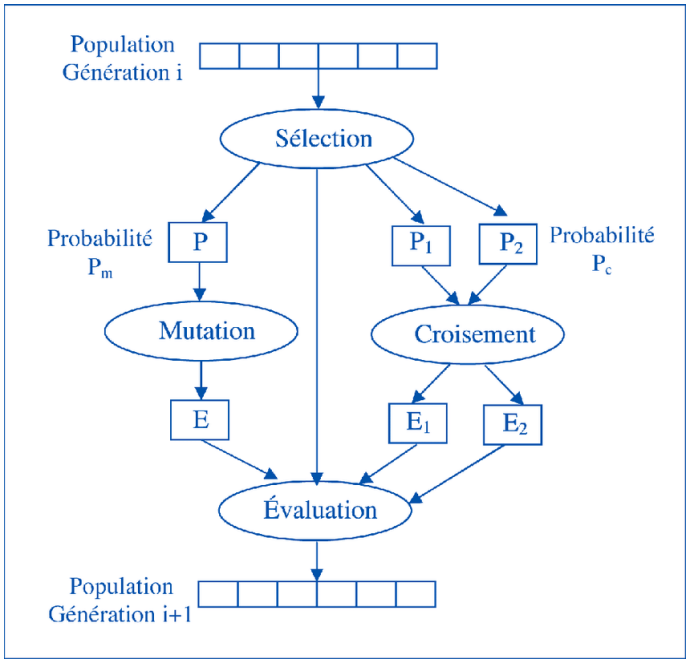


FIGURE 1.13 – Principe général des algorithmes génétiques

La figure ci-dessous représente une classification des approches de résolution :

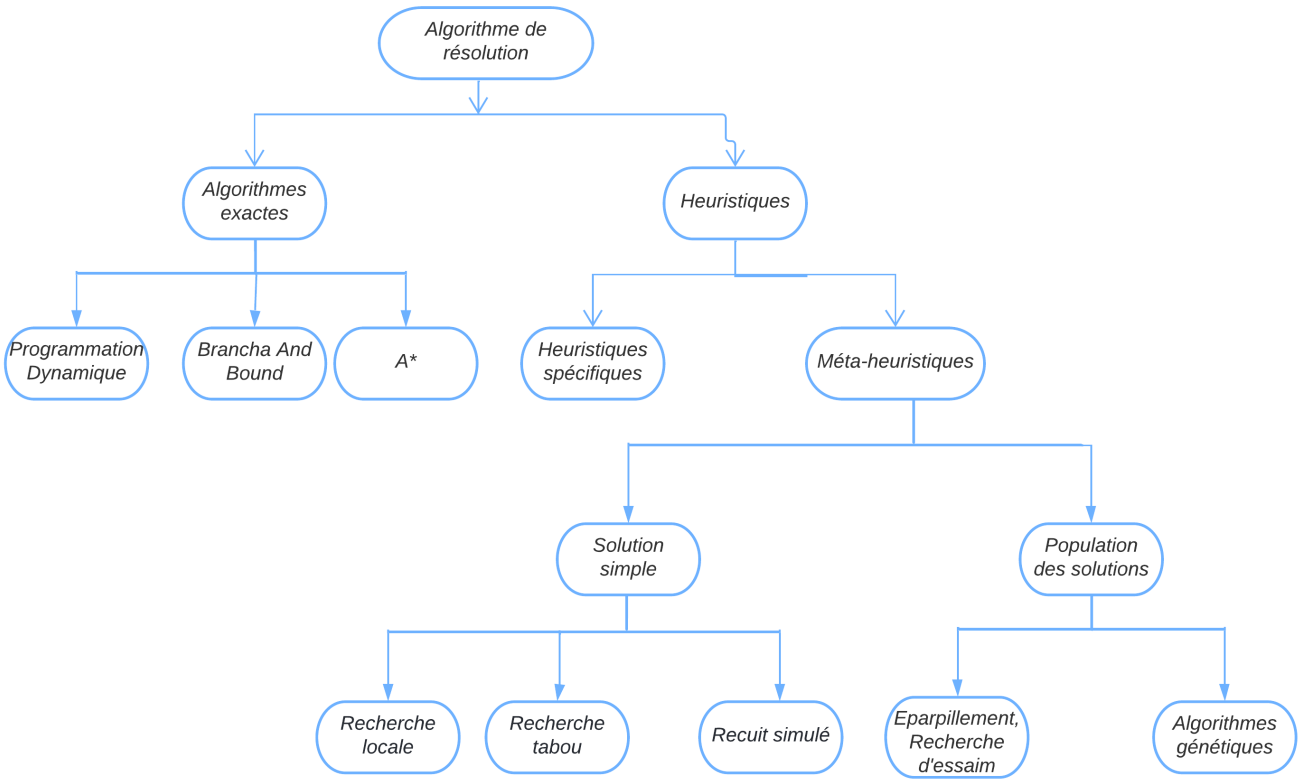


FIGURE 1.14 – Classification des approches de résolution

1.7 Complexité algorithmique

La complexité algorithmique est une notion vitale qui permet d'évaluer l'efficacité d'un algorithme au terme de temps ainsi que l'espace mémoire nécessaire pour la résolution d'un problème. Cela dépend de la difficulté du problème à résoudre et du nombre d'instructions élémentaires effectués pour trouver l'optimum en fonction de la taille du problème [10].
En général, la complexité d'un problème est la complexité du meilleur algorithme qui le résout. Alors il est essentiel de définir une classification permettant de regrouper les problèmes ayant un même niveau de difficulté par exemple si un algorithme s'exécute en temps polynomial le problème est dit facile, sinon il est difficile.

Ces problèmes peuvent être classés en quatre classes :

1. **La classe P :** Cette classe contient tous les problèmes de décision [1] relativement faciles. Autrement dit, elle regroupe les problèmes de décision qui peuvent être résolus dans un temps polynomial [2] sur une machine déterministe [3].

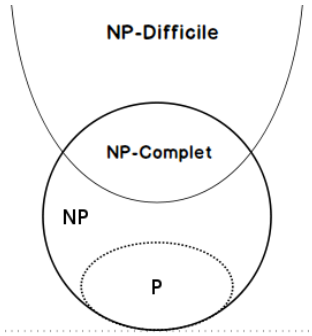
2. **La classe NP :** Cette classe contient tous les problèmes de décision qui peuvent être résolues par un algorithme non déterministe en temps polynomiale.
   En cas d' **algorithme non déterministe**, pour la même entrée, le compilateur peut produire une sortie différente dans différentes exécutions.

3. **La classe NP-complet :** Parmi l'ensemble des problèmes de décision appartenant à NP, il en existe un sous-ensemble qui contient les problèmes les plus difficiles : on les appelle les problèmes NP-complets. Un problème est NP-complet quand tous les problèmes appartenant à NP lui sont réductibles.

   Definition 5. Le concept de réduction consiste à transformer un problème en un autre afin d'utiliser la solution du deuxième problème poure résoudre le problème initiale. Soit P1 et P2 deux problèmes de décision, la réduction de P1 vers P2 notée P1 $\propto$ P2, consiste à transformer chaque instance I de problème P1 en une instance I de P2 avec un algorithme polynomial.

4. **La classe NP-difficile (NP-Hard) :** Un problème P1 est NP-difficile s'il existe un problème NP-complet P2, tel que P2 est réductible à P1 en temps polynomial.
   Un problème NP-difficile n'est pas nécessairement dans la classe NP.

La figure (Figure 2.1) montre la relation qui existe entre les différentes classes de problèmes :



P ≠ NP

FIGURE 1.15 – Relation entre les classes de problèmes

Remarque : Le problème P ≠ NP est une conjecture en informatique théorique qui est considérée par de nombreux chercheurs comme une des plus importantes conjectures du domaine.

1. Un problème de décision est un problème dont la réponse est "oui" ou "non"
2. Un algorithme est dit résolu en temps polynomial si son temps d'exécution est proportionnel à un polynôme $\mathcal{O}(P(n))$.
3. Dans un algorithme déterministe, pour une entrée particulière donnée, la machine produira toujours la même sortie passant par les mêmes opérations.

CHAPITRE 2

L'OPTIMISATION MULTI-OBJECTIFS

Dans ce chapitre nous présentons un ensemble de définitions et de concepts liés aux problèmes d'optimisation multi-objectifs, nous commençons par déterminer les différentes caractéristiques d'un problème d'optimisation pour passer ensuite à définir le problème d'optimisation multicritère et présenter les méthodes de résolutions de ce genre de problème.

2.1 Introduction

L'optimisation est une branche des mathématiques qui s'intéresse à la résolution analytique ou numérique des problèmes qui consiste à chercher la meilleure solution (optimum). En raison de la nature multi-objectif de la plupart des problèmes pratiques, l'optimisation multi-critère est considéré comme un axe de recherche très important. Cette dernière a été appliquée dans de nombreux domaines scientifiques, y compris l'économie et la logistique, où l'on cherche à optimiser simultanément plus d'une fonction objectif. Minimiser le coût et maximiser la production dans la planification de production, minimiser le coût et la matière première tout en minimisant la pollution environnementale dans le raffinage du pétrole sont des exemples de problèmes d'optimisation multi-objectifs impliquant respectivement deux et trois objectifs.

2.2 Problème d'optimisation

Un problème d'optimisation est défini par :

- **Un espace de recherche (de décision) :** est un ensemble de solutions constitué des différentes valeurs prises par les variables de décision.
- **Une ou plusieurs fonction(s)** dite objectif(s) à optimiser (minimiser ou maximiser).
- **Un ensemble de contraintes** à respecter.

Les problèmes d'optimisation sont principalement classés suivant le nombre de critères considérés, en problème d'optimisation mono-objectif ou multi-objectifs.

2.2.1 Problème d'optimisation mono-objectif :

Un problème d'optimisation est mono-objectif lorsqu'un seul objectif est considéré. Dans ce cas, la solution optimale est bien définie et c'est celle qui a le coût optimal.

Définition : un problème d'optimisation mono-objectif peut être représenté sous la forme :

$$Optimiser f(x) \tag{2.1}$$
$$s.c \quad x \in K \tag{2.2}$$

où $f_i : R \longrightarrow R$

2.2.2 Problème d'optimisation multi-objectifs :

Dans un problème d'optimisation multi-objectif, on cherche à trouver une solution réalisable de l'ensemble $\{f^1, f^2, ..., f^K\}$ des fonctions objectifs à optimiser.

Définition : un problème d'optimisation multi-objectifs peut être formulé par :

$$OptimiserF(x) = (f_1(x), f_2(x), ..., f_p(x)) \tag{2.3}$$
$$s.c\ x\ \in K \tag{2.4}$$

où $f_i : D_i \subset R^n \longrightarrow R$, $\forall$ p ≥ 2 le nombre de fonctions objectifs.

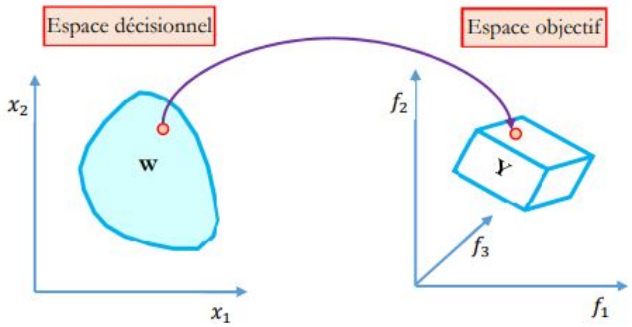


FIGURE 2.1 – problème d'optimisation multiobjectif (2 variables de décision et 3 fonctions objectifs)

2.3 Les approches d'optimisation multi-objectifs

Les approches de résolution des problèmes multiobjectifs peuvent être réparties en classes suivantes :

1. Les approches Pareto.
2. Les approches non Pareto.

2.3.1 Les approches Pareto

Ces méthodes sont les plus utilisées dans l'optimisation multi-objectifs. ils sont basés sur l'hypothèse V. Pareto [18] : « Il existe un équilibre tel que l'on ne peut pas améliorer un critère sans détériorer au moins un des autres critères ». En effet, il n'y a aucun moyen de définir "la valeur optimale" d'un problème d'optimisation multi-objectifs, il existe par conséquent, un ensemble de valeurs optimales, que nous appelons "Optimal de Pareto" selon le critère de dominance au sens de Pareto, dans ce qui suit, nous définissons tout d'abord la notion de dominance au sens de Pareto et la frontière de Pareto, Pareto optimal. Ensuite, nous présentons les méthodes évolutionnaires utilisant cette notion.

2.3.1.1 La notion de dominance au sens de pareto

Ce critère de dominance est la base des méthodes dite de pareto :

Définition : Soient $\{f_i, i \in [1,m]\}$ un ensemble de critères à minimiser, x et $x^{'}$ deux solutions de l'espace réalisable. On dira que x domine $x^{'}$ au sens de Pareto si, $\forall i \in$[1,m], $f_i(x) \leq f_i(x^{'})$, avec au moins une inégalité stricte.

Définition : x domine $x^{'}$ dans le sens du Pareto, signifie que $F(x)$ est mieux que $F(x^{'})$ pour tous les objectifs, et il y a au moins une fonction objectif pour laquelle $F(x)$ est strictement meilleure que $F(x^{'})$.

Définition : La notion de Pareto optimalité a pris ses racines dans les travaux de Edgeworth et Pareto, une solution **Pareto-optimale** est une solution efficace ou solution non dominée, autrement dit aucune autre solution de l'espace réalisable ne domine cette solution, selon le critère de dominance.

Définition : Le front de Pareto est l'ensemble des solutions Pareto optimales composé de points qui ne sont dominés par aucun point (Figure 2.2).

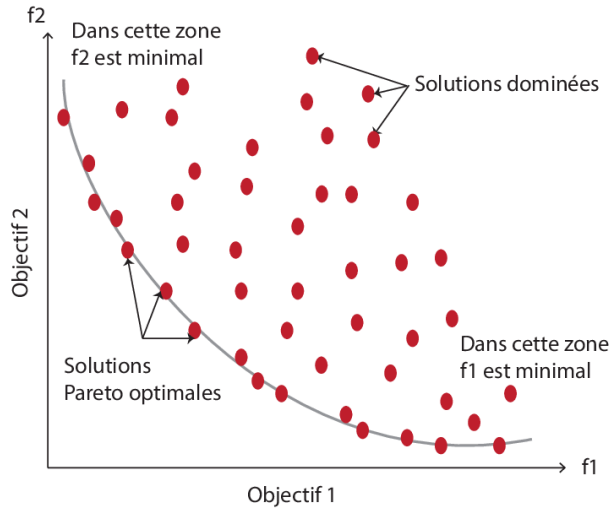


FIGURE 2.2 – Front de Pareto d'une fonction bi-objectifs

Parmis les méthodes qui se basent sur la notion de dominance, on distingue deux catégories :

- Les méthodes non élitistes : ce sont des approches qui ne conservent pas les individus Pareto-optimaux trouvés au cours du temps, ce qui rend la maintenance de la diversité sur la frontière de Pareto difficile et la convergence des solutions vers la frontière de Pareto est lente pour ces méthodes, parmis ces techniques on cite :
  Non dominate Sorting Genetic Algorithm (NSGA) : c'est un algorithme génétique de tri non dominé proposé par Deb et Srinivas[20], il utilise un processus évolutif, et la population générale est classée en sous populations basée sur l'ordre de la domination de Pareto.

- Les méthodes élitistes : ces méthodes utilise une population externe (archive),elles sont utilisées pour résoudre les difficultés des approches non-élitistes, on peut citer :
  Non dominated Sorting Genetic Algorithm II (NSGA-II) : NSGA-II est un algorithme multiobjectif introduit par Gupta et al, il et considéré dans la littérature comme l'un des algorithmes les plus performants pour la résolution d'un problème d'optimisation multicritère en utilisant l'approche de Pareto son but est d'optimiser deux ou plusieurs fonctions objectifs prédéfinies, ses principales caractéristiques peuvent être énumérées comme suit :

  1. NSGA-II adapte une stratégie élitiste qui garantie la qualité de la prochaine génération,

cette technique consiste à comparer le meilleur individu de la nouvelle génération avec celui de la précédente le meilleur individu est conservé et l'autre sera détruit.

2. Il favorise les solutions non dominées.
3. Il met en oeuvre la technique de Crowding Distance dans le but de maintenir la diversité des solutions.

2.3.2 Les approches Non-Pareto

Les approches Non-Pareto possédent un processus de recherche qui traite séparément les objectifs, d'une manière générale, elles cherchent à ramener le problème initial à un ou plusieurs problèmes mono-objectifs. De nombreuses approches ont été proposées pour résoudre ce type de problèmes, nous citons :

1. **Les méthodes agrégées :** ces méthodes reposent sur la connaissance du problème par le décideur. Il sait comment attribuer différentes importances aux critères à optimiser. On dit alors que le décideur optimise une "fonction d'utilité" $F = F(f_1, f_2, ...f_p)$.
   parmi les méthodes qui utilisent cette approche, nous pouvons citer les méthodes suivantes :

   – **Méthode d'agrégation :** elle consiste à transformer un problème multi-objectifs en un problème mono-objectif, en définissant une fonction objectif unique F comme étant la somme pondérée des différentes fonctions objectifs du problème initial. En affectant à chaque objectif un coefficient de poids qui représente l'importance relative que le décideur attribue à l'objectif : F(x) = $\sum_{i=1}^{p} \omega_i \ f_i(x)$

   ω_i représente le poids affecté à la fonction objectif f_i, $\omega_i \in [0,1]$ avec $\sum_{i=1}^{p} \omega_i$=1.

   – **Méthode ε-contrainte :** La méthode ε-contrainte aussi dite méthode du compromis, elle est basée sur la minimisation de l'objectif f_i en considérant que les autres objectifs f_j avec j ≠ i doivent être inférieurs à une valeur ε_j. En général, l'objectif choisie est celui que le décideur souhaite prioriser.

$$Minimiser f_i(x) \tag{2.5}$$

$$sc. \quad f_j \leq \varepsilon_j, \quad \forall j \neq i. \tag{2.6}$$

CHAPITRE 3

DESCRIPTION DU PROBLÈME ET NOTATIONS

Dans ce chapitre, nous étudions un problème d'ordonnancement multi-agents de type flow shop avec deux agents en compétition. Les critères d'optimisation considérés sont le Makespan pour le premier agent autrement dit le temps de fin d'éxecution de l'ensemble des tâches du premier agent, et l'énergie totale consommée pour le deuxième.

3.1 Définition et notation du problème

Dans notre problème, l'agent A et l'agent B sont associés aux sous-ensembles de tâches J_A et J_B respectivement où $J_A \cap J_B = \emptyset$ et chacun des deux agents est prédestiner à minimiser un critère propre à lui. Ces agents sont en compétition car ils partagent les mêmes ressources.

On suppose que toutes les tâches sont disponibles à l'instant zéro, et une machine ne peut traiter qu'une seule tâche à la fois. On suppose aussi qu'une machine M_i consomme δ_i énergie par unité de temps et θ_i est le taux d'augmentation de cette énergie au fil du temps, ainsi l'énergie totale consommée pour l'exécution de la tâche J_j par la machine M_i est donnée par la formule suivante : E_{ij}=$p_{ij}\delta_i$+$\theta_i s_{ij}$ où s_{ij} est la date de début de traiteemnt de la tâche J_j sur la machine M_i. Alors l'énergie totale consommée par les deux machines pour exécuter les tâches de l'agent B est définie comme suit :

$$TEC^B = \sum_{i=1}^{2} \sum_{j \in J_B} ((p_{ij} + \alpha_j s_{ij})\delta_i + \theta_i s_{ij})$$

Pour faciliter la description du problème, une notation de Graham lui est attribuée, cette notation est composée de trois champs différents :

- α : F2 signifie que l'atelier est de type flow shop composé de deux machines.
- β : p_{ij}'= p_{ij}+α_j s_{ij} , j $\in J_B$, cette contrainte assure que toutes les tâches de l'agent B sont déteriorables.
- γ : Ce champs est décomposé en deux sous-champs γ_1 γ_2 tel que :
  - γ_1 : C^A_{max} le premier critère à minimiser est la date de fin d'exécution des tâches de l'agent A.
  - γ_2 : TEC^B le deuxième critère à minimiser est l'énergie totale consommée par l'agent B.

Donc notre problème est noté par F2 | p_{ij}'= p_{ij}+α_j s_{ij} : j $\in J_B$ | C^A_{max} : TEC^B.

À titre d'illustration, nous présentons dans ce qui suit un exemple du problème F2 |p_{ij}'= p_{ij}+α_j s_{ij} : j $\in J_B$| C^A_{max} :TEC^B.

3.2 Critères

Dans notre problème, nous rechrechons un ordonnancement réalisable qui minimise simultanément deux critères. Le premier est Makespan pour le peremier agent, le second est l'energie totale consommée pour le deuxième.

Makespan de l'agent A(C^A_{max}) : Le makespan est un critère d'optimisation, représente la date de fin d'exécution de la dernière opération qui sort de l'atelier. Il est particulièrement intéressant car il permet de déterminer les séquences sur lesquelles tout retard a des conséquence sur le temps de réalisation des tâches.
Le Makespan de l'agent A est défini par la formule suivante :

$$min\ C^A_{max} = min\ \max_{j \in J_B} \{C^{(j)}_{max}\} = min\ \max_{j \in J_B} \{\sum_{i=1}^{2} p_{ij} x_{ij}\}.$$

L'énergie totale consommée par l'agent B($\mathbf{TEC}^B$) : La minimisation de l'énergie totale consommée revient à minimiser l'énergie utilisée pour éxecuter chaque tâche par unité de temps avec un taux d'augmentation d'énergie.
L'energie totale consommée par l'agent B est défini par l'equation suivante :

$$min\ TEC^B = min\ \sum_{i=1}^{2} E_i = \sum_{i=1}^{2} (\sum_{j \in J_B} (p^{'}_{ij} \delta_i + \theta_i s_{ij}))$$

Avec : $p^{'}_{ij} = p^0_{ij} + \alpha_j s_{ij}$, $\forall$ j $\in J_B$.

3.3 Les contraintes considérées

Dans notre problème, les tâches sont ordonnancées sur les deux machines selon l'ordre (M_1, M_2), où l'objectif est de trouver les séquences de compromis qui puissent satisfaire l'objectif des deux agents : minimiser la date de fin de traitement de l'ensemble des tâches pour le premier agent "agent A" et l'énergie totale consommée pour le deuxième "agent B", la préemption des tâches n'est pas autorisée et il n'y a pas de contraintes de précédence entre les tâches. On suppose que les tâches de l'agent B sont soumises à des contraintes de détérioration où chaque tâche possède un temps de traitement variable, on considère que le temps d'exécution d'une tâche J_j est une fonction linéaire croissante en fonction de la date du début de son exécution donnée par la formule $p_{ij}{}^{'} = p_{ij} + \alpha_j\ s_{ij}$ où $p_{ij}{}^{'}$, p_{ij} , α_j et s_{ij} représentent respectivement le temps d'exécution actuel, le temps d'exécution initial, le taux de détérioration et le début d'exécution de la tâche J_j.

Exemple : Considérons le problème à cheminement unique avec m=2 et n=5, les durées d'éxécution des tâches sur les deux machines ainsi que la consommation d'énergie par unité de temps δ_i et le taux d'augmentation d'énergie θ_i de chaque machine sont données dans les tableaux cidessous :

Sachant que $J_A = \{J_1, J_3, J_5\}$ et $J_B = \{J_2, J_4\}$ sont l'ensemble des tâches de l'agent A et l'agent B respectivement alors une solution réalisable est représentée par le diagramme de Gantt dans la figure 3.1 avec C^A_{max}=9 et TEC^B=21.62.

J_j	J_1	J_2	J_3	J_4	J_5
M_1	1	2	3	2	1
M_2	2	1	4	3	2

TABLE 3.1 – Temps opératoires des cinq tâches

	M_1	M_2
δ_i	2	1
θ_i	0.1	0.2

TABLE 3.2 – Données de l'exemple

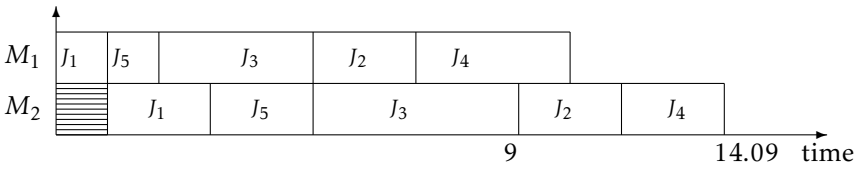


FIGURE 3.1 – Solution réalisable du problème représentée par le diagramme de Gantt

3.4 Ordonnancement et la gestion d'énergie

En Juin 2021, le chiffre de 7.8 milliards d'habitants sur notre planète avait été franchi, et à ce jour on est à 7.9 milliards de personnes sur terre et d'après les dernières projections l'ONU prévoit une augmentation de 2 milliards de personnes au cours des trente prochaines années, où la population devrait atteindre les 9.7 milliards d'individus en 2050 et un nombre proche de 11 milliards d'individus en 2100.

En raison de cette croissance démographique effrénée et la croissance économique la consommation énergétique mondiale est en croissance et devrait continuer d'augmenter dans les décennies à venir. Après un été 2021 historique toutes les principales sources d'énergie ont été concernées par une augmentation considérable des tarifs.
La figure suivante représente la consommation énergitique selon la source depuis l'an 2010 ainsi qu'une projection pour les décennies à venir.

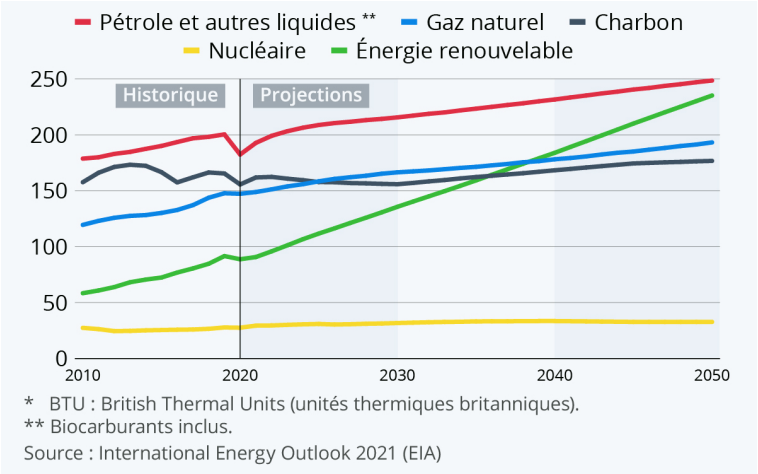


FIGURE 3.2 – Consommation mondiale de l'énergie primaire par source depuis 2010 en BTU*

L'énergie est divisée en deux catégories : renouvelable et non renouvelable, elle est dite renouvlable si elle est inépuisable il s'agit de sources d'énergie qui se renouvellent rapidement lors de son utilisation telles que l'énergie solaire et hydraulique etc ..., quant à l'énergie non renouvlable ses stocks s'épuisent lors d'utilisation car elles se renouvellent moins vite que son utilisation à l'instar du pétrole et le Gaz ainsi que l'énergie nucléaire qui est aussi considérée source d'énergie non renouvlable.

La nature d'énergie et la quantité consommée dépends du domaine d'activité, on distingue plusieurs secteurs consommant de l'énergie tels que :

— Le secteur tertiaire : transports .

— Le secteur résidentiel : électroménager, climatisation, cuisson.

— Le secteur industriel : construction et exploitation minière.

3.4.1 Gestion de l'énergie

Le coût croissant de l'énergie est l'un des facteurs les plus importants liés au coût de la production dans le domaine industriel, c'est pour cela qu'on envisage une démarche d'optimisation énergétique en adaptant différentes stratégies [19] telles que :

— Investissement dans de nouvelles machines plus économiques en terme de consommation énergétique.

— Coupures d'énergie pour réduire les pics de demande d'énergie.

— Prévisions des charges de l'énergie pour réduire les coûts.

La gestion d'énergie est un problème essentiel en particulier dans l'industrie, plusieurs problèmes d'ordonnancement sous contraintes d'énergie ont été étudié, on peut citer le problème à une seule machine minimisant l'énergie totale consommée et le retard total.

CHAPITRE 4

MÉTHODES DE RÉSOLUTION

Dans ce chapitre, nous proposons une modélisation mathématique ainsi que trois méta-heuristiques. Ces méta-heuristiques sont respectivement la recherche tabou, le recuit simulé et NSGA-II.

4.1 Formulation mathématique

4.1.1 L'approche ϵ-contrainte

Dans un prmeier temps, nous considérons l'approche ϵ-contrainte présentée dans le premier chapitre, il s'agit de chercher une solution minimisant le critère de l'agent A tout en bornant supérieurement la valeur de la fonction objectif de l'agent B, le problème sera noté donc :

$$P1 \ : \ F2|p_{ij}^{'} = p_{ij} + \alpha_j s_{ij} : j \in J_B | C_{max}^A \ : \ TEC^B \leq Q.$$
$$P2 \ : \ F2|p_{ij}^{'} = p_{ij} + \alpha_j s_{ij} : j \in J_B | C_{max}^A \leq Q \ : \ TEC^B.$$

Donc deux modèles mathématiques vont être présentés.

4.1.2 Paramètres et indices

Les paramètres utilisés signifient ce qui suit :

- n : nombre de tâches.
- m : nombre de machines.
- j : indice de tâches , où j ∈ {1,...,n}.
- i : indice de machines, où i ∈ {1,2}.
- k : indice de position, où k ∈ {1,...,n}.
- p_{ij} : temps d'éxécution de la tâche j sur la machine M_i.
- c_{ik} : date de fin d'éxécution de la tâche en position k sur la machine M_i.
- s_{ik} : temps de début d'éxécution de la tâche en position k sur la machine M_i.
- α_j : taux de détérioration de la tâche j.
- δ_i : la consommation d'énergie par unité de temps dans M_i.
- θ_i : le taux d'augmentation d'énergie de la machine M_i.
- Q : une borne supérieure pour l'objectif mis en contrainte.
- M : Constante suffisamment grande.
- C_{max}^A : le makespan de l'agent A.
- TEC^B : l'énergie totale consommé par l'agent B.

4.1.3 Variables de décision

– $x_{jk} = \begin{cases} 1, & \text{si la tâche } J_j \text{ se trouve à la position k dans la séquence} \\ 0, & \text{sinon.} \end{cases}$

4.1.4 Modèle mathématique

Premier modèle : Pour le problème F2 $|p_{ij}' = p_{ij} + \alpha_j\, s_{ij} : \mathrm{j} \in \mathrm{J}_B \mid C^A_{max}$: $TEC^B \leq Q$.

$$minC^A_{max} \tag{4.1}$$

$$\sum_{j=1}^{n} x_{jk} = 1, \quad k \in \{1,...,n\} \tag{4.2}$$

$$\sum_{k=1}^{n} x_{jk} = 1, \quad j \in \{1,...,n\} \tag{4.3}$$

$$c_{2k} - M(1 - \sum_{j \in J_A} x_{jk}) \leq C^A_{max} \quad k \in \{1,...,n\} \tag{4.4}$$

$$c_{2k} \geq c_{1k} + \sum_{j=1}^{n} (p_{2j} x_{jk}), \quad k \in \{1,...,n\} \tag{4.5}$$

$$c_{1k} = c_{1k-1} + \sum_{j=1}^{n} (p_{1j} x_{jk}), \quad k \in \{2,...,n\} \tag{4.6}$$

$$c_{2k} \geq c_{2k-1} + \sum_{j=1}^{n} (p_{2j} x_{jk}), \quad k \in \{2,...,n\} \tag{4.7}$$

$$c_{1k} \geq \sum_{j=1}^{n} (p_{1j} x_{jk}), \quad k \in \{1,...,n\} \tag{4.8}$$

$$s_{1k+1} \geq s_{1k} + \sum_{j=1}^{n} (p_{1j} x_{jk}), \quad k \in \{1,...,n\} \tag{4.9}$$

$$s_{11} = 0 \tag{4.10}$$

$$s_{21} \geq s_{11} + \sum_{j=1}^{n} (p_{1j} x_{j1}) \tag{4.11}$$

$$s_{2k} \geq s_{1k} + \sum_{j=1}^{n} (p_{1j} x_{jk}), \quad k \in \{2,...,n\} \tag{4.12}$$

$$s_{2k+1} \geq s_{2k} + \sum_{j=1}^{n} (p_{2j} x_{jk}) \quad k \in \{1,...,n-1\} \tag{4.13}$$

$$TEC^B \leq Q \tag{4.14}$$

$$c_{ik} \geq 0, s_{ik} \geq 0, i \in \{1,2\}, k \in \{1,...,n\} \tag{4.15}$$

$$x_{jk} \in \{0,1\}, \quad j \in \{1,...,n\}, k \in \{1,...,n\} \tag{4.16}$$

Deuxième modèle : Pour le problème F2 $|p_{ij}' = p_{ij} + \alpha_j\, s_{ij} : j \in J_B \mid C^A_{max} \leq Q : \text{TEC}^B$.

$$min\ TEC^B = \sum_{i=1}^{2}(\sum_{j\in J_B}(p'_{ij}\delta_i + \theta_i s_{ij})) \tag{4.17}$$

$$\sum_{j=1}^{n} x_{jk} = 1, \quad k \in \{1,...,n\} \tag{4.18}$$

$$\sum_{k=1}^{n} x_{jk} = 1, \quad j \in \{1,...,n\} \tag{4.19}$$

$$c_{2k} - M(1 - \sum_{j\in J_A} x_{jk}) \leq C^A_{max} \quad k \in \{1,...,n\} \tag{4.20}$$

$$c_{2k} \geq c_{1k} + \sum_{j=1}^{n}(p_{2j}x_{jk}), \quad k \in \{1,...,n\} \tag{4.21}$$

$$c_{1k} = c_{1k-1} + \sum_{j=1}^{n}(p_{1j}x_{jk}), \quad k \in \{2,...,n\} \tag{4.22}$$

$$c_{2k} \geq c_{2k-1} + \sum_{j=1}^{n}(p_{2j}x_{jk}), \quad k \in \{2,...,n\} \tag{4.23}$$

$$c_{1k} \geq \sum_{j=1}^{n}(p_{1j}x_{jk}), \quad k \in \{1,...,n\} \tag{4.24}$$

$$s_{1k+1} \geq s_{1k} + \sum_{j=1}^{n}(p_{1j}x_{jk}), \quad k \in \{1,...,n\} \tag{4.25}$$

$$s_{11} = 0 \tag{4.26}$$

$$s_{21} \geq s_{11} + \sum_{j=1}^{n}(p_{1j}x_{j1}) \tag{4.27}$$

$$s_{2k} \geq s_{1k} + \sum_{j=1}^{n}(p_{1j}x_{jk}), \quad k \in \{2,...,n\} \tag{4.28}$$

$$s_{2k+1} \geq s_{2k} + \sum_{j=1}^{n}(p_{2j}x_{jk}) \quad k \in \{1,...,n-1\} \tag{4.29}$$

$$C^A_{max} \leq Q \tag{4.30}$$

$$c_{ik} \geq 0, s_{ik} \geq 0, i \in \{1,2\}, k \in \{1,...,n\} \tag{4.31}$$

$$x_{jk} \in \{0,1\}, \quad j \in \{1,...,n\}, k \in \{1,...,n\} \tag{4.32}$$

4.1.5 Signification des contraintes

Pour les deux modèles l'ensemble des contraintes est presque le même, donc on s'est contenté d'expliquer les contraintes du premier modèle.

- La fonction C^A_{max} représente la première fonction objectif de notre problème qui consiste à minimiser le temps de fin d'exécution des tâches du premier agent.

- Les contraintes (4.2) et (4.3) garantissent que chaque position contient une seule tâche et q'une tâche occupe une seule position.

- La contrainte (4.4) calcule le makespan de l'agent A.

- La contrainte (4.5) garantit que la deuxième opération de la tâche en position k ne commence l'éxécution qu'après la fin d'éxécution de la première opération.

- La contrainte (4.6) et (4.7) précisent que le traitement d'une tâche en position k commence aprés la fin d'éxécution de la tâche en position k-1 sur la m*ê*me machine.

- La contrainte (4.8) garantit que la date de fin d'exécution de la tâche en position une doit *ê*tre supériuer ou égal à son temps de traitement .

- La contrainte (4.9) calcule la date de début du job en position k sur la première machine.

- La contrainte (4.10) assure que la date de début du job en première position sur la première machine est égal à zéro.

- La contrainte (4.11) calcule la date de début du job en première position sur la machine M_2.

- La contrainte (4.12) assure q'un job doit finir son opération sur la machine M_1 avant de pouvoir démarrer son opération sur la machine suivante M_2.

- La contrainte (4.13) assure que le job en position k+1 ne puisse commencer son traitement sur la machine M_2 tant que le job en position k n'a pas encore terminé son traitement sur cette même machine.

- La contrainte (4.14) assure que TEC^B est inféreiur ou égal une valeur donnée.

- La contrainte (4.15) garantit que les variables doivent *ê*tre positives.

- La contrainte (4.16) garantit que les variables de décisions correspondantes sont binaires.

4.2 Méta-heuristiques

La difficulté rencontrée par les heuristiques pour avoir une bonne solution réalisable pour des problèmes d'optimisation difficiles a donnée naissance à une nouvelle famille d'algorithmes : les méta-heuristiques . Ces algorithmes visent à résoudre des problèmes d'optimisation difficiles pour lesquels on ne connait pas de méthode classique plus efficace.

Nous présentons dans la suite deux méta-heuristiques de recherche locale la recherche tabou et le recuit simulé, ces méthodes construisent une trajectoire dans l'espace de recherche en tentant de se diriger vers des solutions optimales, et NSGAII qui est une méthode à base de population de solutions.

4.2.1 Recherche tabou

La recherche tabou est une métaheuristique de recherche locale utilisée pour résoudre des problèmes complexes ou de grande taille.

Dans cette section un algorithme de recherche tabou est proposé pour résoudre le problème P1 et P2.

Pour mettre en oeuvre l'algorithme de recherche tabou, on doit disposer de :

- La solution initiale S_0.
- Les voisinages, qui décrivent les solutions atteignables à partir d'une solution par un seul mouvement non-tabou.
- Le critère d'arrêt.
- La liste tabou et sa taille.

– Le critère d'aspiration.

1. **Solution initiale :** une bonne solution non optimale peut être générer rapidement à l'aide des heursitiques où d'une manière aléatoire, dans notre étude on propose deux algorithmes pour l'obtention de la solution initiale et une troisième solution sera générée aléatoirement. :

   – **Algorithme H1 :** pour générer la solution initiale de cette méthode, on divise la séquence en deux parties où les tâches de l'agent A sont ordonnancées selon la règle de Johnson suivies des tâches de l'agent B dans leurs ordre donné.
   – **Algorithme H2 :** dans cette algorithme la séquence initiale est composée des tâches de l'agent A ordonnancées en alternance avec les tâches de l'agent B.

2. **Génération du voisinage :** à chaque itération de l'algorithme un ensemble de solutions voisines de la solution courante est engendré, une solution est représenté dans notre étude par une liste de n éléments "tâches" donc trouver une solution voisine consiste à faire l'échange de deux positions **pos** et **pos'** de deux tâches différentes pour passer de la solution courante **S** à la meilleure solution voisine **S'** en s'assurant que le mouvement (pos , pos') ne figure pas dans la liste tabou.

   Pour les petites instances n $\in$ {10 , 30 , 50} on explore tout les voisins possibles d'une solution candidate pour trouver le meilleur mouvement non tabou à effectuer. Cependant cette manière de procéder peut se révéler très coûteuse en terme de temps d'exécution et de mémoire pour des instances plus grandes, le voisinage ne sera pas totalement généré durant chaque itération. donc pour y remédier un sous ensemble de voisins de cardinal inférieur à $\frac{n(n-1)}{2}$ est engendré d'une manière aléatoire.

3. **Critère d'arrêt :** cet algorithme calcule une nouvelle itération jusqu'à ce qu'un critère d'arrêt soit vérifié, en général le critère d'arrêt est fixé à un certain nombre d'itération à ne pas dépasser ou un nombre d'itération sans amélioration de solution.

4. **Liste tabou TL :** cette liste permet d'éviter que l'algorithme boucle, comme il est coûteux en temps de calcul de vérifier si une solution figure dans la liste tabou, alors la liste n'enregistre pas toutes les solutions déjà visitée mais uniquement les mouvements effectués, donc dés son exécution, le mouvement (pos , pos') est inséré dans la liste TL.

5. **Taille de liste tabou :** c'est considérée comme l'un des facteurs de succés d'une bonne implémentation, une relation entre la taille de la liste tabou et la taille du voisinage fut l'objet de nombreux articles, il s'agit de fixer la taille de la liste tabou à la racine carrée de la taille du voisinage[5], et c'est la procédure adoptée pour notre étude.

6. **Critère d'aspiration :** dans notre implémentation ce critère consiste à révoquer le statu tabou associé à un mouvement particulièrement utile tant qu'il conduit à une amélioration de la solution courante.

4.2.1.1 Description des étapes de l'algorithme de recherche tabou :

L'algorithme de recherche tabou commence par générer une solution initiale S_0 qui est une variable d'entrée pour la quelle nous calculons les deux objectifs, on va utiliser cette solution pour générer son voisinage pour ensuite parcourir l'ensemble des voisins et choisir le meilleur comme solution courante, ensuite on enregistre le mouvement qui nous a permis d'atteindre cette solution voisine dans la liste tabou **TL** ce qui permet de réduire les risques de cycles, le processus s'arrête lorsque le nombre d'itérations réalisé devient égal à la variable d'entrée $iter_{max}$. L'algorithme retourne la meilleure solution trouvé, la figure ci-dessous présente le déroulement d'une recherche tabou :

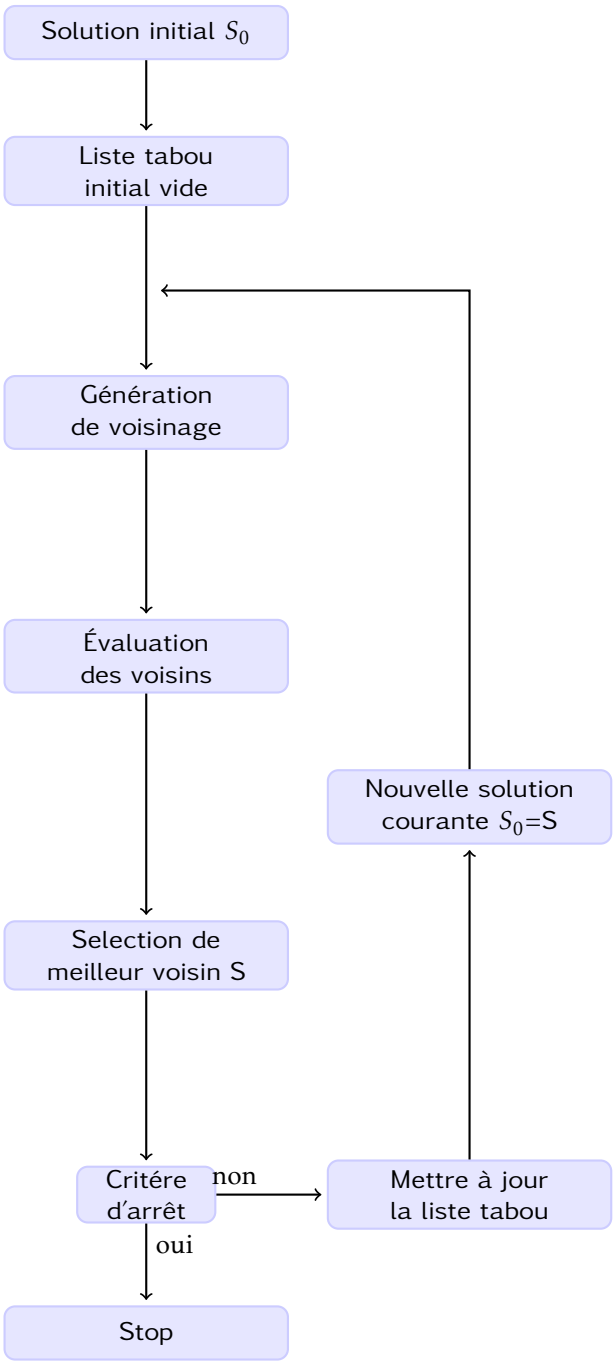


FIGURE 4.1 – Organigramme de l'algorithme recherche tabou

L'algorithme suivant décrit le déroulement de la recherche tabou pour le problème F2 $|p_{ij}\text{'}= p_{ij}+\alpha_j\ s_{ij} : \mathrm{j} \in \mathrm{J}_B \mid C^A_{max}\ :\ TEC^B \leq Q$.

Algorithm 4: Algorithme de recherche tabou pour P1

Input : Solution initiale S_0, sizeTL, $iter_{max}$, Q :borne supérieur de l'énergie consommé par l'agent B.
Begin
Initialiser la solution courante à S_0 ;
Evaluer la solution courante et poser $C^A_{max}(S_0)$ la valeur du makespan pour la solution courante ;
$C_{best} = C^A_{max}(S_0)$;
$E_{best} = TEC^B(S_0)$;
iter = 0 , TL = $\emptyset$;
while iter < $iter_{max}$ **do**
  Déterminer une solution voisine S' en effectuant un mouvement non tabou.
  if ($C^A_{max}(S') < \mathrm{C}_{best}$ **and** $\mathrm{TEC}^B(S') \leq Q$) :
    $C_{best} = C^A_{max}(S')$;
  Endif
  Mettre à jour la liste tabou ;
  iter += 1 ;
Endwhile
Return meilleure solution , C_{best}

4.2.2 Recuit Simulé

Le recuit simulé introduit en 1983 par Kirkpatrick , Gelat et vecchi[15] démarre par une solution initiale aléatoire, ou obtenue par une heuristique, et explore l'espace de solutions en effectuant des modifications mineures à la solution initiale. Si après modification, la nouvelle solution obtenue est meilleure que la précédente elle est retenue. Si ce n'est pas le cas, elle peut être retenue suivant une probabilité $P = EXP(-\Delta E/T)$, où ΔE représente la détérioration de la nouvelle solution et T un paramètre inversement proportionnel au nombre d'itérations.

Pour appliquer le recuit simulé à notre problème, nous devons disposer de :

- La solution initiale.
- La température initiale.
- Un coefficient de refroidissement α.
- Une probablité $\beta \in [0,1]$ d'accepter ou refuser les solutions.

1. **Solution initiale :** il existe plusieurs manières pour la génération de la solution initiale : elle peut être prise au hasard dans l'ensemble des solutions réalisables, à l'aide d'une heuristique.

2. **Température initiale :** elle doit être choisi suffisamment élevée pour atteindre rapidement l'équilibre.

3. **Génération du voisinage :** nous effectuons des modifications élémentaires sur la solution courante à chaque itération de l'algorithme tout en gardant la température constante cela pour se déplacer d'une solution courante à une solution voisine, la nouvelle solution obtenue permet soit d'améliorer notre critère à optimiser, soit de le dégrader. Dans notre algorithme nous utilisons une permutation aléatoire donc il suffit de permuter deux positions

choisies aléatoirement pour le déplacement.

4. **Le critére de Métropolis :** si la solution voisine est moins bonne que la solution courante, alors elle est acceptée avec une probabilité P calculée en fonction de la température T et de l'énergie E.

$$P = EXP(-\Delta E/T) \quad avec \quad \Delta E = E(solution\ voisine) - E(solution\ prcdente).$$

   Cette règle d'acceptation est appelée critère de Metropolis.

5. **Variation de la température :** pendant le processus la température décroit lentement, en suivant une loi de décroissance : $T_{i+1} = T_i \ \alpha$ (en général $\alpha = 0.99$).

6. **Critère d'arrêt :** l'algorithme s'arrête lorsque la température atteint la température finale ou une valeure presque nulle.

4.2.2.1 Description des étapes de l'algorithme de Recuit Simulé :

Le principe consiste à générer successivement des voisins qui améliorent strictement la fonction objectif à partir d'une solution initiale S_0 et d'une température de départ T_0 qui abaissera pendant le processus jusqu'à atteindre une température finale ou un état d'équilibre qui correspond à une énergie minimale, cette énergie correspond à la fonction objectif à optimiser et l'état d'équilibre à la solution du problème.

L'algorithme suivant décrit le déroulement du recuit simulé pour le problème F2 $|p_{ij}' = p_{ij} + \alpha_j$ $s_{ij} : j \in J_B \mid TEC^B \ : \ C^A_{max} \leq Q$.

Algorithm 5: Algorithme Recuit Simulé pour P2

Input : Solution initiale S_0, Température initiale T_0, Température finale T_f, α, β.
Begin
Initialiser la solution courante à S_0 ;
Initialiser la température courante à T_0 Evaluer la solution courante par la fonction d'énrgie ;
while T > T_f **do** :
  Générer une solution $S' \in$ N(S) aléatoirement ;
  Evaluer la solution voisine ;
  Calculer $\Delta E = TEC^B(S')$ -TEC^B(S) ;
  If ($\Delta E < 0$)and ($C^A_{max}(s')$< Q) :
    S $\longleftarrow S'$;
  else :
    If (β < EXP(-ΔE/T)) and ($C^A_{max}(s')$< Q) :
      S $\longleftarrow S'$;
    else :
      S' est rejetée ;
    EndIf ;
  EndIf ;
  T $\longleftarrow$ T α ;
Endwhile

4.2.3 Algorithme génétique : NSGA2

Les algorithmes génétiques sont des algorithmes d'optimisation s'appuyant sur des mécanismes dérivés de la génétique, pour résoudre un problème à l'aide d'un algorithme génétique, un ensemble de solutions possibles est généré aléatoirement où par des méthodes spécifiques au problème donné cet ensemble est appelé population d'individus où population de chromosomes, ces individus sont constitués d'un ensemble d'éléments appelés gènes.

L'algorithme génétique fut développé en 1975 par Holland se base sur le principe de survie des plus forts, le processus de génération de nouveaux individus s'effectue par les trois opérateurs : selection, croisement et mutation, parmi ces algorithmes on trouve l'algorithme de tri non dominée qui est utilisé pour la résolution de notre problème.

On implémente le NSGA-II pour résoudre notre problème d'ordonnancement d'atelier de type flowshop avec deux agents en compétition minimisant la date de fin de traitement de l'ensemble des tâches pour le premier agent et l'énergie totale consommée pour le deuxième, dans ce qui suit les solutions candidates sont appelées individus, et l'ensemble de ces solutions constitue une population où chaque individu est représenté par un chromosome et une génération est une itération de notre algorithme.

Les principaux composants de cet algorithme sont :

- Un principe de codage de l'élément de population.
- Un mécanisme de génération de la population initiale.
- Une fonction fitness à optimiser.
- Un opérateur de croisement et un autre de mutation.
- Taille de la population, un critère d'arrêt, les probabilités d'application des opérateurs de croisement et de mutation.

Adaptation de NSGA-II à notre problème :

4.2.3.1 Codage

Le choix du codage est le premier pas dans l'implémentation de notre algorithme, on considère un n-vecteur où un individu est représenté, tel que la k^{ime} composante contient le numéro de la tâche exécuté dans la position k, Soit l'exemple avec n=6. Les travaux sont notés $\{J_1,...,J_5,J_6\}$, le codage d'une solution possible est présenté ci-dessous :

Codage du chromosome :	2	6	3	1	5	4

4.2.3.2 Population initiale

Par analogie avec la diversité génétique des populations on peut assumer que plus la population initiale à faire évoluer sera hétérogène, moins on aura de risque de manquer l'optimum global. Cette hétérogénéité peut être générer d'une façon totalement aléatoire où en utilisant des heuristiques, pour notre cas la population initiale P_0 est générée aléatoirement.

4.2.3.3 Fonction d'évaluation

Fitness est une fonction qui sera optimisé par NSGA-II, dans notre cas on évalue chaque individu en calculant sa fonction fitness : la date de fin de traitement des tâches associées à l'agent A et l'énergie totale consommée en réalisant les tâches de l'agent B.

4.2.3.4 Sélection

Le mécanisme de sélection assure le fait que l'individu le mieux adapté à son environnement aura plus de chances à se reproduire. Ainsi cet opérateur nous permet de déterminer les individus parents qui vont contribuer à la génération suivante, dans notre algorithme nous avons adopté deux

principes de sélection :

- **Non domination au sens de Pareto :** Une solution S_i domine S_j si et seulement si $F_1(S_i) \leq F_1(S_j)$ et $F_2(S_i) < F_2(S_j)$, où F_1 correspond au C^A_{max} et F_2 correspond au TEC^B. Toutes les chromosomes de la population vont être trier en suivant cette règle, deux solutions appartiennent au même front pareto s'il ne y a pas de dominance entre elles.

- **Crowding distance :** La technique "crowding distance" est un opérateur de sélection utilisé par NSGA-II pour estimer la densité au voisinage d'une solution i dans l'espace de recherche. Cet opérateur calcule la distance moyenne sur chaque objectif entre les deux points les plus proches de la solution i, Sachant que :

f_i : correspond à la i^{me} fonction objectif
f_m^{max} : le maximum de la m^{me} fonction objectif dans la population courante.
f_m^{min} : le minimum de la m^{me} fonction objectif dans la population courante.
L'algorithme suivant est utilisé pour calculer le Crowding-Distance de chaque point du front :

Algorithm 6: Algorithme de Crowding Distance

Input : N nombre de solutions du même front ;
Begin

1. For $i\ in\ 1...N$:
2. End For ;
3. $dist_i = 0$;
4. For $m\ in\ 1...2$:
5. $Trier(f, m)$;
6. $dist_1 = dist_N = \infty$;
7. For $i\ in\ 2...N-1$:
8. $dist_i = dist_i + \frac{dist^m_{i+1} - dist^m_{i-1}}{f^{max}_m - f^{min}_m}$;
9. End For ;
10. End For ;

End

4.2.3.5 Mutation

L'opérateur de mutation modifie de manière aléatoire un gène ou plus d'un chromosome, ce mécanisme assure une exploration totale du domaine de recherche, dans notre cas l'opérateur de mutation utilisé échange le contenu de deux gènes choisis au hasard comme le montre la figure ci-dessous :

FIGURE 4.2 – Opération de mutation

Les étapes de mutation sont données par l'algorithme suivant :

Algorithm 7: Algorithme de mutation par échange

Input : chromosome ; k , j entiers ∈ [1,n] ; P_{mut} probabilité de mutation
Begin

1. Choisir aléatoirement un individu i ;
2. Générer une probabilité P ∈ [0,1] aléatoirement ;
3. **If** p ≤P_{mut} :
4. Choisir deux positions k et j dans [1,n] aléatoirement ;
5. Permuter les tâches $J_{[k]}$ et $J_{[j]}$;
6. **Endif**

Return : chromosome muté ;
End

4.2.3.6 Croisement

Cet opérateur a pour but d'enrichir la diversité de la population en manipulant la structure des chromosomes afin de générer de nouveaux individus appelés enfants qui sont encore plus performants, dans notre algorithme nous utilisons le cas élémentaire de croisement à un point dit point de croisement ou point de coupure comme le montre la figure ci-dessous :

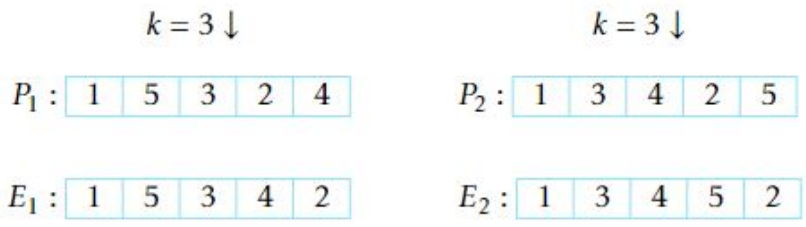


FIGURE 4.3 – Opération de croisement

L'algorithme de l'opérateur de croisement utilisé dans notre méthode est donné ci-dessous.

Algorithm 8: Algorithme de croisement

Input : liste des parents ; P_{crois} probabilité de croisement
Begin

1. Sélectionner deux parents P_1 et P_2 pour générer deux enfants E_1 et E_2 ;
2. Générer une probabilité P $\in$ [0,1] aléatoirement ;
3. **If** p $\leq$P$_{crois}$:
4. Soit t = nombre de tâches / 2 ;
5. Générer aléatoirement une sequence de t positions
6. **For** toute position k dans la séquence précédente :
7. Transmettre la tâche qui se trouve au point k du P_1 resp(P_2) à E_2 resp(E_1) .
8. Compléter les cases vides de E_1 resp(E_2) par les composantes du parent P_1 resp(P_2) en allant du début vers la fin et en ne prenant que les éléments non encore copiés ;
9. **End For** ;
10. **Endif** ;

Return : E_1 et E_2 ;
End

4.2.3.7 Critère d'arrêt

Notre algorithme s'arrête après un nombre de générations prédéfini noté itermax.

Description des étapes de l'algorithme évolutionnaire NSGA-II :

L'algorithme NSGA-II commence par une génération aléatoire d'une population de taille N, pour passer ensuite à l'évaluation des individus de la population par la fonction fitness, tant que le nombre de génrations maximum n'est pas encore atteint l'algorithme procède comme suit : il sélectionne les parents pour faire le croisement et la mutation, il applique les opérateurs génétiques cités ci-dessus pour produire une population d'enfants Q(t), une nouvelle population R(t) de taille 2N est construite R(t) = $P(t) \cup Q(t)$, les individus de R(t) vont être classés en différents fronts non dominé pour sélectionner les meilleurs N individus de cette population, ces meilleurs solutions vont servir de population prochaine.

Algorithm 9: Algorithme NSGA-II

1. Initialise L = taille de population, $Itermax$, $Pcroi$, $Pmut$;
2. Générer une population initiale P_0 de taille L d'une façon aléatoire ;
3. $i = 1$;
4. Évaluation de tout les chromosomes par " fitness=C^A_{max} et TEC^B " ;
5. Trier les solutions par **non domination** et **crowding distance** ;
6. **While** i < Itermax :
   Générer une nouvelle population C de taille L :
7. Sélectionner deux parents de la population ;
8. Générer une probabilité P ;
9. **If** $P < Pcroi$:
10. Croiser les deux parents sélectionnés ;
11. **Endif**
12. **If** $P < Pmut$:
13. Muter les parents sélectionnés ;
14. **Endif** ;
15. Evaluation des individus de la population P ∪ C ;
16. Trier les solutions par **non domination** et **crowding distance** ;
17. $i = i + 1$;
18. **EndWhile** ;

Le schéma de fonctionnement général de l'algorithme NSGA-II est présenté dans la figure ci-dessous :

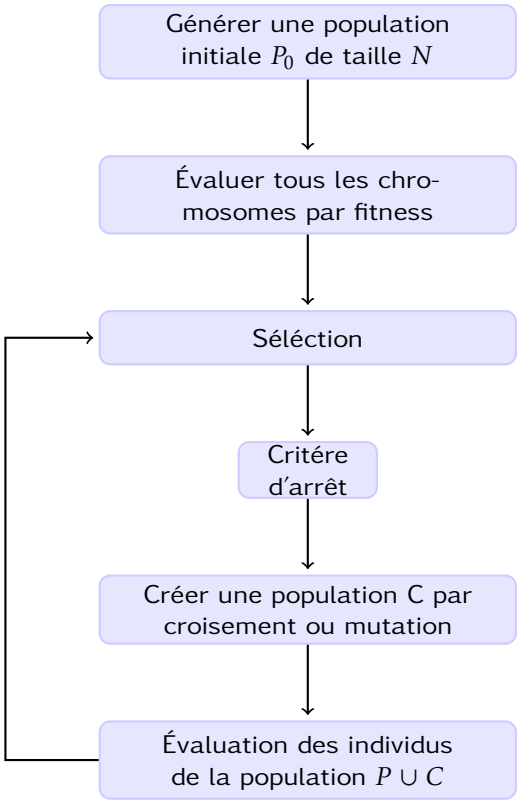


FIGURE 4.4 – Organigramme de l'algorithme NSGA-II

4.3 La prise de décision multicritères

Les méthodes d'optimisation multi-objectifs mentionnées auparavant fournissent un ensemble de solutions Pareto optimales en laissant le choix au décideur de désigner la meilleure solution parmi toutes les alternatives en fonction de la priorité de chaque objectif, dans la suite nous allons présenter l'une de ces méthodes adoptée pour notre étude.

4.3.1 Technique pour l'ordre de préférence par similarité de solution idéale (TOPSIS)

Cette méthode a été développée par Yoon et Hwang - 1981 [13], elle considère les solutions comme des alternatives et les valeurs des fonctions objectifs de chaque alternative comme des critères, son principe est basé sur le choix d'une solution qui se rapproche le plus d'une solution idéale et s'éloigne de la solution qui dégrade tous les objectifs, cela en calculant la distance euclidienne pour chaque alternative par rapport à l'idéal et l'anti-idéal.

4.3.1.1 La procédure de TOPSIS

D'abord, on crée la matrice de décision D = $(x_{ij})_{nm}$ pour n alternatives et m critères, où x_{ij} est la valeur du j^{ime} critère de la i^{ime} alternative.

– **étape 1 :** Calcul de la matrice normalisée :

$$r_{ij} = \frac{x_{ij}}{\sum_{k=1}^{n} x_{kj}^2}; \; i = 1, 2, ..., n \; et \; j = 1, 2, ..., m$$

– **étape 2 :** Calcul du produit des performances normalisées par les valeurs des pondérations relatives au critère pour l'obtention de la matrice normalisée pondérée :

$$T = (t_{ij}) \quad ou \quad t_{ij} = r_{ij} * w_j$$

Sachant que le décideur assigne à chaque critère un poids $w_j \in [0,1]$ $telque$ $\sum_{j=1}^{m} w_j = 1$.

– **étape 3 :** La détermination de la meilleure alternative (l'idéal) et la pire alternative (l'anti-idéal), on divise l'ensemble des critères en deux sous-ensembles F^- et F^+ où F^- (resp F^+) représente l'ensemble de critère de coût (resp de bénéfice) :

$$t_{bj} = \{ (\max_i t_{ij} \mid j \in F^+) , (\min_i t_{ij} \mid j \in F^-) \} \quad i = 1,2,...,n \; et \; j = 1,2,...,m$$

$$t_{wj} = \{ (\min_i t_{ij} \mid j \in F^+) , (\max_i t_{ij} \mid j \in F^-) \} \quad i = 1,2,...,n \; et \; j = 1,2,...,m$$

– **étape 4 :** Calcul des distances d_{ij} entre chaque alternative i ∈ {1..n } et j ∈ {1..m }la meilleure et pire alternatives pour chaque critère

$$d_{ib} = \sqrt{\sum_{j=1}^{m} (t_{ij} - t_{bj})^2} \quad i = 1,2,...,n$$

$$d_{iw} = \sqrt{\sum_{j=1}^{m} (t_{ij} - t_{wj})^2} \quad i = 1,2,...,n$$

– **étape 5 :** Calcul de similarité à la pire alternative.

$$s_{iw} = \frac{d_{iw}}{d_{ib} + d_{iw}} \quad i = 1,2,...,n$$

– **étape 6 :** Trier les alternatives en les classant selon l'ordre de préférence.

CHAPITRE 5

EXPÉRIMENTATIONS NUMÉRIQUES

Dans ce chapitre nous nous focalisons sur les tests et les résultats obtenus après l'implémentation des méthodes développées précédemment, ces méthodes sont codées en langage de programmation Python et executées sur un PC exécutant Windows 10 Professionnel, doté d'un CPU M380 de 2.53 GHz et de 4 Go de RAM.

5.1 Résolution exacte du modèle mathématique :

5.1.1 Implémentation du modèle sur CPLEX

Afin de résoudre ce problème, nous allons implémenter le modèle mathématique sous Cplex. Celà, tout d'abord en créant un fichier .mod dans lequel nous allons insérer le modèle, et un fichier .dat dans lequel nous allons introduire l'instance à n tâches donnée.

5.1.1.1 Solveur ILOG CPLEX

Le Cplex est un solveur de programmation mathématique, son nom est la composition de C= faisant référence au langage de programmation C, et plex pour Simplex.

Il repose sur une implémentation performante du simplexe primal-dual.

Initialement développé par l'équipe de Robert Bixby, ensuite commercialisé par la société CPLEX qui a été rachetée par ILOG en 1996. En 2009, ILOG a été rachetée par IBM d'ou l'appelation IBM ILOG CPLEX.

OPL (Optimization Programming Language) est un langage permettant d'écrire des problèmes d'optimisation dans une syntaxe abstraite que les solveurs peuvent exploiter. L'interface utilisée pour écrire des problèmes d'optimisation en OPL est l'IBM Ilog CPLEX Optimisation Studio.

IDE (Integrated Development Environment) est un environnement de développement intégré pour développer et tester les modèles.

Les différents tests réalisés à l'aide du solveur mathématique CPLEX, ont été tous effectués sur un PC exécutant Windows 10 Professionnel, doté d'un CPU M 380 de 2.53 GHz et de 4 Go de RAM.

L'implémentation du modèle mathématique sur le solveur Cplex a pu nous donner des résultats, ceux ci sans prendre en considération la détérioration des tâches il est à noter que Cplex donne des résultats jusqu'à 25 tâches, ces résultats sont donnés dans le tableau suivant où n définit le nombre de tâches et CPU time est le temps de calcul moyen en secondes :

n	C^A_{max}	TEC^B	CPU time
5	9	24.87	0.58
10	18	146.77	0.95
15	32	886.67	15.9
20	45	1010.14	34.97
25	56	1633.34	70.24
29	-	-	> 1800

En effet, lorsque la déterioration des tâches est incluse le modèle mathématique devient un modèle quadratique dû à la variable de décision de la détérioration des tâches, ce qui nous mène à explorer d'autres méthodes de résolution.

5.1.2 Méta-heuristiques :

5.1.2.1 Mise en place des paramètres nécessaires à la résolution :

Les méthodes proposées sont testées sur quatre classes d'instances, nous avons généré ces instances pour le problème de taille n = 10, n = 30, n = 50, n = 100 et n = 400 et m = 2, elles sont caractérisées par des temps d'exécution générés d'une manière aléatoire :

- **instance 1 :** $p_{ij} \in [1\ ,\ 10\ ]$.
- **instance 2 :** $p_{ij} \in [10\ ,\ 20\ ]$.
- **instance 3 :** $p_{ij} \in [20\ ,\ 30\ ]$.
- **instance 4 :** $p_{ij} \in [30\ ,\ 50\ ]$.

Les valeurs proposées pour les paramètres des algorithmes sont données dans le tableau suivant :

Paramètres	valeurs selectionées
recherche tabou	
nbr-itération	20n
Recuit simulé	
T_0	100
T_f	0.01
α	0.999
β	1
NSGAII	
nbr-générations	200-700
taille de population	10n
proba-crois	0.8
proba-mut	0.2

Le taux de déterioration des tâches est considéré fixe à 0.1 pour toutes les tâches de l'agent B, la consommation d'énergie par unité de temps δ_i et le taux d'augmentation d'énergie θ_i de chaque machine sont données dans le tableau :

	M_1	M_2
δ_i	2	1
θ_i	0.1	0.2

TABLE 5.1

5.1.2.2 Implémentation de recherche tabou et recuit simulé :

Pour la résolution du problème général P nous allons considérer les deux problèmes P_1 et P_2 où P_1 est un problème d'ordonnancement de type flow shop avec deux agents en compétition où le makespan du premier agent est à minimiser en gardant l'énergie totale consommée par le deuxième agent inférieure à une valeur donnée quant au problème P_2 l'objectif est de minimiser l'énergie consommée par l'agent B et maintenir le makespan du premier agent inférieur à une valeur donnée.

Lors de l'implémentation des deux méthodes proposées pour la résolution des problèmes P_1 et P_2, et en faisant des tests avec les algorithmes proposés pour la solution initiale, la solution générée avec l'heuristique H1 est retenu pour le problème P_1 et celle générée avec H2 pour le problème P_2, les résultats obtenus pour les deux méta-heuristiques le recuit simulé et la recherche tabou sont présentés dans les tableaux ci-dessous.

Problème (P1)								
	instance 1		**instance 2**		**instance 3**		**instance 4**	
	TS	**SA**	**TS**	**SA**	**TS**	**SA**	**TS**	**SA**
$n = 10$								
C^A_{max}	15	15	88	88	150	150	248	248
TEC^B	145.32	147.94	652.17	658.1	1109.6	1129.84	1850.82	1850.83
time(s)	0.6	0.71	0.9	1.59	0.9	2.28	0.8	2.53
$n = 30$								
C^A_{max}	75	73	232	232	407	414	632	637
TEC^B	2130.06	2051.99	6843.09	7092.7	11925.87	12015.47	20062.89	20227.12
time(s)	2.75	2.45	4.16	4.33	5.9	4.8	5.61	3.72
$n = 50$								
C^A_{max}	129	134	372	376	642	653	1033	1043
TEC^B	10257.86	10349.54	32074.75	32009.6	56956.84	57701.7	91842.80	9205.94
time(s)	7.12	3.45	7.93	3.44	8.2	4.51	9.3	4.32
$n = 100$								
C^A_{max}	249	249	765	765	1288	1288	2095	2091
time(s)	10.4	8.8	14.16	10.17	14.69	11.05	15.14	9.25
$n = 400$								
C^A_{max}	1029	1029	2857	2977	5088	5088	8203	8196
time(s)	103.6	79.27	115.6	73.93	120.35	72.11	134.2	72.24

TABLE 5.2 – Résultats expérimentaux des instances 1, 2, 3 et 4 pour $P1$

Observations : En regardant les résultats obtenus après l'implémentation de la recherche tabou (TS) et le recuit simulé (SA) pour le problème P_1, on constate que pour n = 10 les valeurs obtenus par les deux méthodes sont similaires or que l'algortihme de recherche tabou converge plus rapidement à la solution optimale.

Pour $n \in \{30, 50, 100, 400\}$ en général l'algorithme TS donne un meilleur résultat, mais en terme de temps d'exécution ce dernier converge moins rapidement que l'algorithme SA pour donner sa solution optimale.

	Problème (P2)							
	instance 1		instance 2		instance 3		instance 4	
	TS	SA	TS	SA	TS	SA	TS	SA
$n = 10$								
C^A_{max}	43.49	43.49	194.92	194.92	320.52	320.52	548.8	548.89
TEC^B	87.2	87.2	347.66	347.66	592.6	592.64	1013.62	1013.63
time(s)	0.78	0.72	1.7	1.03	1.7	2.29	1.8	2.01
$n = 30$								
C^A_{max}	184.65	183.80	684.6	683.11	1276	1264.66	2057.25	2046.67
TEC^B	496.36	494.67	2038.4	2031.95	3918.08	3822.92	6524.8	6520
time(s)	15.4	1.84	16.2	2.06	18.65	2.29	20.46	0.99
$n = 50$								
C^A_{max}	550.47	554.23	1821.6	1818	3245.27	3246.88	5229.76	5226
TEC^B	2051.51	2062	7464	7461.36	13819.16	13822.57	21945.91	21938.16
time(s)	28.4	3.83	29.2	2.83	27.7	0.99	24.3	2.78
$n = 100$								
TEC^B	21149.74	20264.99	97984.7	99865.16	172299.1	174914.71	276211.10	279069
time(s)	30.4	6.96	31.8	5.79	28.73	6.27	34.26	5.63
$n = 400$								
TEC^B	26128536.42	26118720.31	84451654.89	84423425.57	166997981.36	166991575	323919654.67	323917132.21
time(s)	206.3	53.17	298.5	54.22	318.79	59	478.9	54.14

TABLE 5.3 – Résultats expérimentaux des instances 1, 2, 3 et 4 pour $P2$

Observations : En observant les résultats obtenus après l'implémentation de la recherche tabou (TS) et le recuit simulé (SA) pour le problème P_2, on voit que pour n = 10 les deux méthodes donnent le même résultat pour les quatre instances, pour $n \in \{30, 50, 100, 400\}$ le recuit simulé donne de meilleurs valeurs que l'algorithme de recheche tabou en un temps d'exécution meilleur.

5.1.2.3 Implémentation de l'algorithme génétique de tri non dominé II :

Les résultats expérimentaux obtenus par la métaheuristique NSGA-II pour les quatre classes d'instances sont reportés dans le tableau suivant :

TABLE 5.4 – Front Pareto pour n = 10

Sol_i	Sol_1	Sol_2	Sol_3	Sol_4	Sol_5	Sol_6	Sol_7	Sol_8	Sol_9	Sol_{10}
C^A_{max}	15	43.48	32.37	18	26.8	28.8	23.72	22.4	23.24	22.92
TEC^B	145.32	87.2	96.72	131.38	107.61	107.17	118.38	122.92	121.02	121.11

On applique la méthode TOPSIS au front-Pareto obtenu dans la table 5.4 utilisant plusieurs poids.

Le premier poids coressponds au critère du makespan et le deuxième à l'énergie consommée, les différents résultats sont présentés dans le tableau suivant :

Poids	Ordre obtenu
0.9-0.1	1-4-8-10-9-7-5-6-3-2
0.6-0.4	4-1-8-10-9-7-5-6-3-2
0.5-0.5	4-1-8-10-9-7-5-6-3-2
0.4-0.6	4-7-10-8-5-9-1-6-3-2
0.1-0.9	2-3-6-5-7-9-10-8-4-1

TABLE 5.5 – Résultats de TOPSIS

La solution (15 , 145.2) est la meilleure alternative selon la méthode TOPSIS dans le cas où on assigne un poids de 0.9 et 0.1 pour les deux objectifs C^A_{max} et TEC^B respectivement.
La figure ci-dessous représente le front de Pareto pour n = 10 :

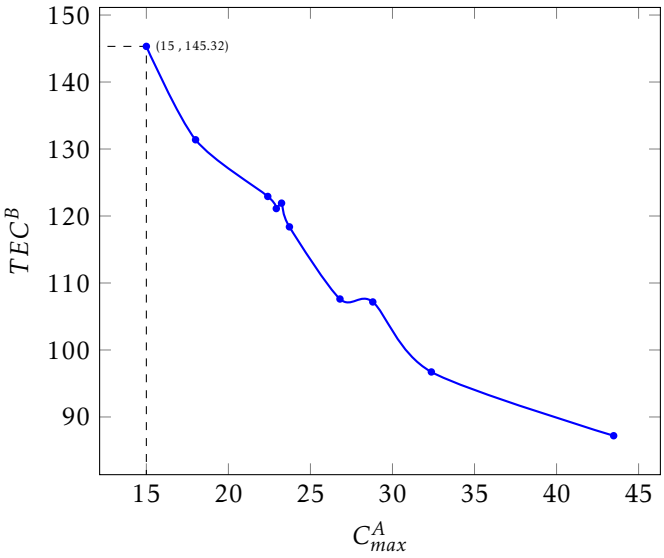


FIGURE 5.1 – Front de Pareto pour n =10

En appliquant la méthode TOPSIS au front-pareto obtenu lors de l'implémentation de NSGA-II pour n ∈ {30,50,100}, on obtient le résultats suivant :

0.5-0.5	n = 30	n = 50	n = 100
C^A_{max}	121.9	259.3	744.4
TEC^B	796.3	2900.1	27830.9

TABLE 5.6 – Résultats de TOPSIS avec un poids de 0.5 pour chaque critère

0.9-0.1	n = 30	n = 50	n = 100
C^A_{max}	93.4	181.1	453.4
TEC^B	1164.13	4115.8	39048.9

TABLE 5.7 – Résultats de TOPSIS avec un poids de 0.9 pour le makespan et 0.1 pour la consommation énergétique

CONCLUSION ET PERSPECTIVES DE RECHERCHES

Dans ce mémoire, nous nous sommes intéressées à un problème d'ordonnancement d'atelier de type Flow Shop avec deux agents en compétition. Ce problème considère un ensemble de tâches à ordonnancer sur deux machines dédiées dans le même ordre et deux sous-ensembles de tâches sont définis tels que chacun d'eux est associé à un agent, dont l'objectif est de minimiser simultanément la date de fin de traitement des tâches du premier agent et l'énergie totale consommée pour le deuxième, en respectant la détérioration des tâches pour ce dernier.

En premier lieu, nous avons construit deux modèles mathématique du problème étudié en appliquant l'approche ϵ-contrainte à notre problème, Les deux modèles ont été formulés et validés par le solveur CPLEX sans tenir compte de la détérioration des tâches de l'agent B. Les résultats obtenus montrent que le modèle proposé peut résoudre de manière exacte jusqu'à 25 tâches.

En deuxième lieu pour la résolution du problème deux méthodes de recherche locale à savoir la recherche tabou et le recuit simulé et un algorithme génétique NSGA-II «Non-dominated Sorting Genetic Algorithm-II » ont été proposées et testées.
A l'issue de ce travail, plusieurs perspectives de recherches seraient pertinentes pour poursuivre cette étude :

— Proposer des bornes inférieures.
— L'étude de la complexité du problème.
— il est intéréssant de comparer les méthodes utilisées avec d'autres méthodes comme AMOSA.
— il est aussi intéréssant d'étudier ce problème en considérant d'autres critères d'optimalité.

BIBLIOGRAPHIE

[1] Alessandro Agnetis, Jean-Charles Billaut, Stanisław Gawiejnowicz, Dario Pacciarelli, Ameur Soukhal, et al. Multiagent scheduling. *Berlin Heidelberg : Springer Berlin Heidelberg. doi*, 2014.

[2] Allesandro Agnetis, Pitu B Mirchandani, Dario Pacciarelli, and Andrea Pacifici. Scheduling problems with two competing agents. *Operations research*, 2004.

[3] Abdennour Azerine, Mourad Boudhar, and Djamal Rebaine. A two-machine no-wait flow shop problem with two competing agents. *Journal of Combinatorial Optimization*, 43(1) :168–199, 2022.

[4] Kenneth R Baker and J Cole Smith. A multiple-criterion model for machine scheduling. *Journal of scheduling*, 2003.

[5] Mathieu Bouchard, Alain Hertz, and Guy Desaulniers. Lower bounds and a tabu search algorithm for the minimum deficiency problem. *Journal of combinatorial optimization*, 2009.

[6] Jean-Pierre Briot and Yves Demazeau. *Principes et architecture des systèmes multi-agents*. 2001.

[7] Jacques Carlier and Bruno Latapie. Une méthode arborescente pour résoudre les problèmes cumulatifs. *RAIRO-Operations Research*, 1991.

[8] Johann Dréo, Alain Pétrowski, Patrick Siarry, and Eric Taillard. *Métaheuristiques pour l'optimisation difficile*. 2003.

[9] Edward A Feigenbaum, Julian Feldman, et al. *Computers and thought*. 1963.

[10] Michael R Garey and David S Johnson. Computers and intractability. w. h, 1979.

[11] Fred Glover and Manuel Laguna. Tabu search. In *Handbook of combinatorial optimization*. 1998.

[12] Ronald Lewis Graham, Eugene Leighton Lawler, Jan Karel Lenstra, and AHG Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling : a survey. In *Annals of discrete mathematics*. 1979.

[13] Ching-Lai Hwang and Kwangsun Yoon. Methods for multiple attribute decision making. In *Multiple attribute decision making*. 1981.

[14] Selmer Martin Johnson. Optimal two-and three-stage production schedules with setup times included. *Naval research logistics quarterly*, 1954.

[15] Scott Kirkpatrick, C Daniel Gelatt Jr, and Mario P Vecchi. Optimization by simulated annealing. *science*, 1983.

[16] Michel Minoux. *Programmation mathématique. Théorie et algorithmes*. 2008.

[17] Ibrahim H Osman and Gilbert Laporte. Metaheuristics : A bibliography, 1996.

[18] Vilfredo Pareto. *Cours d'économie politique : professé à l'Université de Lausanne*. 1896.

[19] Agnes Pechmann and Ilka Schöler. Optimizing energy costs by intelligent production scheduling. In *Glocalized Solutions for Sustainability in Manufacturing*. 2011.

[20] Nidamarthi Srinivas and Kalyanmoy Deb. Muiltiobjective optimization using nondominated sorting in genetic algorithms. *Evolutionary computation*, 1994.