N° d'ordre : 15/013-M/MT

République Algérienne Démocratique et Populaire

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université des Sciences et de la Technologie "Houari Boumediene"

Faculté de Mathématiques

MEMOIRE Présenté pour l'obtention du diplôme de Magister

En : MATHEMATIQUES

Spécialité : Recherche Opérationnelle : Génie Mathématiques

Présenté par

Drifa HETTAK

Résolution d'un Problème d'Ordonnancement Bicritère à Machines Parallèles

soutenu le 15 Janvier 2013, devant le jury composé de

Mrs	**M. MOULAÏ**	Professeur	U.S.T.H.B	Président de jury
	M. BOUDHAR	Professeur	U.S.T.H.B	Directeur de thèse
	M. E-A. CHERGUI	M.A.A	U.S.T.H.B	Examinateur
	K. BOUIBEDE	M.C.A	U.S.T.H.B	Invitée

Remerciements

Louange à Dieu miséricordieux, sans lui rien de tout cela n'aurait pu être.

Je remercie chaleureusement Madame BOUIBED Karima, qui a guidé mes premier pas sur les chemins de l'ordonnancement multicritère et qui a su me supporter avec un constant esprit à la fois de rigueur et de convivialité. Pour qui disponibilité, gentillesse et qualité des idées et/ou critiques sont naturelles.

Mes grands remerciements et sincères respects vont à Monsieur BOUDHAR Mourad, mon encadrant, pour l'aide précieuse qu'il a apportée pour la réalisation de ce travail, pour l'intérêt qu'il a porté à mon travail ainsi que ses précieux conseils et ses critiques fondées.

J'adresse mes remerciements aux membre de jury
Monsieur MOULAI Mustapha, Professeur à l'université des sciences et de la technologie Houari Boumedienne pour m'avoir honoré en acceptant de présider ce jury.
Monsieur CHERGUI Mohamed El Amin, Maître de Conférences à l'université des sciences et de la technologie Houari Boumedienne d'avoir bien voulu juger ce travail.

Pour finir, un travail de mémoire ne se fait pas sans soutiens, des proches qui vous apportent équilibre et réconfort dans les moments difficiles. Je veux donc remercier de tout mon coeur mes parents, ma famille et enfin mon Mari Ali qui a fait de grand sacrifices pour que vous puissiez lire cela aujourd'hui.

TABLE DES MATIÈRES

LISTE DES TABLEAUX

TABLE DES FIGURES

INTRODUCTION

En raison de l'ouverture du marché, les entreprises sont confrontées à une concurrence redoutable, et faire une place dans ce monde n'est pas toujours facile. En effet, seule une démarche stratégique permet de guider l'entreprise dans des choix à long terme.

Les industriels sont impliqué dans une problématique d'organisation, consistant à maximiser les profits en maîtrisant les coûts de production et en minimisant des délais tout en mettant sur le marché des produits innovant à des prix compétitifs. Ainsi pour atteindre ces objectifs, cette organisation doit se baser sur la mise en œuvre d'un certain nombre de procédés en général et sur l'ordonnancement en particulier. En effet, ces soixante dernières années, l'ordonnancement a suscité un grand intérêt en raison de son applicabilité à un nombre important de problèmes liés au monde industriel.

Les problèmes d'ordonnancements consistent à parvenir à planifier, les tâches à accomplir, en précisant les instants auxquelles elles vont débuter et les ressources nécessaires à utiliser. En général, ces problèmes sont de nature multicritère. En effet, plusieurs critères sont souvent à satisfaire afin de pouvoir fournir aux décideurs une solution de meilleur compromis.

L'enjeu économique considérable de ces problèmes à susciter une recherche intensive pour développer des outils et des méthodes efficace de résolution. Pour les problème à un seul critère, la solution fournie par une méthode de résolution est clairement définie, tandis que, pour les problèmes multicritères dont les critères sont souvent contradictoires, la solution optimale recherchée n'est plus un point unique mais un ensemble de points correspondant aux meilleurs compromis.

De nombreuses méthodes de résolution des problèmes multicritères ont été développées , elles sont classées en différentes catégories : méthodes exactes, méthodes approchées et méthodes hybrides. Pour les problèmes de taille réduites, les méthodes exactes fournissent des solutions exactes. Par contre pour les problèmes de grande taille, le temps de calcul devient si important que la méthode développée devient inutilisable en pratique.

Dans de telles situations et afin de palier à ce problème, des méthodes approchées ont vu le jour, elles fournissent des solutions non pas "optimale" mais proche de l'optimalité ou "de bonne solution", en un temps raisonnable. Et pour mieux rendre ces solutions encore plus satisfaisantes, des méthodes hybrides combinant plusieurs méthodes ont été développées.

Dans ce mémoire, nous étudions le problème d'ordonnancement bicritère à machines parallèles uniformes $Q/r_i, d_i/C_{max}, L_{max}$. L'objectif est la minimisation de la date de fin de traitement (makespan) et du retard algébrique. Ce problème étant $\mathcal{NP}$-difficile au sens fort, différentes approches ont été développées.

Le chapitre 1 est une présentation générale des problèmes d'ordonnancement, nous définissons les concepts fondamentaux d'un problème d'ordonnancement, ses contraintes et ses objectifs et sa classification, ainsi que, les notions sur la théorie de la complexité.

Le chapitre 2 constitue une introduction aux problèmes combinatoire multi-objectif en général et plus particulièrement les problèmes d'ordonnancement multicritère, nous définissons des notions liées à l'optimalité ainsi que les méthodes de résolutions des problèmes d'ordonnancement.

Le chapitre 3 présente un état de l'art du problème d'ordonnancement bicritère.
Il s'agit d'un problème à machines parallèles uniformes où chaque travail possède une date de début au plus tôt, et une date échue. L'objectif est de minimiser simultanément la date de fin de l'ensemble des travaux et leur retard algébrique. Ce problème se note $Q/r_i, d_i/C_{max}, L_{max}$.

Dans le chapitre 4 nous résolvons le problème décrit précédemment par trois versions de recherche tabou basées chacune sur une technique pour la détermination des solutions non-dominées : une marche aléatoire, une combinaison linéaire des critères et une approche epsilon-contrainte, nous développons aussi deux algorithmes génétiques et deux algorithmes mémétiques combinant la recherche tabou et un algorithme génétique.

Le dernier chapitre est consacré à l'expérimentation numérique des différentes méthodes décrites dans le chapitre précédent.

Notre travail s'achèvera par une conclusion et quelques perspectives.

CHAPITRE 1

GÉNÉRALITÉS SUR L'ORDONNANCEMENT

Ce chapitre est consacré à la présentation des notions fondamentales concernant les problèmes d'ordonnancement et à la théorie de la complexité.

1.1 Introduction à l'ordonnancement

L'ordonnancement est un outil qui vise à améliorer l'efficacité d'une entreprise en terme de coûts de production et de délais de livraison, cela consiste à programmer l'exécution d'un ensemble de travaux en leurs attribuant des ressources, et en respectant des contraintes fixées, afin d'optimiser un critère donné ou de trouver un compromis entre plusieurs critères.

On rencontre les problèmes d'ordonnancement dans différents domaines d'application. Tel que l'informatique (gestion de fichiers, exécution de programmes), en administration (gestion des emplois du temps) et bien d'autres domaines.

À partir de la définition d'un problème d'ordonnancement, nous distinguons les quatres notions fondamentales. Ce sont les travaux, les ressources, les contraintes et le ou les critères à optimiser.
Les travaux : Un travail ou une tâche (job en anglais) est une activité élémentaire dont la réalisation nécessite une durée dite opératoire, il est localisé dans le temps par une date de début et une date de fin. Dans certains problèmes, l'exécution des travaux peut être interrompue puis reprise ultérieurement mais seulement la partie restante du travail est exécutée au moment de la reprise, on parle ici d'un problème préemptif.
Souvent les travaux sont liés par des relations de précédence, dans le cas contraire on dit qu'ils sont indépendants.

Les ressources : Une ressource est un moyen matériel ou humain mis en oeuvre pour permettre l'exécution d'un ou plusieurs travaux. On distingue des ressources consommables donc limitées

(matières premières, budget) et des ressources renouvelables, qui ne s'épuisent pas après utilisation (machines, main d'oeuvre). Par ailleurs, on distingue les ressources disjonctives qui n'exécutent qu'un seul travail à la fois et des ressources cumulatives qui disposent d'une capacité qui leur permet d'exécuter simultanément plusieurs travaux.

Les contraintes : Elles expriment les conditions techniques que doivent respecter simultanément les variables de décision, liées aux travaux et aux ressources, et qui sont représentatives d'un problème d'ordonnancement. on distingue :

- *Contraintes temporelles ou de potentiels* : elles contiennent les contraintes représentant les dates limites d'exécution des travaux : contraintes de localisation temporelle c'est la date de disponibilité et les contraintes de délai (délai de livraisons), contraintes de succession ou de précédence appelées aussi contraintes de cohérence technologique. différent types de succession existent tel que la succession simple, succession avec attente,...etc.
- *Contraintes de ressources* : elles expriment la disponibilité et les caractéristiques d'utilisation des moyens utilisés par les travaux.

Le critère d'optimisation : C'est la fonction objectif à maximiser ou à minimiser (cela dépend du problème étudié). On parle d'un modèle d'optimisation mono-critère ou mono-objectif quand il s'agit d'un seul critère à optimiser sinon il est multicritère ou multi-objectif.
On distingue quatres catégories de critères :

- Des critères liés au temps.
- Des critères liés aux ressources.
- Des critères liés à l'équilibrage de la charge.
- Des critères liés aux coûts (coût de lancement, de fabrication,...ect) et aux revenus (profit,...etc).

1.1.1 Caractérisation des machines

Les machines peuvent être parallèles et réaliser la même fonction, ou être spécialisées (dedicated machines) dans la réalisation de certains types d'opérations.

Les machines parallèles

Ces machines sont capables de réaliser n'importe quel travail, nous distinguons trois types de machines :

- Les machines identiques (identical machines) : Dans ce cas, la durée opératoire d'un travail est la même sur toutes les machines.
- Les machines uniformes (uniform machines) : Les vitesses de traitement des travaux sont différentes mais la vitesse de chaque machine est constante et ne dépend pas des travaux.

– Les machines générales (unrelated machines) : La vitesse de traitement d'une machine dépend du travail qui est traité.

Les machines spécialisées

Ces machines sont spécialisées aux traitements de certains travaux seulement, et nous distinguons trois types de mode de traitement :

– Flow Shop : C'est un cas particulier d'ordonnancement d'atelier pour lequel le cheminement des travaux est unique, c'est à dire, les travaux doivent être exécutées sur l'ensemble des machines dans un même ordre (ligne de production).

FIGURE 1.1 – Flow Shop

.

– Open Shop : Les travaux doivent être traités par toutes les machines mais l'ordre du traitement n'est pas nécessairement le même pour tous les travaux, l'ordre de traitement est quelconque. Comparé aux autres modèles d'ateliers multi-machines, l'open shop qui n'est pas non plus courant dans les entreprises, n'est pas très étudié dans la littérature.
– Job Shop : les n travaux doivent être exécutés sur les m machines, sous des hypothèses identiques à celles du flow shop, la seule différence est que les séquences opératoires relatives aux différents travaux peuvent être distinctes et sont propres à chaque travail.

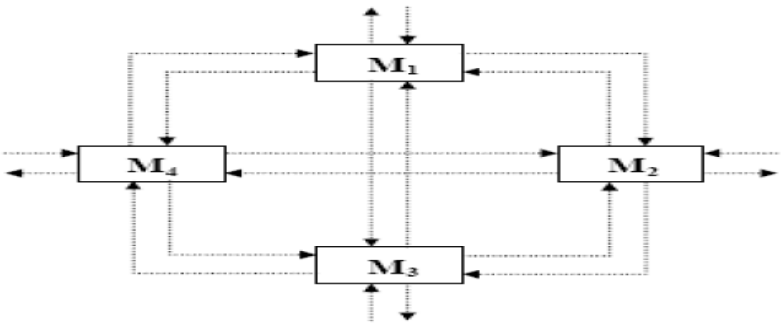

FIGURE 1.2 – Représentation d'un atelier Job Shop

.

1.1.2 Description des travaux

Sur les machines parallèles, les travaux sont traités sur n'importe quelle machine, contrairement, aux travaux qui sont traitées sur les machines spécialisées, chaque travail T_i est décomposé en un nombre fini d'opérations $O_{i1}, O_{i2}, ..., O_{ik}$.

- Dans un système Flow Shop, le nombre d'opérations par travail est égal à m et leur ordre de traitement est tel que l'opération O_{ij} est traitée sur la machine M_j avant l'opération O_{ij+1} traitée sur la machine M_{j+1}.
- Le système Open Shop est le même que le système Flow Shop mais sans la contrainte de précédence.
- Dans un système Job Shop, le nombre d'opérations par travail, leur allocation à des machines et l'ordre de leur traitement sont arbitraires mais doivent être donnés à l'avance.

Comme dans la plupart des ouvrages [Brucker, 2004] et [Graham et al., 1979]sur les problèmes d'ordonnancement, nous utilisons les notations présentées dans le tableau suivant :

m :	nombre de ressources (machines) $M_j(j = 1, \dots m)$
n :	nombre de travaux $T_i(i = 1 \dots n)$
p_i :	durée de traitement du travail T_i (processing time)
r_i :	date de disponibilité du travail T_i (release date)
d_i :	date de livraison (ou d'achèvement maximal souhaité) du travail T_i (due date)
w_i :	poids, relatif à un coût unitaire, du travail T_i(weight)
t_i :	date de début de traitement du travail T_i(starting time)
c_i :	date de fin de traitement du travail T_i (completion time)
f_i :	$f_i = c_i - r_i$: temps de séjour du travail T_i
ou c'est la somme du temps d'attente et du temps de traitement (flow time)	
l_i :	$l_i = c_i - d_i$: retard algèbrique du travail (lateness)
t_i :	$t_i = max(c_i - d_i, 0)$: (tardiness)
u_i :	indicateur de retard $u_i = 1$ si $t_i > 0$ et $u_i = 0$ sinon

TABLE 1.1 – Les caractéristiques d'un travail

1.2 Notation usuelles

[Graham et al., 1979] ont proposé une notation qui permet la classification, la distinction et l'identification des différents types de problèmes d'ordonnancement. Un problème d'ordonnancement,est désigné par une notation composée de trois champs : $\alpha/\beta/\gamma$.

Champ α : les ressources

Ce champ caractérise les machines (ressources). Il est constitué de deux sous-champs $\alpha = \alpha_1\alpha_2$:

- $\alpha_1 = \{\emptyset, P, Q, R, F, J, O\}$: représente le type de ressources principales (machines) utilisées :

- $\alpha_2 = \{\emptyset, m\}$: dénote le nombre de machines composant le système. Lorsque $\alpha_2 = \{\emptyset\}$, le nombre de machines est variable, tandis que lorsque $\alpha_2 = \{m\}$, le nombre de machines est égal à m ($m > 0$).

Champ β : contraintes

Ce champ $\beta = \{\beta_1\beta_2\beta_3\beta_4\beta_5\beta_6\beta_7\beta_8\}$: décrit les caractéristiques des ressources et des travaux.

- $\beta_1 \in \{\emptyset, pmtn\}$: indique la possibilité de préemption (β_1= pmtn). Si $\beta_1 = \emptyset$, la préemption n'est pas autorisée.
- $\beta_2 \in \{\emptyset, res\}$: caractérise les ressources supplémentaires nécessaires au traitement d'un travail (outils, ressources de transport). $\beta_2 = \emptyset$ indique qu'il n'y a pas de ressource complémentaire. $\beta_2 = \{res\}$ spécifie les ressources complémentaires.
- $\beta_3 \in \{\emptyset, prec, tree, chain\}$: représente les contraintes de précédence entre travaux spécifiant qu'un travail doit être traité avant un autre. Les valeurs $\emptyset$, prec, tree, chain représentent respectivement des travaux indépendants, des relations de précédence générales, des relations de précédence particulières sous forme d'arbres ou de chaînes.
- $\beta_4 \in \{\emptyset, r_i\}$: décrit les dates d'arrivées (ou dates de disponibilité) des différents travaux dans le système. Ces dates peuvent être identiques et égales à zéro pour tout les travaux ($\beta_4 = \{\emptyset\}$) ou différents suivant les travaux ($\beta_4 = \{r_i\}$).
- $\beta_5 \in \{\emptyset, p_i = p, \underline{p} \leq p \leq \overline{p}\}$: détaille les temps de traitement des différents travaux. Ces temps peuvent être ou ne pas être en fonction de la machine.
  $\beta_5 = \emptyset$: les travaux ont des temps de traitement arbitraires.
  $\beta_5 = p_i = p$: tous les travaux ont un temps de traitement égale à p.
  $\beta_5 = \underline{p} \leq p \leq \overline{p}$: les temps de traitement des travaux sont compris entre $\underline{p}$ et $\overline{p}$
- $\beta_6 = \{\emptyset, d_i, \hat{d}_i\}$: indique les éventuelles dates d'échue (ou dates au plus tard) des travaux, dates pour lesquelles les travaux doivent être terminés.
  $\beta_6 = \emptyset$: les travaux n'ont pas de date d'échue.
  $\beta_6 = d_i$: chaque travail a une date d'échue souhaitée. Généralement, un non respect de ces dates entraîne une pénalisation.
  $\beta_6 = \hat{d}_i$: chaque travail a une date d'échue impérative (date limite), qu'il faut absolument respecter.

- $\beta_7 = \{s, s_i, s_{ij}, nwt\}$: permet de spécifier des contraintes sur les enchaînements des travaux très souvent introduites pour représenter au mieux les problèmes réels. Il est parfois nécessaire de considérer un temps de changement entre traitement de deux travaux différents sur une même machine pour représenter les changements et réglages d'outils. Ces temps de changement peuvent être constants (s), en fonction du nouveau travail (s_i) ou bien en fonction de l'enchaînement des deux travaux (s_{ij}). Comme, on peut imposer que toutes les opérations d'un travail soient traitées sans temps d'attente ($\beta_7 = nwt(no - wait)$).
- Enfin, $\beta_8 = \{\emptyset, M_j\}$: indique, dans le cas de machines parallèles, des restrictions sur la polyvalence des machines. L'ensemble M_j représente l'ensemble des machines capables de réaliser le travail j. Lorsque β_8 est vide, toutes les machines sont capables de traiter tous les travaux.

Champ γ : champ du critère d'optimisation

Ce dernier champ γ indique le critère à optimiser. Les fonctions à minimiser les plus utilisées dans la littérature sont les suivantes :

- C_{max} : la durée totale de l'ordonnancement qui est la plus grande date de fin d'exécution d'un travail (makespan).
- $\sum_{i=1}^{n} C_i$: la somme des dates de fin des travaux (total completion time).
- $\sum_{i=1}^{n} w_i C_i$: la somme pondérée des dates de fin des travaux (total weighted completion time).
- $\sum_{i=1}^{n} T_i$: la somme des retards (total tardiness).
- $\sum_{i=1}^{n} w_i T_i$: la somme pondérée des retards (total weighted tardiness).
- L_{max} : le retard algébrique maximum (lateness).
- $\sum_{i=1}^{n} U_i$: le nombre de travaux en retard.
- $\bar{C} = \frac{1}{n} \sum_{i=1}^{n} c_i$: Le temps de fin de traitement moyen (mean completion time).
- $\bar{L} = \frac{1}{n} \sum_{i=1}^{n} l_i$: La tardiveté moyenne (mean lateness).
- $\bar{F} = \frac{1}{n} \sum_{i=1}^{n} f_i$: Le temps de flot moyen (mean flow time).
- $\bar{C_w} = \frac{\frac{1}{n} \sum_{i=1}^{n} c_i}{\sum w_i}$: Le temps de fin de traitement moyen pondéré (mean weighted completion time).
- $\bar{L_w} = \frac{\frac{1}{n} \sum_{i=1}^{n} l_i}{\sum w_i}$: La tardiveté moyenne pondérée (mean weighted lateness).
- $\bar{F_w} = \frac{\frac{1}{n} \sum_{i=1}^{n} f_i}{\sum w_i}$: Le temps de flot moyen pondéré (mean weighted flow time).

1.3 Notions sur la complexité algorithmique

Nous nous intéressons ici à la théorie de complexité car elle permet de déterminer le degré de difficulté d'un problème donné c'est à dire le classer mathématiquement en un problème "facile" ou "difficile" à résoudre. La connaissance de la difficulté d'un problème nous aidera à choisir le type d'algorithme à utiliser.

Généralement, on distingue deux types de problèmes : les problèmes de décision et les problèmes d'optimisation. Un problème de décision est un problème pour lequel on cherche à répondre à une question par un "oui" ou par un "non". Pour un problème d'optimisation, on cherche à trouver une solution au problème considéré.

Pour un algorithme, le temps nécessaire pour obtenir une solution optimale détermine sa complexité temporelle et l'espace mémoire de cet algorithme nous renseigne sur sa complexité spaciale mais en général, c'est la première complexité qui nous intéresse.

Définition 1.3.1. *Un algorithme est dit polynomial si sa complexité est bornée par un $O(p(n))$ où p est un polynôme et n est la longueur d'une instance du problème. Un algorithme est pseudo-polynomial si le polynôme bornant le temps d'execution dépend de la taille de l'instance et de l'amplitude de plus grand entier. Sinon l'algorithme est dit exponentiel.*

Les problèmes de décision sont classés en deux classes : la classe $\mathcal{P}$ et la classe $\mathcal{NP}$.

Définition 1.3.2. *La classe $\mathcal{P}$ contient les problèmes pouvant être résolu par un algorithme polynomial et ce sont des problèmes dit "facile".*

Définition 1.3.3. *La classe $\mathcal{NP}$ est la classe des problèmes pour lesquels on peut vérifier une réponse "oui" par un algorithme polynomial.*

Définition 1.3.4 (Réduction polynomial)**.** *On dit que P_1 se réduit polynomialement à P_2 si et seulement s'il existe un algorithme polynomial construisant à partir des données d'une instance P_1 les données d'une instance P_2, de sorte que la réponse à P_1 est "oui" si et seulement si la réponse à P_2 est oui.*

Définition 1.3.5. *La classe $\mathcal{NP}$-complet est inclut dans la classe $\mathcal{NP}$, elle contient les problèmes les plus difficile. Un problème est $\mathcal{NP}$-complet si tous les problèmes de $\mathcal{NP}$ se réduisent polynomialement à lui.*

Définition 1.3.6. *On dit qu'un problème est $\mathcal{NP}$-complet au sens fort s'il ne peut pas être résolu par un algorithme pseudo-polynomial sinon le problème est $\mathcal{NP}$-complet au sens faible.*

On peut aussi définir les problèmes $\mathcal{NP}$-difficiles, ce sont des problèmes d'optimisation pour lesquels les problèmes de décision associés sont $\mathcal{NP}$-complet [Garey et Johnson, 1979].

[Cook, 1971] fut le premier à démontrer la $\mathcal{NP}$-complétude d'un problème appelé SAT (satisfaisabilité), d'autres problèmes ont été démontré comme étant des problèmes $\mathcal{NP}$-complet, on peut citer : problème de voyageur de commerce, problème de cycle hamiltonien, problème de la clique maximum, problème de couverture de sommets dans un graphe,...etc.

CHAPITRE 2

L'ORDONNANCEMENT MULTICRITÈRE

Dans ce chapitre, nous présentons l'optimisation multi-objectif en général et l'ordonnancement multicritère en particulier. Nous introduisons la notion d'optimalité à savoir la notion de dominance et d'optima de Pareto. Par la suite, nous décrivons les principales méthodes de résolution des problèmes multi-objectif.

2.1 Optimisation multi-objectif

La plupart des problèmes d'optimisation combinatoire rencontrés en pratique sont de nature multi-objectif et les critères à optimiser sont souvent conflictuels.

L'optimisation multi-objectif a été utilisé initialement en économie et dans les sciences du management dans les travaux de Edgeworth et Pareto [Pareto, 1896]. Puis, elle s'est propagé dans les domaines des sciences de l'ingénieur. Malgré l'importance de ce type de problèmes, on trouve peu d'ouvrages dans la littérature avant les années 80 et 90, mais depuis, plusieurs chercheurs se sont intéressés à ce genre de problèmes.

La résolution d'un problème multi-objectif cherche à trouver les meilleurs compromis entre les différents objectifs conflictuels c'est à dire, chercher l'ensemble des solutions non dominées connu comme solutions de Pareto optimales. Ainsi la notion d'optima de Pareto et de dominance sont fondamentals en optimisation multi-objectifs.

2.1.1 Formulation des problèmes multi-objectif

Les problèmes d'optimisation multi-objectif sont une généralisation à n fonctions objectifs des problèmes d'optimisation classique. Il sont définis comme suit :
Considérons un problème de minimisation :

$$\begin{cases} \text{min Z(s) avec } Z(s) = [Z_1(s), \ldots, Z_k(s)]^T,\ s \in S\,; \\ S = \big\{s/[h_1(s), \ldots, h_p(s)]^T = 0,\ [g_1(s), \ldots, g_M(s)]^T \leq 0\big\}. \end{cases}$$

La fonction vectorielle $Z = [Z_1, \ldots, Z_k)]^T$ est appelée fonction multi-objectif ou vecteur des critères, dont les composantes $Z_i, i \in \{1, 2, \ldots, k\}$ sont des critères à optimiser (fonctions coûts). Les équations $h_j(s) = 0, j = \overline{1, p}$ et les inégalités $g_l(s) = 0, l = 1, \ldots, M$ sont les contraintes.

2.1.2 Notions d'optimalité

Même si les études concernant les problèmes d'ordonnancement ont d'abord considéré les problèmes dont l'objectif est l'optimisation d'un seul critère, il est clair que les problèmes réels sont parfois bien différents. En effet dans la pratique, on est souvent confronté à deux ou plusieurs critères à optimiser. Par exemple, on peut chercher à maximiser la qualité d'un projet tout en minimisant une fonction de coût total. Naturellement, une solution optimisant en même temps les deux critères n'a que très peu de chances d'exister puisqu'ils sont conflictuels. On est donc amené à rechercher un compromis entre une solution de grande qualité ayant un coût élevé, et une solution de faible coût et de qualité moindre. [TKindt et Billaut, 2006] fournissent un tour d'horizon des problèmes d'ordonnancement multicritères et le lecteur intéressé y trouvera de nombreuses informations. La notion d'optimalité d'une solution d'un problème multicritère peut être définie à l'aide de la notion d'*optimum* de *Pareto*.
Nous rappelons dans un premier temps les définitions de base en optimisation multicritère, qui seront utilisées par la suite ([TKindt et Billaut, 2006]). Soit $\mathcal{S}$ l'ensemble des solutions d'un problème et x un élément de $\mathcal{S}$. On note $Z = \{Z_j, 1 \leq j \leq k\}$ l'ensemble des critères conflictuels considérés et $\mathcal{Z}$ l'image de $\mathcal{S}$ dans l'espace des critères. On note z l'image de $x \in \mathcal{S}$ dans l'espace des critères.

Définition 2.1.1. *$s_1 \in \mathcal{S}$ est un optimum de Pareto faible (ou solution faiblement efficace) si et seulement si $\nexists s_2 \in \mathcal{S}$ tel que $\forall j, 1 \leq j \leq k, Z_j(s_2) < Z_j(s_1)$. L'ensemble des optima de Pareto faibles définit dans l'espace des critères la courbe de compensation, appelée aussi courbe d'efficacité.*

Définition 2.1.2. *$s_1 \in \mathcal{S}$ est un optimum de Pareto strict (ou solution efficace) si et seulement si $\nexists s_2 \in \mathcal{S}$ tel que $\forall j, 1 \leq j \leq k, Z_j(s_2) \leq Z_j(s_1)$ avec au moins une inégalité stricte. Un optimum de Pareto strict est également un optimum de Pareto faible.*

Dans le cas monocritère, une solution est représentée dans l'espace des critères par un point sur un axe, et on a affaire à un ordre total lorsque l'on compare plusieurs solutions. Dès lors que l'on est confronté à plusieurs critères à optimiser, une solution est représentée dans l'espace des critères par un vecteur comportant autant de composantes qu'il y a de critères. Ainsi, deux solutions données ne sont pas nécessairement comparables et on a affaire à un ordre partiel entre les solutions. On n'a donc plus, comme dans le cas monocritère, une seule valeur pour l'ensemble des solutions optimales, qui peut être réduit ou non à un singleton, mais bien un ensemble de vecteurs de dimension K. Si cet ensemble est de faible cardinalité, on dit que les critères sont *peu conflictuels*. Par contre si cet ensemble contient beaucoup d'éléments, les critères sont dits *très conflictuels*.

Ces notions sont illustrées dans la Figure 2.1 dans le cas bicritère. Dans cette figure z^2, z^3, z^4 et z^5 sont des optima de Pareto stricts et z^1 et z^6 sont des optima de Pareto faibles non stricts. Bien souvent, on préférera s'intéresser à l'ensemble des Pareto stricts plutôt qu'à l'ensemble des Pareto faibles, car ce dernier peut contenir des solutions peu intéressantes pour le décideur.

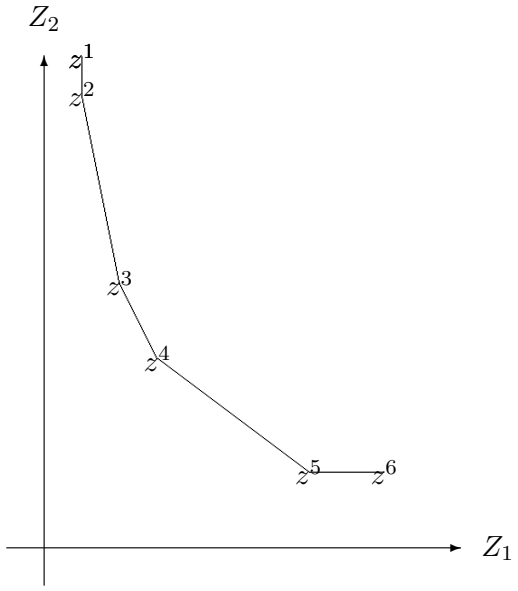

FIGURE 2.1 – Illustration des définitions 2.1.1 et 2.1.2 pour deux critères

Définition 2.1.3. *Le front de Pareto est l'ensemble de tous les optima de Pareto stricts, il est aussi appelé surface de compromis.*

Un exemple de surface de compromis (front de Pareto) dans le cas bicritère est montré à la Figure 2.2. Dans cet exemple, le problème considéré est un problème de minimisation avec deux critères. Deux points particuliers apparaissent clairement : le point idéal et le point Nadir. Ces deux points sont calculés à partir du front de Pareto. Le point idéal (resp. le point Nadir) domine (resp. est dominé par) tous les autres points de la surface de compromis. Bien que ces points ne soient pas forcément compris dans la zone réalisable, ils servent souvent de pôle d'attraction (resp. de répulsion) lors de la résolution du problème. Les coordonnées de ces deux points sont maintenant définies formellement.

Définition 2.1.4. *Les coordonnées du point idéal correspondent aux meilleurs valeurs de chaque objectif des points du front Pareto. Les coordonnées de ce point correspondent aussi aux valeurs obtenues en optimisant chaque fonction objectif séparément.*

Définition 2.1.5. *Les coordonnées du point Nadir correspondent aux pires valeurs de chaque objectif des points du front Pareto.*

Définition 2.1.6. *Un ensemble A est convexe, si et seulement si :*

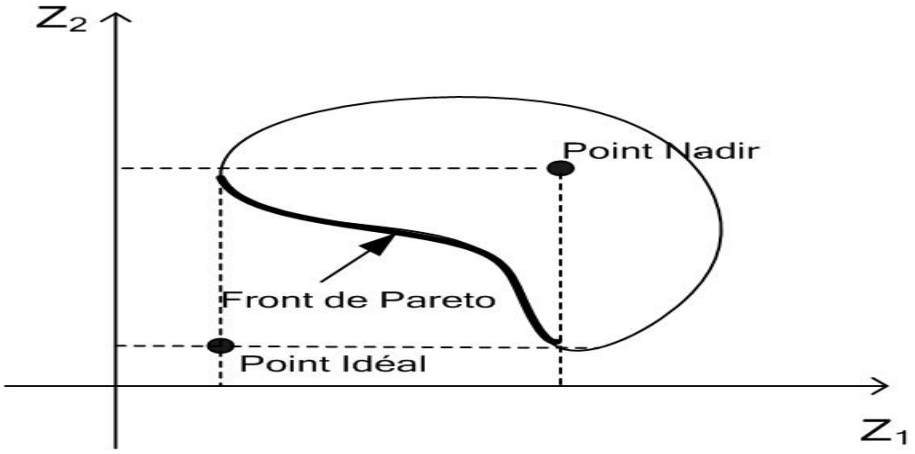

FIGURE 2.2 – le front de Pareto et les points de référence

.

$$\forall \lambda_1, \lambda_2 \geq 0,\ \lambda_1 + \lambda_2 = 1,\ \forall z^1 \in A \text{ et } \forall z^2 \in A,\ \lambda_1 z^1 + \lambda_2 z^2 \in A$$

Définition 2.1.7. *Une solution $s \in \mathcal{S}$ est dite un optimum de Pareto supportée si elle est située sur l'enveloppe convexe d'un ensemble de solutions réalisables.*

Définition 2.1.8. *Une solution $s \in \mathcal{S}$ est dite un optimum de Pareto non supportée si elle est située à l'intérieur d'un ensemble de solutions réalisables.*

2.2 Notations des problèmes d'ordonnancement multicritères

Un problème d'ordonnancement multicritère, peut se noter de façon générale en utilisant la notation à trois champs de [Graham et al., 1979], ou le champs γ contient la liste des critères défini précédemment : $\alpha/\beta/Z_1, Z_2, ..., Z_k$. L'étape de prise en compte des critères constitue une grande partie de difficulté pour résoudre un problème multicritère. Lors de cette phase, nous pouvons déduire si nous autorisons ou non des compensations entre les critères. Si ce n'est pas le cas, nous aurons l'ordre dans lequel les critères doivent être optimisés. En revanche, si les compensations sont permises, nous pouvons déduire s'il est possible de pondérer les critères. Le problème d'ordonnancement issu de l'étape de prise en compte des critères peut également se noter à l'aide des trois champs, où seul le champ γ est étendu. Le Tableau 2.1 présente les nouvelles fonctions définis dans [TKindt et Billaut, 2006] pour le champs γ.

2.3 Classes des méthodes de résolution

La résolution d'un problème multi-objectif demande l'intervention du décideur pour le choix de la solution. Evans ([Evans, 1984]) présente trois moments où le décideur peut intervenir dans le processus

Fonction	Signification
Z	Si l'objectif est de minimiser l'unique critère Z (problème monocritère).
$F_l(Z_1, ..., Z_k)$	Si l'objectif est de minimiser une combinaison linéaire (convexe) des k critères.
$P(Z_1, ..., Z_k)$	Indique une fonction non décroissante des critères à minimiser, si on suppose que tous les critères sont bornés supérieurement par une valeur connue, c'est à dire qu'il n'est pas possible pour les critères de prendre une valeur supérieure à la borne indiquée.
$\epsilon(Z_u/(Z_1, ..., Z_{u-1}, Z_{u+1}, ..., Z_k)$	Indique que seul le critère Z_u est minimisé, sachant que tous les autres critères sont bornés supérieurement par une valeur connue.
$F_T(Z_1, ..., Z_k)$	Indique une fonction objectif qui est l'expression d'une distance à une solution idéale connue, en utilisant pour le calcul de la distance la métrique de Tchebycheff. Cette solution idéale ne doit pas être atteignable.
$F_{Tp}(Z_1, ..., Z_k)$	Indique une fonction objectif qui est l'expression d'une distance à une solution idéale connue, en utilisant pour le calcul de la distance la métrique de Tchebycheff pondérée. Cette solution idéale ne doit pas être atteignable.
$F_{Tpa}(Z_1, ..., Z_k)$	Indique une fonction objectif qui est l'expression d'une distance à une solution idéale connue, en utilisant pour le calcul de la distance la métrique de Tchebycheff pondérée augmentée. Cette solution idéale ne doit pas être atteignable.
$F_s(Z_1, ..., Z_k)$	Indique une fonction bien particulière qui prend en compte une solution idéale connue pour trouver la solution cherchée.
$Lex(Z_1, ..., Z_k)$	Indique qu'on n'autorise pas la compensation entre les critères. L'ordre dans lequel les critères sont donnés est lié à l'importance qui leur est accordée, le plus important étant en première position.

TABLE 2.1 – Le champ γ pour les problèmes d'ordonnancement multicritères

de résolution : avant, pendant ou après (Figure 2.3). A chacune de ces phases correspond une grande catégorie de méthodes :

1. Les méthodes *a priori* :

   Dans ces méthodes, le décideur définit le compromis qu'il désir réaliser (il fait part de ses préférences) et fourni un ensemble d'informations au processus automatique de résolution pour pouvoir lancer son exécution, comme par exemple la valeur des poids pour un problème de minimisation d'une combinaison linéaire des critères. La détermination de la valeur de ces paramètres constitue un problème en soi, qui nécessite l'utilisation de méthodes dédiées. Le lecteur intéressé peut se reporter notamment à [Roy, 1985].

2. Les méthodes *interactives* :

   Dans les méthodes *interactives*, le processus de résolution est itératif. A chaque itération il fournit au décideur une solution, qui n'est pas nécessairement un optimum de Pareto. Ce dernier oriente le processus en lui fournissant de nouvelles valeurs pour les paramètres du problème. Par exemple, il peut s'agir de nouveaux poids pour la combinaison linéaire des critères. Le processus est alors capable de calculer une nouvelle solution et l'itération suivante peut débuter. Cette classe de méthodes a fait l'objet de nombreuses études dans le cadre de l'optimisation multicritère.

3. Les méthodes *a posteriori* :

   Ces méthodes visent à fournir au décideur un ensemble exhaustif d'optima de Pareto, parmi lesquels celui-ci retiendra la solution la plus satisfaisante à ses yeux en examinant toutes les solutions de compromis. Ces méthodes sont également appelées méthodes d'énumération.

2.4 Approches de résolution

Les problèmes d'optimisation combinatoire en général, et les problèmes d'ordonnancement ont donné lieu au développement de nombreux types de méthodes de résolution. Certaines utilisent des connaissances du problème pour fixer des préférences sur les critères et ainsi contourner l'aspect multicritère du problème. Plusieurs ouvrages ou articles de synthèse ont été rédigés, dans lesquels des états de l'art peuvent être consultés notamment dans [Ulungu et Teghem, 1994], [Ehrgott, 2005], [Ehrgott et Gandibleux, 2000], [Collette et Siarry, 2003] et [TKindt et Billaut, 2006].

2.4.1 La méthode d'agrégation par combinaison linéaire

Cette méthode a été proposée par [Geoffrion, 1968], c'est l'une des premières approches utilisée dans la littérature sur les problèmes multicritères. Elle consiste à transformer un problème multicritère en un problème à un seul critère en agrégeant les différents critères sous la forme d'une somme pondérée :

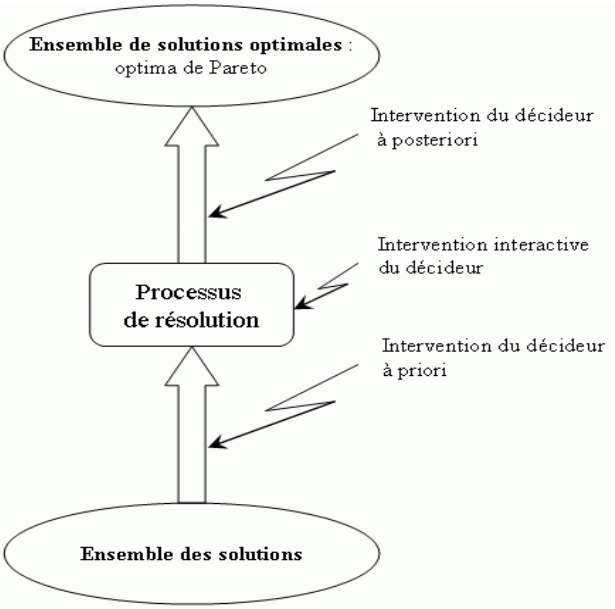

FIGURE 2.3 – Intervention du décideur dans les méthodes de résolution

.

$$Z_{\sum} = \sum_{i=1}^{n} \omega_i . Z_i(x)$$

Les w_i, appelés poids, peuvent être normalisés sans perte de généralité : $\sum_{i=1}^{K} w_i = 1$. Il est clair que la résolution d'un problème pour un vecteur de poids $w \in \Re_+^K$ ne permet de calculer que quelques optima de Pareto. Pour obtenir un ensemble contenant un grand nombre d'optima de Pareto, il faut résoudre plusieurs fois le problème en changeant à chaque fois les valeurs de w. Cette approche a l'avantage évident de pouvoir réutiliser tous les algorithmes classiques dédiés aux problèmes d'optimisation à un seul critère. C'est souvent la première approche adoptée lorsqu'un chercheur se retrouve devant un nouveau problème multicritère. Cependant cette approche a aussi deux inconvénients importants. Le premier est dû au fait que pour avoir un ensemble de points bien répartis sur le front de Pareto, les différents vecteurs w doivent être choisis judicieusement. Il est donc nécessaire d'avoir une bonne connaissance du problème. Le deuxième inconvénient provient du fait que cette méthode ne permet pas, de calculer intégralement la surface de compromis lorsque celle-ci n'est pas convexe. La Figure 2.4 illustre ce cas de figure dans le cas bicritère. En effet, pour un vecteur de poids $w = (w_1, w_2)$ fixé, la valeur optimale atteignable pour la fonction objectif crée est Z. Les deux optima de Pareto trouvés sont z^1 et z^3. En faisant varier le vecteur w, il est possible de trouver d'autres optima de Pareto. Seulement, tous ces points se trouveront sur les parties convexes de la surface de compromis. Il n'existe pas de valeur possible pour w permettant de trouver par exemple le point z^2. En effet cette méthode ne nous permet pas de calculer les optima de Pareto stricts non supportés dans le cas d'un problème non convexe. Grâce au principe de pondération, il est possible d'obtenir le front de Pareto d'un problème. Il est nécessaire pour cela d'exécuter plusieurs optimisations successives en faisant varier les facteurs de pondération (Figure 2.5). Généralement dans cette méthode nous commençons par déterminer les

solutions extrêmes càd les solutions qui minimisent un seul critère à la fois. En partant de ces solutions nous calculons de nouvelles valeurs pour les paramètres de pondérations pour pouvoir déterminer de nouvelles solutions.

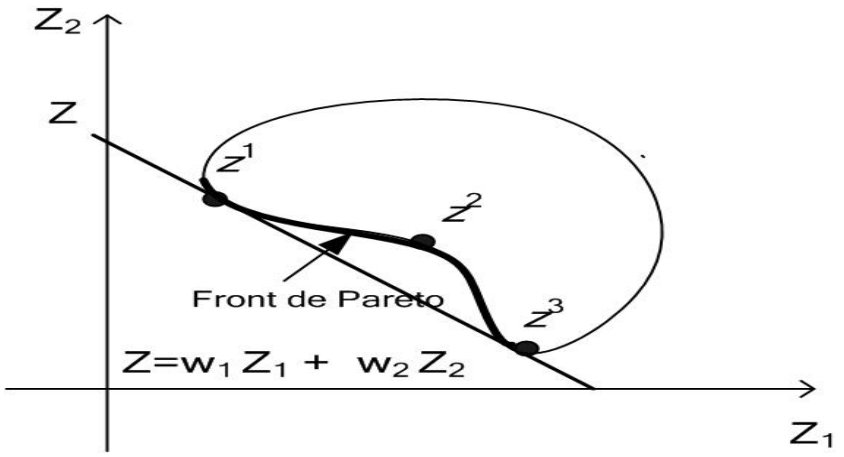

FIGURE 2.4 – Interprétation graphique de l'approche par combinaison linéaire

.

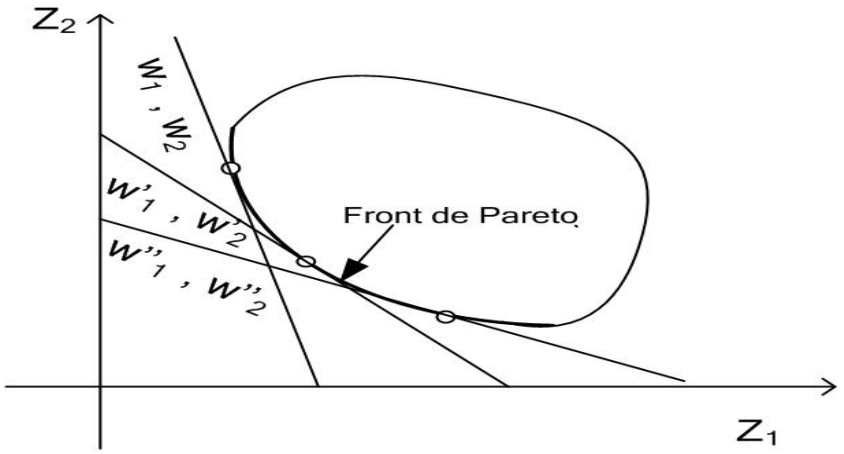

FIGURE 2.5 – Illustration de l'énumération des solutions avec la méthode d'agrégation par combinaison linéaire

.

2.4.2 L'approche ϵ-contrainte

L'approche ϵ-contrainte est très souvent utilisée dans la littérature. Elle consiste à ramener un problème d'optimisation multicritère à un problème monocritère en transformant itérativement les $K-1$ des K critères du problème en contraintes et à optimiser séparément le critère restant ([Haimes et al., 1971] et [Haimes et al., 1975]). La fonction objectif se note $\epsilon(Z_1/Z_2, ..., Z_k)$ dans le

cas où on recherche la solution minimisant Z_1 dans l'ensemble des solutions ayant des valeurs de Z_i telles que $Z_i \leq \epsilon$, $\forall i \in 2, ..., k$. [Klein et Hannan, 1982] présentent un principe d'énumération reposant sur cette approche. L'approche ϵ-contrainte peut aussi être appliquée plusieurs fois en faisant varier les $k-1$ valeurs du vecteur ϵ. Sachant qu'au début la valeur d'ϵ bornant chaque critères est initialisée à l'infini puis on lui affecte les valeur des critères de la meilleur solution trouvée, ainsi de suite.
Cette approche a l'avantage par rapport à la précédente de ne pas être trompée par les problèmes non convexes. Ainsi la Figure 2.6 illustre, dans le cas bicritère, le cas où un point (ϵ, Z_{2min}), de la partie non convexe, est trouvé.

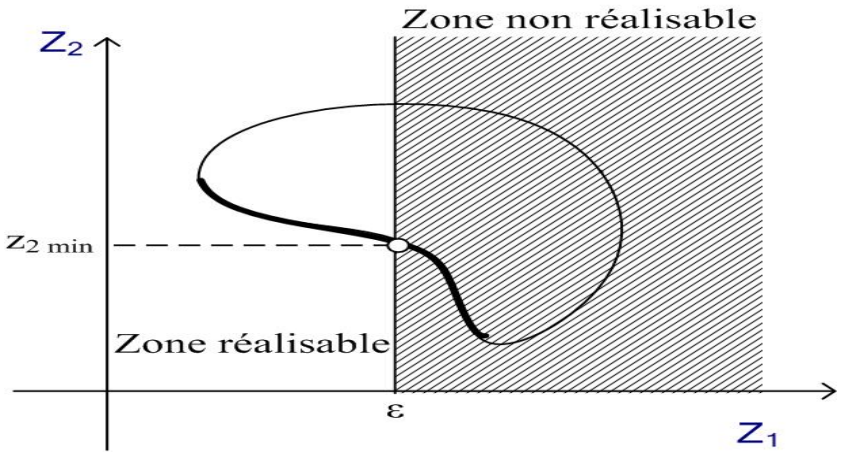

FIGURE 2.6 – Interprétation graphique de l'approche ϵ-contrainte

.

Considérons un problème bicritère avec Z_1 et Z_2 les deux critères à minimiser, cette méthode consiste à borner Z_1 par une valeur ϵ càd $Z_1 \leq \epsilon$ et à minimiser le critère non borné Z_2. Une fois qu'une solution est trouvée nous recalculons la valeur du paramètre ϵ pour déterminer une nouvelle solution. Pour énumérer l'ensemble des optima de Pareto nous réitérons le processus d'optimisation en recalculant à chaque étape une nouvelle valeur de ϵ la Figure 2.7 illustre le principe d'énumération dans le cas bicritère. En transformant des fonctions objectifs en contraintes, elle diminue la zone réalisable par paliers. Ensuite, le processus d'optimisation trouve le point optimal sur le critère restant. L'inconvénient de cette approche réside dans le fait qu'il faut lancer un grand nombre de fois le processus de résolution. De plus, pour obtenir des points intéressants et bien répartis sur la surface de compromis, les valeurs de ϵ doivent être choisies judicieusement. Il est clair qu'une bonne connaissance du problème a priori est requise.

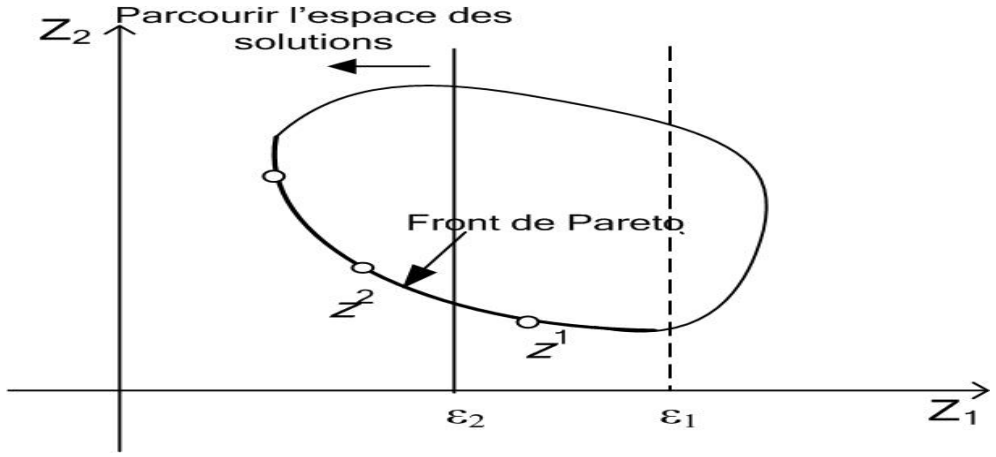

FIGURE 2.7 – Illustration de l'énumération des solution avec l'approche ϵ-contrainte

.

2.5 Les métaheuristiques pour les problèmes d'optimisation multicritère

Les méthodes exactes donnent des solutions optimales pour certains problèmes, toutes fois ces méthodes de résolution ne peuvent pas être appliquées pour beaucoup de problèmes sur des instances à grande taille.
Les méta-heuristiques sont conçues afin de trouver de bonnes solutions mais pas nécessairement optimales en un temps raisonnable. On peut citer comme méthodes : les algorithmes constructives (l'algorithme glouton), les méthodes de recherche locale [Selman et al., 1994], les algorithmes évolutionnaires [Back et al., 1997], les algorithmes de colonies de fourmis [Colorni et al., 1991], [Colorni et al., 1992], les méthodes hybrides [Talbi, 2002]...ect.

2.5.1 La recherche locale

Les méthodes de recherche locale sont des méthodes de résolution qui résolvent les problèmes d'optimisation multicritère de manière itérative, elles évoluent la solution courante en la remplaçant par une autre solution issue de son voisinage et c'est ce qu'on appelle le mouvement. La détermination de ce voisinage consiste à caractériser tous ses éléments.
La méthode de la descente, le recuit simulé et la recherche tabou sont des méthodes de la recherche locale. Nous nous intéresserons à la méthode de recherche tabou.

La recherche Tabou

La méthode de la recherche Tabou a été introduite par [Glover, 1986]. Son principe repose sur la recherche d'une meilleure solution parmi l'ensemble des solutions qui appartiennent au voisinage de la

solution courante.

À partir d'une solution initiale s_{ini} qui devient la solution courante s, le meilleur voisin v_{best} devient la nouvelle solution courante s, ce processus recommence jusqu'à ce qu'une condition d'arrêt soit satisfaite.

Ce processus peut conduire à un cyclage et pour empêcher ce type de problème d'apparaître, une liste de taille tl est crée et on mémorise les tl dernières solutions déjà visitées dans cette liste et on interdit de retenir toute solution qui en fait déjà partie. Cette liste est appelé la **liste Tabou**.

Certaines recherches Tabou utilisent un critère d'aspiration. Dans ce cas, on s'autorise à choisir un voisin tabou s'il est strictement améliorant. Le critère d'aspiration n'est utilisé que lorsque la liste Tabou ne contient que la description du mouvement. La mémorisation des solutions s'avère trop coûteuse en temps de calcul et espace mémoire d'où l'intérêt de mémoriser des caractéristiques relatives aux solutions par exemple mémoriser un attribut de cette solution.

La génération de la solution initiale se fait généralement en utilisant des heuristiques, cela dépend des problèmes à traiter, dans le cas du problème d'ordonnancement, les règles de trie sont beaucoup utilisées et donnent de bonnes solutions initiales mais cela n'empêche pas l'utilisation d'autres méthodes.

Il existe d'autres techniques qui permettent d'améliorer la performance de la recherche tabou : l'intensification et la diversification. *L'intensification* permet de se focaliser sur certaines solutions de très bonne qualité rencontrées au cours de la recherche. Par contre *la diversification* permet de rediriger la recherche vers des zones inexplorées dans l'espace des solutions.

2.5.2 Les algorithmes évolutionnaires

Les algorithmes évolutionnaires [Back et al., 1997] ont été inspirés de l'évolution biologique, ils sont des méthodes d'optimisation stochastiques qui simulent le processus de l'évolution naturelle.
Les algorithmes évolutionnaires se caractérisent par :

- Une représentation spécifique des solutions potentielles du problème.
- Un ensemble d'individus formant la population permettant de mémoriser l'ensembles des résultats à chaque étape du processus de recherche.
- Un processus de création aléatoire d'un individu. Cette caractéristique offre une capacité exploratoire importante à la méthode.
- Un ensemble d'opérateurs de modification permettant de créer de nouvelles solutions à partir des informations mémorisées. Ces opérateurs offrent une capacité de recherche locale à la méthode.
- ...etc.

On distingue plusieurs types d'algorithmes évolutionnaires [Back et al., 1997] : les stratégies d'évolution [Schwefel, 1984], la programmation évolutive introduite par [Fogel, 2000] et les algorithmes

Algorithme de la recherche tabou (TS)	
1	Pré-condition : $\gamma(x)$:critère à maximiser ou à minimiser
2	Générer la solution initiale s_{init}
3	$s := s_{best} := s_{init}$
4	$tl = \emptyset$: liste tabou
5	TantQue (condition d'arrêt non satisfaite) faire
6	$v_{best} = \emptyset$
7	Construire l'ensemble des voisins $V(s)$ de la solution courante
8	Pour chaque $v \in V(s)$ Faire
9	Si $v \notin V(s)$ Alors
10	$v_{best} = v$;
11	FinSi
12	FinPour
13	$s = v_{best}$; $tl = tl \cup v_{best}$ (mise à jour de la liste tabou)
14	Si $\gamma(s_{best}) < \gamma(s)$ Alors
15	$v_{best} = s$ (mise à jour de la meilleure solution)
16	FinSi
17	FinTantQue

FIGURE 2.8 – Principe général de la recherche tabou(TS)

génétiques [Holland, 1975] et [Goldberg, 1989] qui sont à l'origine le fruit des recherches de John Holland.

On s'intéressera dans ce qui suit aux algorithmes génétiques.

- **A. Les algorithmes génétiques :**

  Les algorithmes génétiques font partie de la famille des algorithmes évolutifs, ils imitent des phénomènes d'adaptation des êtres vivants et en s'inspirant de : "la survie est pour l'individu le mieux adapté à l'environnement". Ces algorithmes ont suscité l'intérêt de nombreux chercheurs à commencer par Holland en 1975 qui a développé les principes fondamentaux des algorithmes génétiques [Holland, 1975], puis Goldberg [Goldberg, 1989] les a utilisés pour résoudre des problèmes concrets d'optimisation.

  Les algorithmes génétiques sont basés sur :

  1. Le choix d'un codage des solutions.
  2. La génération d'une population initiale.
  3. La définition d'une fonction d'évaluation (fitness) permettant d'évaluer une solution et la comparer aux autres.
  4. Le choix des solutions par un mécanisme de sélection.
  5. La génération de nouvelles solutions à l'aide des opérateurs génétiques en utilisant :

     - Opérateur de croisement : il manipule la structure des chromosomes des parents afin de produire des individus meilleurs et/ou différents, cet opérateur opère selon une probabilité.
     - Opérateur de mutation : il évite d'établir des populations uniformes incapable d'évoluer. Il consiste à modifier les valeurs des gènes de chromosomes selon une probabilité.
  6. Établir un compromis entre les solutions produites (progénitures) et les solutions productrices (parents) en utilisant un mécanisme d'insertion.

  Le déroulement des algorithmes génétiques peut être résumé en :

  1. ***Codage des solutions :*** L'utilisation efficace d'un algorithme génétique dépend du codage de la solution, ce codage dépend essentiellement de la nature du problème traité. Souvent le codage le plus utilisé est le codage numérique.

  2. ***Génération de la population initiale :*** La bonne connaissance du problème à traiter permet de générer une population initiale contenant des individus qui convergent vers l'optimum plus rapidement.

  3. ***Méthode de sélection :*** Cette méthode permet d'identifier les individus candidats à

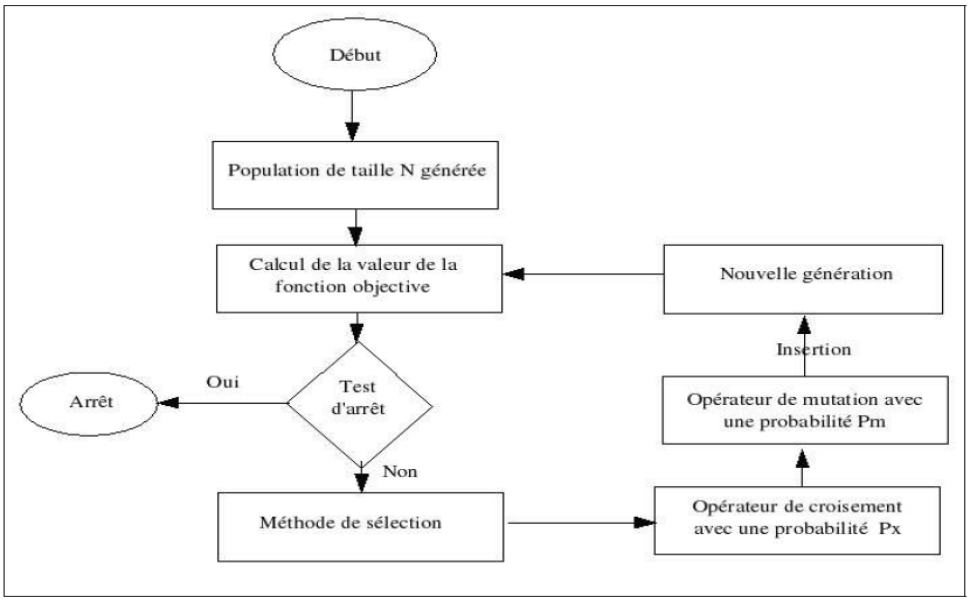

FIGURE 2.9 – L'organigramme de l'algorithme génétique

.

être croiser dans une population, on distingue :

Sélection par rang : Elle consiste à attribuer à chaque individu son classement par ordre d'adaptation.

Pour un problème de maximisation, nous classons les individus selon l'ordre croissant des valeurs de la fonction objectif. Ainsi, le plus mauvais individu (c'est à dire, celui qui possède la plus petite valeur de la fonction objectif) prend le $n^{\circ}1$ ainsi de suite. Pour un problème de minimisation, nous classons les individus selon l'ordre décroissant. La nouvelle population sera constitué de cet ensemble d'individus ordonnés, en utilisant des probabilités indexées sur les rangs des individus.

$$Pro\ \ de\ \ sélection(parent_i) = \frac{rang(parent_i)}{\sum_{j \in pop} rang(parent_j)}$$

Sélection par la roulette Dans un problème d'optimisation de maximisation, on associe à chaque individu i une probabilité de sélection ($prob_i$) proportionnelle à sa valeur de fitness F_i de la fonction objectif.

$$Prob_i = \frac{F_i}{\sum_{j \in pop} F_j}$$

Pour un problème de minimisation, on utilise une probabilité de sélection pour un individu i égale à $\frac{1-prob_i}{N-1}$

Sélection aléatoire La sélection se fait aléatoirement, uniformément et sans intersection de la valeur d'adaptation, chaque individu a donc une probabilité uniforme ($\frac{1}{N}$) d'être

sélectionner.

Selection par tournoi Cette méthode augmente les chances des individus de mauvaise qualité par rapport à leurs fitness de participer à l'amélioration de la population. En effet, c'est une compétition entre les individus d'une sous- population de taille M ($M \leq N$) prise au hasard dans la population (M est fixé par l'utilisateur). L'individu de meilleure qualité par rapport à la sous- population sera considéré comme vainqueur et sera sélectionné pour l'application de l'opérateur de croisement.

4. ***Opérateur de croisement :*** Le croisement à pour but d'enrichir la diversité de la population en manipulant les composantes du chromosome. Le croisement se fait entre deux parents et génère deux enfants (progénitures). Après la sélection des deux individus, nous générons un nombre aléatoire $\alpha \in [0, 1]$, si $\alpha \leq P_x$ (P_x probabilité de croisement), nous appliquons l'opérateur de croisement sur le couple.
   Les opérateurs de croisement les plus utilisés sont les opérateurs à un point et les opérateurs à deux points.
   Exemple 1 : Opérateur de croisement à un point. L'opérateur de croisement à un point

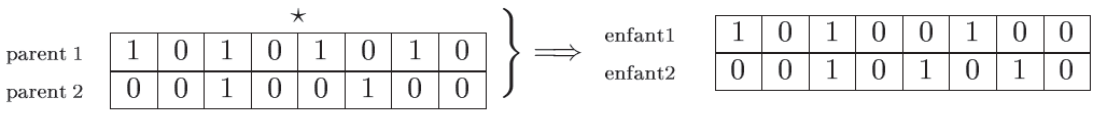

FIGURE 2.10 – L'opérateur de croisement à un point

.

consiste à partager les deux parents en deux parties tel que :
L'enfant 1 est composé de la première partie du parent 1 et la deuxième partie du parent 2.
L'enfant 2 est composé de la première partie du parent 2 et la deuxième partie du parent 1.

Exemple 2 : Opérateur de croisement à deux points. l'opérateur de croisement à deux points consiste à fixer deux positions tel que l'enfant 1 est constitué du parent 1 mais en remplaçant la partie entre les deux positions du parent 1 par celle du parent 2 et l'enfant 2 sera déterminer de la même manière mais en inversant les rôles des deux parents.

5. ***Opérateur de mutation :*** Il apporte aux algorithmes génétiques l'alia nécessaire à une exploration efficace de l'espace.

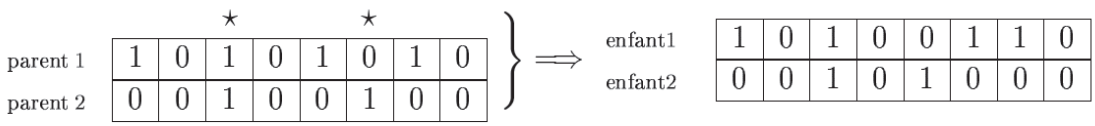

FIGURE 2.11 – L'opérateur de croisement à deux points

.

L'opérateur de mutation est utilisé avec une probabilité P_m. Dans les algorithmes génétiques à codage binaire, cette probabilité s'effectue sur les gènes en échangeant sa valeur de 0 à 1 ou de 1 à 0.

Exemple : Opérateur de Mutation d'un bit.

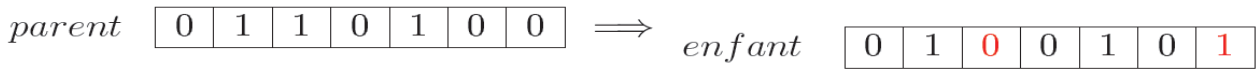

FIGURE 2.12 – Opérateur de mutation d'un bit

.

6. ***L'évaluation :*** Comme dans la nature, certain individus sont plus adaptés que d'autres, d'ou la nécessité de faire évaluer ces individus par une fonction d'évaluation ou fonction fitness, cette valeur d'adaptation est utilisée lors des opérations de selection afin de choisir les individus amenés à se reproduire par des croisement ou être utilisé par d'autres opérateurs génétiques.

– **B. La selection Pareto :**

La selection Pareto utilise la relation de dominance pour affecter des rangs aux individus de la population comme par exemple la technique de ranking [Goldberg, 1989] et fut utilisée dans l'algorithme NSGA [Srinivas et Deb, 1995] .

Goldberg propose une procédure itérative pour calculer ce rang : initialement, tous les individus non dominés reçoivent le rang 1 et ils sont temporairement retirés de la population, puis les nouveaux individus non dominés reçoivent le rang 2 avant d'être à leurs tour retirés de la population, ce processus s'itère tant qu'il y a des individus, le rang de l'individu dans la population

est sa valeur d'adaptation.

- **C. L'élitisme :**

  L'utilisation de l'élitisme dans un algorithme génétique revient à [De Jong, 1975]. L'élitisme permet de conserver les solutions non dominées ou les meilleures solutions tout au long du processus de recherche dans un ensemble appelé *archive*.

  Dans le cadre des problèmes multiobjectifs la technique élitiste prend deux adaptations :

  - La première adaptation regroupe les algorithme fondés sur les travaux de De Jong. L'archive comprend k meilleures solutions [Tamaki et al., 1994] et [Deb et al., 2000], mais l'inconvénient de cette approche est qu'elle perdent un grand nombre de meilleures solutions à cause de la taille de l'archive.
  - La seconde adaptation comprend les récentes approches apparues [Ishibuchi et Murata, 1996], [Parks et Miller, 1998] et [Zitzler et Thiele, 1999], ces approches utilisent une archive externe dont les individus sont mise à jour d'une génération à l'autre durant toute la recherche afin de conserver que les meilleurs éléments et cela en utilisant l'opérateur de sélection.

- **D. La diversification :**

  Maintenir la diversité de la population d'un algorithme génétique ou plus général, d'un algorithme évolutionnaire permet d'éviter que l'espace de recherche soit restreint à une petite zone ne contenant pas sûrement la solution optimale. Dans le cas, des problèmes multiobjectifs, la diversité permet d'explorer la totalité du front de Pareto.

  Pour maintenir cette diversité, des techniques ont été développées afin d'influencer sur la sélection de certains individus et permettre la diversification dans la population.

  Nous allons présenter dans ce qui suit les techniques les plus couramment utilisées dans les algorithmes évolutionnaires :

1. **Le sharing :** Consiste à modifier la valeur du coût d'un individu, cette valeur sera utilisée par l'opérateur de selection. Pour éviter qu'un grand nombre d'individus ne se concentrent autour d'un même point, il faut pénaliser la valeur d'adaptation en fonction du nombre d'individus au voisinage du regroupement ; plus les individus sont regroupés, plus leurs valeurs d'adaptation est faible. Dans la pratique, on ouvre un domaine autour d'un individu puis on calcule les distances entre les individus contenus dans ce domaine.

   Les bornes du domaine :

   Soient σ_{share} : distance maximale.

   d(i,j) : la distance séparant l'individu i et j.

   F(i) : valeur d'adaptation d'un individu $i \in P$(population).

$$F(i) = \frac{F'(i)}{\sum_{j \in P} sh(d(i,j))}$$

où

$$sh(d(i,j)) = \begin{cases} 1 - (\frac{d(i,j)}{\sigma_{share}})^2, si & \text{d(i,j)}< \sigma_{share}\,; \\ 0, sinon & . \end{cases}$$

2. **La réinitialisation :** Ce mécanisme permet d'introduire régulièrement des nouveaux individus générés aléatoirement ce qui permet parfois d'explorer des nouvelles zones de recherche.

3. **Le crowding :** Il consiste à déterminer un representant pour chaque niche découvert (ensemble des solutions) seuls les individus représentants participeront aux différentes étapes de l'algorithme (selection, mutation et croisement).

2.5.3 Quelques algorithmes génétiques :

Le Non Sorting Dominated genetic Algorithm (NSGA) :

Cette méthode a été proposée par [Srinivas et Deb, 1995] appliquée au problème multiobjectif. L'approche utilise la dominance au sens de Pareto pour déterminer les solutions non-dominées. NSGA vise à classer la population en différentes classes en fonction du niveau de dominance des solutions. Le tri de la solution est effectué par étape. Initialement, on détermine l'ensemble des solutions non dominées dans la population, cet ensemble forme la première frontière Pareto puis ces solutions sont écartées de la population, et recommence cette procédure avec le reste des individus de la population jusqu'à ce que toute la population soit triée. Pour la première frontière de Pareto, la valeur du fitness ou la valeur d'adaptation est $F_1 = N$, N taille de la population, pour les frontières supérieures la valeur d'adaptation sera dégradée. Afin de maintenir la diversité dans la population, il est nécessaire d'appliquer le sharing sur certain individus.
[Srinivas et Deb, 1995] utilisent la distance génotypique calculée dans l'espace des décisions pour corriger la valeur d'adaptation. Pour un ensemble de taille q de solutions de la même frontière de Pareto, la distance est définie entre deux solution i et j :

$$d(i,j) = \sqrt{\sum_{k=1}^{q} (\frac{x_k^i - x_k^j}{x_k^{max} - x_k^{min}})^2}$$

Le Strength Pareto evolutionary algorithm (SPEA)

Cette approche a été proposée par [Zitzler et Thiele, 1999] pour résoudre les problèmes d'optimisation multicritères et c'est un algorithme élitiste, SPEA maintien une archive externe contenant les meilleurs fronts de compromis rencontrés durant la recherche. On peut résumer les différentes caractéristiques de cet algorithme :

- Utilisation du concept de Pareto pour comparer les solutions.
- Un ensemble de solutions Pareto optimales est maintenu dans une population appelé **archive**.

– La valeur d'adaptation de chaque individu est calculée par rapport aux solutions de l'archive.
– Les solutions de l'archive participent à la selection.
– Une technique de clustering est utilisée pour réduire la taille de l'archive.

Déroulement de l'algorithme

La première étape consiste à créer une population initiale (générer des individus aléatoirement) sachant que l'archive externe au départ est vide puis sera remplie d'individus non dominés de la population.
À chaque itération, on retrouve les étapes classiques selection, mutation et croisement, puis les nouveaux individus non dominés sont rajoutés à l'archive et les individus de l'archive qui sont dominés par les nouveaux sont supprimés et si la taille de l'archive excède une taille fixée au départ, on applique une technique de clustering afin de garder que les meilleurs individus.
Calcul de la valeur d'adaptation : SPEA utilise la dominance afin de détecter les niches d'individus et comme les individus de l'archive participeront eux aussi à la selection leurs valeurs d'adaptation est calculée.
Le calcul de la valeur d'adaptation s'effectue en deux temps :

1. On attribut la valeur de dureté s aux individus de l'archive externe. Cette valeur est calculée comme suit :
$$s(i) = \frac{|\{j \backslash j \in P_t \bigwedge f(i) \leq f(j)\}|}{|P_t| + 1}$$
La valeur d'adaptation pour les individus de l'archive externe $\bar{P}$($i \in \bar{P}$ est : F(i)=s(i)).
2. Pour un individu j de la population courante $j \in P_t$, la valeur d'adaptation est la somme des valeurs de dureté des individus de l'archive $\bar{P}$ dominant j :
$$F(j) = 1 + \sum_{i \in \bar{P} \bigwedge f(i) \leq f(j)}$$
Le 1 est ajouté afin d'éviter que des individus de $\bar{P}$ aient une valeur d'adaptation plus grande que certain individus de P_t

Réduction par la technique de clustering : Dans certain cas des problèmes multicritères, la taille de l'archive externe est très grande, on utilise une technique de clustering afin alléger l'ensemble des individus tout en conservant des représentants. Cela, afin d'éviter que la recherche se focalise sur des zones déjà explorées.

La technique de clustering procède comme suit : au début, chaque individu constitue son propre groupe, puis on fusionne deux à deux les groupes les plus proches en terme de distance, cette étape est itérée jusqu'à on obtient le nombre de groupes désiré. Après, on choisit un représentant pour chaque groupes, c'est ce dernier qui sera gardé et les autres individus du groupe sont supprimés.
Remarque : L'inconvénient de cette méthode est que sa fonction d'évaluation dépend de la taille de l'archive choisie par l'utilisateur.

Le Non Sorting Dominated genetic Algorithm II (NSGA-II)

NSGA-II a été présenté par [Deb et al., 2000], ces derniers tentent de résoudre toutes les critiques et les manques faites sur NSGA : complexité, non élitisme et l'utilisation de sharing, cet algorithme intègre un opérateur de selection basé sur le calcul de la distance de crowding.

Déroulement de l'algorithme

NSGA-II est un algorithme élitiste, il n'utilise pas d'archive externe mais à chaque nouvelle génération, les meilleurs individus rencontrés sont conservés. Afin de comprendre le déroulement de cet algorithme plaçons nous à la génération t :
$\acute{A}$ la génération t : P_t et Q_t deux populations qui coexiste.
P_t : contient les meilleurs individus rencontrés jusqu'à la génération t.
Q_t : contient les autres individus des phases précédentes de l'algorithme.
Cet algorithme s'opère en trois phases :

La première phase : Créer la population $R_t = P_t \bigcup Q_t$ et appliquer une procédure de ranking pour identifier les différents fronts F_j de solutions non dominées (les meilleures solutions ou individus se trouvent dans les $1^{\texttt{ier}}$ fronts).

La deuxième phase : Construire une nouvelle population P_{t+1} contenant les N meilleurs individus de R_t, on utilise la technique de crowding pour identifier les représentants afin de remplir P_{t+1}, sachant que P_{t+1} contient N individus. On procède comme suit tant que $N - |P_{t+1}| \neq \emptyset$.

La troisième phase : Remplir la population Q_{t+1}, en utilisant les opérateurs de selection, mutation et croisement sur la population P_{t+1} et insérer les descendants dans Q_{t+1}.

La distance de crowding **:** Le calcul de valeur d'adaptation pour NSGA-II décrit dans [Deb et al., 2000] ne sert pas uniquement pour la sélection des opérateurs de croisement et de mutation, mais intervient aussi dans la sélection des individus à inclure dans P_{t+1} (la population contenant les élites). C'est donc une phase importante pour laquelle les auteurs de NSGA-II ont développé une méthode particulière : *la distance de crowding.*
La distance de crowding d_i d'un point particulier i se calcule en fonction du périmètre de l'hypercube ayant comme sommets les points les plus proches de i sur chaque critère. Un algorithme de calcul de la distance de crowding est détaillé dans [Deb, 2002]. Cet algorithme est de complexité O(MN log(N)), où M est le nombre de critères du problème et N le nombre d'individus à traiter. Une fois que les d_i sont calculées, on les trie par ordre décroissant et on sélectionne les individus possédant la plus grande valeur de crowding.

2.5.4 Les méthodes hybrides

Bien qu'au début, la plupart des chercheurs n'y accordaient peu d'intérêt aux méthodes hybrides, des meilleures solutions ont été trouvées en utilisant ces méthodes pour résoudre plusieurs problèmes

d'optimisation combinatoire, cela a rendu les méthodes hybrides plus populaire et à attirer l'attention des nouveaux chercheurs.

L'hybridation est une combinaison entre deux ou plusieurs méthodes, elles ont été divisées en deux groupes :

- Les métaheuristiques hybrides impliquant une combinaison entre deux ou plusieurs métaheuristiques.
- Les méthodes hybrides impliquant une combinaison d'une méthode exacte et d'une métaheuristique.

Les métaheuristiques hybrides

On peut classifier différentes hybridations entre métaheuristiques selon la taxonomie de Talbi. Cette classification permet de comparer les métaheuristiques hybrides de façon qualitative. Pour plus de détail, Talbi [Talbi, 2002] fournit en détail les classifications de métaheuristiques hybrides. Dans la littérature, la recherche locale, le recuit simulé et les algorithmes évolutifs ont déjà été hybridés avec succès dans plusieurs applications.

L'hybridation permet d'améliorer la performance des algorithmes et de combler certaines lacunes. Dans notre travail, on s'intéresse à l'hybridation entre une méthode de recherche locale (recherche tabou) et des algorithmes génétiques (NSGA II et SPEA). Cette hybridation sera détaillée dans les prochains chapitres.
En général, l'hybridation se fait de deux manières :

- **L'hybridation parallèle :** Cette hybridation implique un ensemble de métaheuristiques complètes qui travaillent en parallèle et coopèrent pour trouver l'optimum recherché.
- **L'hybridation séquentielle :** Dans cette hybridation, un premier algorithme est exécuté afin de trouver l'espace des solutions, ces résultats sont ensuite utilisés par un autre algorithme comme sa donnée d'entrée. C'est ce type d'hybridation que nous utiliserons dans notre travail.

CHAPITRE 3

POSITION DE PROBLÈME ET ÉTAT DE L'ART

Ce chapitre est consacré à la présentation du problème d'ordonnancement bicritère à machines parallèles uniformes et à donner quelques exemples sur des problèmes multicritères déjà traités dans la littérature.

3.1 Présentation du problème $Q/r_i, d_i/C_{max}, L_{max}$

Considérons un problème à machines parallèles défini de la façon suivante : un ensemble de n travaux doivent être exécutés sur m machines parallèles uniformes. Chaque travail J_i est défini par un temps opératoire absolu p_i, une date de début au plus tôt r_i, une date due d_i et doit être totalement exécuté par une des machines. Les machines qui exécutent un travail à la fois, ne sont pas nécessairement identiques mais uniformes ce qui implique que chaque machine M_j possède une vitesse V_j. Le temps opératoire d'un travail dépend de la vitesse de la machine sur laquelle il est ordonnancé, donc le temps opératoire du travail J_i sera défini par $p_{ij} = p_i/V_j$ lorsqu'il est exécuté sur M_j. Notre objectif est de déterminer un ou plusieurs ordonnancements des travaux sur les machines, i.e. déterminer pour chaque travail J_i la machine sur laquelle il sera affecté et sa date de fin C_i. Cet ordonnancement est effectué dans le but de minimiser deux critères. Le premier est le makespan, il correspond à la date de fin de tous les travaux et il est défini par $C_{max} = \max_{\overline{1,n}}\{C_i\}$. Le second critère est le retard algébrique et il est défini par $L_{max} = \max_{\overline{1,n}}\{C_i - d_i\}$. En utilisant la notation standard à trois-champs de [TKindt et Billaut, 2006], ce problème se note $Q/r_i, d_i/C_{max}, L_{max}$.

La minimisation du makespan vise à diminuer les délais de la production, donc cela peut conduire à l'augmentation de la productivité des ateliers, puisque les machines seront disponibles le plutôt possible pour une nouvelle exécution. Ainsi, ce critère représente une mesure interne d'efficacité d'une entreprise. Par contre, le retard algébrique reflète une mesure externe d'efficacité. En effet, ce dernier critère peut modéliser une situation où l'entreprise doit respecter des dates de livraison autorisées par

le client. Ainsi, la minimisation de ce critère augmente la qualité de l'entreprise vis-à-vis du client.

Le problème $Q/r_i, d_i/C_{max}, L_{max}$ est un problème bicritère pour lequel il n'existe pas forcément de solution qui minimise simultanément les deux critères. Dans ce travail, nous nous intéressons à déterminer les optima de Pareto stricts.

Le problème $Q/r_i, d_i/C_{max}, L_{max}$ est $\mathcal{NP} - difficile$ au sens fort parce que le problème $Q/r_i/C_{max}$ l'est.

3.2 Modélisation mathématique

Nous présentons une modélisation du problème $Q/r_i, d_i/C_{max}, L_{max}$ sous la forme d'un programme linéaire en nombres entiers. Les variables du modèle sont les suivantes :

$$x_{ijk} = \begin{cases} 1, & \text{si le travail i est affecté en position k sur la machine j ;} \\ 0, & \text{sinon.} \end{cases}$$

t_{kj} : est la date de début de traitement d'un travail en position k sur la machine j.

Avec i=1,...,n ; j=1,...,m ; k=1,...,n.

$$
\begin{cases}
min(C_{max}, L_{max}) & \\
\sum_{j=1}^{m}\sum_{k=1}^{n} x_{ijk} = 1, & i = 1, ..., n; \quad (1) \\
\sum_{i=1}^{n} x_{ijk} \leq 1, & k = 1, ..., n,\ j = 1, ..., m \quad (2) \\
t_{kj} - \sum_{i=1}^{n} r_i x_{ijk} \geq 0, & k = 1, ..., n,\ j = 1, ..., m \quad (3) \\
t_{kj} + \sum_{i=1}^{n} \frac{p_i}{V_j} x_{ijk} \leq t_{k+1j}, & k = 1, ..., n,\ j = 1, ..., m \quad (4) \\
t_{kj} + \sum_{i=1}^{n} (\frac{p_i}{V_i} - d_i) x_{ijk} \leq L_{max}, & k = 1, ..., n,\ j = 1, ..., m \quad (5) \\
t_{nj} + \sum_{i=1}^{n} \frac{p_i}{V_i} x_{ijn} \leq C_{max}, & j = 1, ..., m \quad (6) \\
x_{i,j,k} \in \{0, 1\}, & \forall i = 1, ..., n\ \forall\ j = 1, ..., m\ \forall\ k = 1, ..., n \\
t_{kj} \geq 0, & \forall k = 1, ..., n\ \forall\ j = 1, ..., m \\
C_{max} \geq 0,\ L_{max} \in R &
\end{cases}
$$

- La contrainte (1)impose le traitement de tous les travaux ;
- La contrainte (2) impose le traitement d'un seul travail par position quelque soit la machine ;
- La contrainte (3) empêche le démarrage d'un travail avant sa date de disponibilité ;
- La contrainte (4) impose au travail en position k+1 de commencer son exécution après la fin d'exéction du travail en position k ;
- La contrainte (5) indique que la tardivité de chaque travail doit être $\leq L_{max}$.
- La contrainte (6) indique que la date de fin de traitement du dernier travail doit être $\leq C_{max}$.

Exemple On considère un problème pour lequel le nombre de travaux $n = 8$ et le nombre de machine $m = 2$.

Les caractéristiques des travaux sont :

i	1	2	3	4	5	6	7	8
r_i	2	10	3	0	0	5	15	21
p_i	5	3	2	4	1	3	6	4
d_i	15	14	8	16	20	10	28	25

Les vitesses des machines sont données par :

j	1	2
V_j	1	2

Une solution est obtenue en rangeant les travaux par ordre croissant de leurs dates de disponibilité r_i et en essayant toujours de ne pas dépasser leurs dates échues d_i. $\{J_4, J_5, J_1, J_3, J_6, J_2, J_7, J_8\}$ est la liste des travaux triés par ordre croisant de leurs dates de disponibilité.

$$\begin{cases} C_{max} = 23 \\ L_{max} = -2 \end{cases}$$

- Les travaux $\{J_4, J_5, J_1, J_3, J_6, J_7\}$ sont ordonnancés sur la machine M_1.
- Les travaux $\{J_3, J_6, J_2, J_8\}$ sont ordonnancés sur la machine M_2.

La valeur $C_{max} = 23$ représente la date de fin de traitement du dèrnier travail ordonnancé et $L_{max} = -2$ est la plus grande différence entre la date de fin de traitement d'un travail et da date échue.

3.3 État de l'art

Peu de travaux relatifs à l'étude du problèmes $Q/r_i, d_i/C_{max}, L_{max}$ ont été trouvés dans la littérature. Lorsqu'il s'agit de minimiser uniquement le critère C_{max},
[Dell'Amico et Martello, 1995] proposent une méthode exacte de type procédure par séparation et évaluation pour résoudre le problème $P//C_{max}$, qui résout dans les cas les plus favorables des instances qui comportent jusqu'à 10000 travaux. Le problème à machines parallèles non reliées, noté $R//C_{max}$, a été résolu par [Van de Velde, 2005] et [Martello et al., 1997] par une méthode exacte et [$Fanjul - Peyro$ et Ruiz, 2010] ont proposé une métaheuristique basée sur l'application d'une méthodologie gloutone itérative introduite par [Ruiz et Stutzle, 2007]. Plusieurs travaux ont été conduits pour le problème de minimisation du critère L_{max}. [Carlier, 1987] avait proposé un algorithme de type procédure par séparation et évaluation pour résoudre le problème $P/r_i, d_i/L_{max}$. [Carlier et Pinson, 1998] ont également proposé un algorithme très efficace pour le résoudre et plus connu sous le nom de *Jackson's Pseudo Préemptive Schedule (JPPS)*. Cet algorithme est une extension de l'algorithme précédent. Dans le nouvel algorithme les auteurs autorisent la préemption des tâches mais plusieurs machines peuvent partager l'exécution d'une tâche à un même instant, tout comme une machine peut ordonnancer plusieurs tâches au même instant. [Haouari and Gharbi, 2003] ont proposé une borne inférieure dont le principe se base sur le calcul d'un flot maximum sur un graphe. Cette nouvelle borne inférieure domine celle proposée par Carlier et Pinson. Ils ont également proposé une procédure par séparation et évaluation qui résout des instances comportant jusqu'à

300 travaux. Un problème équivalent, qui correspond au problème $P/r_i, q_i/C_{max}$ a été traité par [Tercinet, 2004] et [Tercinet et al., 2004] qui ont proposé une procédure par séparation et évaluation avec des bornes inférieures qui domine celles proposées par Haouari et Gharbi. Une extension du problème $P/r_i, q_i/C_{max}$ qui est défini par $R2/r_i, q_i/C_{max}$ a été étudiée par [Lancia, 2000], qui a proposé une heuristique et une méthode exacte pour sa résolution.

CHAPITRE 4

MÉTHODES PROPOSÉES

Dans ce chapitre , nous nous somme intéressé à la résolution du problème de minimisation de makespan et du retard algébrique à machines parallèles uniformes $Q/r_i, d_i/C_{max}, L_{max}$ par des méthodes approchées. Nous résolvons ce problèmes grâce à plusieurs algorithmes : deux heuristiques génétiques (NSGA-II et SPEA), trois versions de recherche Tabou et deux algorithmes mémétiques.

4.1 Le problème $Q/r_i, d_i/C_{max}, L_{max}$

Le problème que nous considérons est l'ordonnancement bicritère à machines parallèles uniformes. Bien qu'il est bien détaillé dans le précédent chapitre, nous voyons nécessaire de le représenter.

On considère un ensemble de n travaux, ces travaux doivent être ordonnancés sur m machines parallèles uniformes, chaque travail J_i est caractérisé par une date de disponibilité r_i et une date échue d_i (due date) et chaque machine M_j possède une vitesse V_j. L'objectif de cet ordonnancement est la minimisation simultanément de la date de fin de traitement (makespan) et du retard algébrique.

Notre problème étant bicritère, la recherche d'une solution qui minimise simultanément les deux critères revient à déterminer l'ensemble des optima de Pareto stricts.

4.2 Méthodes proposées

4.2.1 Approches génétiques pour le problème $Q/r_i, d_i/C_{max}, L_{max}$

Les algorithmes génétiques sont beaucoup répondus dans la résolution des problèmes d'optimisation multiobjectifs, ils fournissent des résultats satisfaisants. Ces algorithmes permettent de déterminer une approximation de l'ensemble des optima de Pareto, ils se basent sur un ensemble de solutions contrairement aux autres méthodes tel que les méthodes constructives ou encore les méthodes de la

recherche locale.

Les algorithmes que nous allons utilisés dans notre mémoire sont NSGA II [Deb, 2002] et SPEA [Zitzler et Thiele, 1999], nous présentons l'adaptation de ces deux algorithmes génétiques au problème d'ordonnancement bicritère.

Nous présentons dans la suite de cette section, les principales caractéristiques communs de nos deux algorithmes.

4.4.1.1 Codage des chromosomes

Dans ces algorithmes, nous utiliserons un codage directe des solutions.

Soit pour un ordonnancement $s \in S$, une fonction g : $\mathbb{N} \times S \to \mathbb{N}$ où g(i,s) : la position du travail J_i dans s sur la machine sur laquelle il est ordonnancé.

Soit f une fonction définie par $f : \mathbb{N} \times \mathbb{N} \times S \to \mathbb{N}$ telle que :

$$f(i,j,s) = \begin{cases} g(i,s), & \text{si le travail } J_i \text{ est traité sur la machine } M_j \text{ dans s}\,; \\ 0, & \text{sinon.} \end{cases}$$

Par conséquent, le chromosome C(s) associé à l'ordonnancement s est une matrice définie par $C(s) = [f(i,j,s)]_{1 \le i \le n, 1 \le j \le m}$

4.4.1.2 Opérateur de croisement

Nous décrivons maintenant les opérateurs génétiques utilisés en commençant par l'opérateur de croisement. Souvent cet opérateur permet à l'algorithme génétique d'intensifier la recherche en prenant des dispositifs qui permettent de créer de meilleures progénitures. Pour notre problème, nous avons décidé d'utiliser un croisement à un point qui procède comme suit :

Soient C et C' deux chromosomes, appelés parents, et soit ρ un entier généré aléatoirement en utilisant une loi uniforme entre 1 et n. Deux progénitures O et O' sont définies par $O = [f_O(i,j)]_{1 \le i \le n, 1 \le j \le m}$ et $O' = [f_{O'}(i,j)]_{1 \le i \le n, 1 \le j \le m}$ comme suit :

$$f_O(i,j) = \begin{cases} f(i,j,s), & \text{si i} \le \rho\,; \\ f(i,j,s'), & \text{sinon} \end{cases}$$

$$f_{O'}(i,j) = \begin{cases} f(i,j,s'), & \text{si i} \le \rho\,; \\ f(i,j,s), & \text{sinon} \end{cases}$$

avec s et s' des ordonnancements qui correspondent aux chromosomes C(s) et $C(s')$. Notons que les progénitures O et O' peuvent ne pas correspondre à des solutions faisables, du moment que pour la même machine M_j deux travaux peuvent avoir la même valeur de position, $i.e. f(i,j,s) = f(k,j,s')$ pour certain $i \le \rho \le k$. Par conséquent une telle machine doit être partiellement réordonnancée en utilisant l'heuristique de réparation suivante :

Soit M_j une machine dans O telle que $\exists i,k, f_O(i,j) = f_O(k,j)$. Nous considérons que ℓ est la position courante, et que initialement $\ell = 1$. Rénumeroter consécutivement et en commençant par 1 toutes les valeurs de $f_O(i,j) \neq 0$ avec $i \le \rho$, tout en tenant compte de l'ordre relatif à $f_O(i,j)$. Nous procédons de la même façon pour les valeurs $f_O(i,j) \neq 0$ avec $i > \rho$.

À la fin, nous aurons deux travaux J_i et J_k tel que $f_O(i,j) = f_O(k,j) = 1$, etc. Maintenant nous

devons décider lequel des deux travaux J_i et J_k sera ordonnancé en première position dans la séquence. Pour se faire nous appliquons la règle suivante de l'heuristique :

J_i précède J_k si	$d_i \leq r_k$
ou	$r_i \leq r_k$ et $d_i \leq d_k$
ou	$r_i \geq r_k$ et $d_i \leq d_k$ et $r_k + p_k > r_i$
sinon J_k précède J_i.	

Supposons que J_i précède J_k. Dans ce cas de figure, nous posons $f_O(i,j) = 1$ et $f_O(u,j) = f_O(u,j) + 1$, $\forall u = 1, ..., \rho$ tel que $f_O(u,j) \geq \ell$. Si la règle précédente donne que J_k précède J_i, nous posons $f_O(k,j) = 1$ et $f_O(u,j) = f_O(u,j) + 1$, $\forall u = \rho + 1, ..., n$ tel que $f_O(u,j) \geq \ell$. Posons également $\ell = \ell + 1$ et soient J_i et J_k deux travaux $f_O(i,j) = f_O(k,j) = \ell$. Nous appliquons le même procédé décrit précédemment jusqu'à ce qu'il n y'ait plus deux valeurs $f_O(i,j)$ égales.

4.4.1.3 Opérateur de mutation

Cet opérateur permet d'interchanger deux travaux entre deux machines.
Considérons un individu O et deux entiers distincts π_1 et π_2 générés aléatoirement en utilisant une loi uniforme entre 1 et n. L'individu mute O est obtenu en interchangeant les valeurs $f_O(\pi_1, j) = f_O(\pi_2, j)$ pour toutes les machines M_j. Ceci est equivalent à interchanger les deux travaux $J\pi_1$ et $J\pi_2$ même s'ils ne sont pas ordonnancés sur la même machine. Lorsqu'on utilise une heuristique qui se base sur une recherche de voisinage, on doit prouver qu'elle converge vers des solutions optimales. Dans le cas de nos algorithmes génétiques et notre problème d'énumération ceci est equivalent à prouver que, en commençant par une population initiale quelconque, ils sont capables de trouver l'ensemble des optima de Pareto stricts si on leurs alloue un nombre infini d'itérations.

Théorème 4.2.1. *Pour un individu faisable donné C(s), il est possible de calculer pour le problème $Q/r_i, d_i/C_{max}, L_{max}$ un optimum de Pareto en appliquant un nombre de fois fixé des opérateurs génétiques définis précédemment.*

`Preuve`

Pour prouver ce résultat il suffit de montrer que, en partant d'un individu C(s), nous pouvons atteindre n'importe quelle solution faisable en appliquant des opérateurs génétiques. Nous considérons qu'à partir d'un ordonnancement s code avec un chromosome C(s) nous voudrions obtenir une autre solution s'. Nous exhibons les conditions sous lesquelles l'algorithme peut converger de s vers s'. Premièrement, nous rappelons que les opérateurs génétiques sont utilises avec des probabilités. i.e. pour une instance donnée, l'opération de croisement est appliquée sur deux parents seulement si le nombre généré aléatoirement entre 0 et 1 appartient à la probabilité qui est donnée lors de l'initialisation de l'algorithme. Supposons que après un nombre suffisamment grand de generations l'opération de croisement n'a pas été execute sur C(s).
Réciproquement, supposons que l'opération de mutation est appliquée à chaque generation sur cet

individu. En outre, supposons que d'une generation à une autre C(s) reste present dans la population et est toujours sélectionne pour l'opération de mutation. Sous cette condition, nous constatons que l'application d'une série d'opérations de mutation peut conduire à une solution s'. L'opérateur de mutation consiste à interchager deux travaux dans l'ordonnancement s. Donc, pour une instance donnée, si le travail J_3 dans s n'est pas sur la même machine et sur la même position que dans s', nous effectuons juste un échange de J_3 dans s afin de rendre s plus proche de s'. Nous réitérons ce procède jusqu'à ce que s devienne s'. Ainsi, nous avons montre d'un point de vue statistique, nous pouvons faire converger une solution initiale vers un optimum de Pareto strict.

Il est important de mentionner que le théorème précédent n'est pas suffisant pour prouver que l'utilisation des opérateurs génétiques détermine l'ensemble des optima de Pareto stricts. Ceci dépend du nombre de vecteurs de critères non dominés et de l'utilisation d'une archive externe pour stocker ces solutions. En fait, dans le cas ou un algorithme génétique n'emploie pas une telle archive, l'ensemble des optima de Pareto stricts ne peut pas être entièrement calculé si sa taille excède le nombre de ses individus dans la population. Dans ce cas particulier, nous déterminons seulement un sous ensemble.

4.4.1.4 Le Non Sorting Dominated Genetic Algorithm II (NSGA-II)

[Deb et al., 2000] ont proposé cet algorithme, il se base sur un schéma élitiste et sur la notion de "non-dominated sorting" qui fonctionne comme suit.
Soient :

- P : une population de N chromosomes, chacun d'entre eux représentent une solution.
- F^j le $j^{\text{ième}}$ ensemble non dominé dans P, ie, F^j est un ensemble de solutions non dominées dans $P - \bigsqcup_{k=1}^{j-1} F^{j-1}$ avec $F^0 = \emptyset$

L'idée principale de NSGA-II est de préserver d'une génération à une autre les meilleurs fronts F_j qui affecte à chaque chromosome une valeur de fitnesse f qui dépend du front sur lequel le chromosome correspondant appartient et sur la répartition du chromosome sur l'espace des solutions.

principaux facteurs de l'algorithme Soient :
P_0 : la population initiale à N chromosomes (solutions).
$Q = \emptyset$: ensemble des individus de rang inférieur à ceux dans P durant toutes les générations.
Les opérateurs de selection utilisés sont :

- Opérateur de selection basé sur la distance de crowding.
- Opérateur de sélection basé sur crowded tournoi utilise aussi la distance de crowding.

La distance de crowding : est une estimation de la densité des solutions autour d'une solution $s \in F^j$.
Soit π_s la valeur de la distance de crowding d'un ordonnancement s tel que $C(s) \in F^j$.

Soient s_1 et s_2 deux ordonnancement tel que :

$$\nexists s'/C_{max}(s_1) < C_{max}(s') < C_{max}(s)$$

ou

$$C_{max}(s) < C_{max}(s') < C_{max}(s_2)$$

et soient s_3 et s_4 tel que

$$L_{max}(s_3) < L_{max}(s) < L_{max}(s_4)$$

La valeur de la distance de crowding pour un ordonnancement s est définit comme suit :

$$\pi_s = \frac{C_{max}(s_2) - C_{max}(s_1)}{C_{max}^{max} - C_{max}^{min}} + \frac{L_{max}(s_2) - L_{max}(s_1)}{L_{max}^{max} - L_{max}^{min}}$$

avec :

$$C_{max}^{min} = min_{C(s')\in F^j}(C_{max}(s'))$$

$$C_{max}^{max} = max_{C(s')\in F^j}(C_{max}(s'))$$

$$L_{max}^{min} = min_{C(s')\in F^j}(L_{max}(s'))$$

$$L_{max}^{max} = max_{C(s')\in F^j}(L_{max}(s'))$$

π_s (la distance de crowding) : c'est le périmètre de l'hypercube ayant comme sommets les points les plus proches de l'ordonnancement s sur chaque objectif.

- La sélection basé sur la distance de crowding appliquée sur l'ensemble F^j consiste à sélectionner $(N - |P_{t+1}|)$ chromosomes $C(s) \in F^j$ ayant la plus grande valeur de distance de crowding.
- La sélection basé sur crowded tournoi utilise la distance de crowding quand il sélectionne un chromosome $C(s) \in P_{t+1}$

Soient $C(s)$ et $C(s'')$ deux chromosomes choisis aléatoirement dans P_{t+1} avec $C(s) \in P^k$ et $C(s) \in P^\ell$. Le chromosomes $C(s)$ est sélectionné au lieu de $C(s')$ si et seulement si

$$k < \ell \quad ou \quad (k = \ell \quad et \quad \pi_s > \pi_{s''})$$

Le figure 4.1 et l'algorithme 4.1 [Deb, 2002] illustrent le schéma général de fonctionnement de l'algorithme.

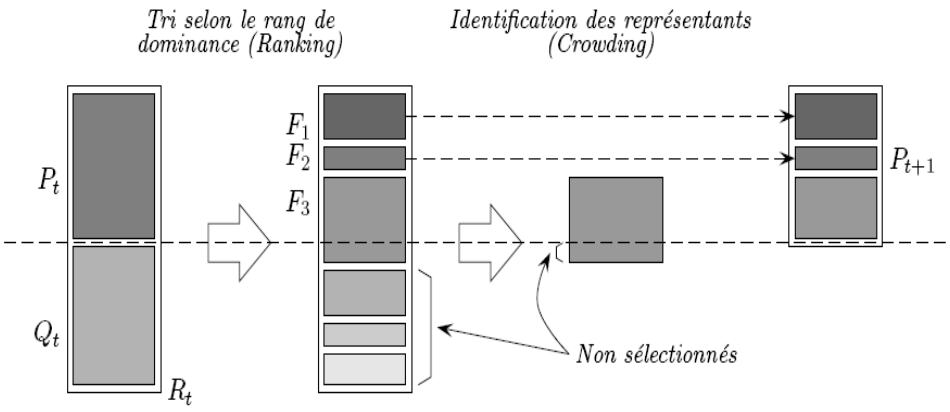

FIGURE 4.1 – Fonctionnement de NSGA-II

.

4.4.1.5 Le Strength Pareto Evolutionary Algorithm (SPEA)

La méthode SPEA implémenté par [Zitzler et Thiele, 1999] est une méthode élitiste, elle contient une archive externe contenant les meilleurs fronts de compromis rencontrés durant la recherche.

Déroulement de l'algorithme : Au début, elle crie une population initiale contenant d'individus générés aléatoirement et à chaque itération, on retrouve les principaux opérateurs génétiques (la sélection, mutation et croisement). Les nouveaux individus non dominés trouvés sont ajoutés à l'archive en supprimant les individus dominés par les nouveaux arrivés. La figure 4.2 et l'algorithme 4.2 [Zitzler et Thiele, 1999] illustrent le schéma général de fonctionnement de l'algorithme.

/* P_0, Q_0 : les deux populations initiale */
/* N : la taille de la population */
/* tmax : Le nombre maximum de générations*/
t=0 ;
TantQue ($t < tmax$)
    $R_t = P_t \cup Q_t$;
    Calcul des différents fronts F^j de la population R_t
    avec un algorithme de "ranking",
    qui permet de calculer le rang de Pareto de chaque individu ;
    $P_{t+1} = P_t$; $j = 1$;
    TantQue ($|P_{t+1}| + |F^j| < N$)
        $P_{t+1} = P_{t+1} \cup F^j$;
        $j = j + 1$
    FinTantQue
    Appliquer la sélection basée sur la distance de crowding sur F^j
    pour compléter P_{t+1} ;
    $i = 1$; $Q_{t+1} = \emptyset$;
    TantQue ($i \leq N/2$)
        Appliquer la séléction basée sur le crowded tournoi sur P_{t+1}
        pour sélectionner $\mathcal{C}$ et $\mathcal{C}'$;
        Appliquer l'opérateur de croisement avec la probabilité p_N^C
        sur $\mathcal{C}$ et $\mathcal{C}'$ pour créer $\mathcal{O}$ et $\mathcal{O}'$;
        Appliquer l'opérateur de mutation avec la probabilité p_N^M
        sur $\mathcal{O}$ et $\mathcal{O}'$;
        $Q_{t+1} = Q_{t+1} \cup \{\mathcal{O}\} \cup \{\mathcal{O}'\}$;
        $i = i + 1$;
    FinTantQue
    $t = t + 1$;
FinTantQue
Sortie :Front F_1 de l'ensemble $P_{tmax} \cup Q_{tmax}$

FIGURE 4.2 – Algorithme NSGA-II

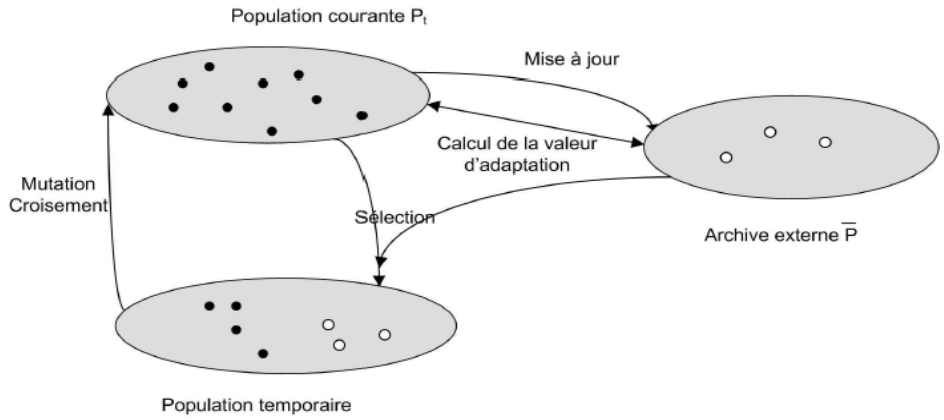

FIGURE 4.3 – Fonctionnement de SPEA

.

```
1   /* P_0 : Population initiale */
2   /* P̄ : Archive externe */
3   /* tmax : Le nombre maximum des générations */
4   t = 0)
5   TantQue (t < tmax)
6       Affecter une fitnesse pour chaque individu de P_t ∪ P̄
7       Appliquer la sélection par tournoi binaire avec ces valeurs de fitnesse,
8       l'opérateur de croisement et l'opérateur de mutation pour créer P_{t+1}
9       à partir de P_t ∪ P̄
10      Mise à jour de P̄ (suppression des individus dominés)
11      t = t + 1
12  FinTantQue
13  Sortie : P̄
```

FIGURE 4.4 – Algorithme SPEA

4.2.2 Approches Tabou pour le problème $Q/r_i, d_i/C_{max}, L_{max}$

Les algorithmes de recherche Tabou [Glover, 1986] ont démontré leur efficacité pour résoudre des problèmes d'ordonnancement. Dans le cadre d'une approche multicritère, on peut appliquer différentes techniques pour la détermination d'optima de Pareto. Nous nous sommes intéressés à trois approches pour la détermination de solutions non dominées [Bouibede et al., 2012] : une marche aléatoire, une combinaison linéaire des critères et une approche epsilon-contrainte.

```
1   /*nb − iteration : nombre d'itérations*/
2   /s : solution réalisable*/
3   /*TL : ensemble des solutions tabou*/
4   /*E : ensemble de solutions visitées*/
5   /*ND : ensemble des solutions non-dominées*/
6   Générer une solution initiale s en utilisant l'heuristique H
7   E = E ∪ s_init
8   s := s_init
9   tl = ∅ : liste tabou, indice = 0
10  TantQue (indice < nb − iteration) faire
11      | Construire le voisinage V(s)
12      Cherche meilleur − voisin v_best
13      s = v_best, TL = TL ∪ v_best (mise à jour de la liste tabou)
14      E = E ∪ v_best (mise à jour de l'ensemble des solutions visitées)
15      indice = indice + 1
16  FinTantQue
17  Construire l'ensemble des solutions non-dominées ND
```

FIGURE 4.5 – Approche tabou pour le problème $Q/r_i, d_i/C_{max}, L_{max}$

4.4.2.1 Les points communs

Les trois versions de la recherche Tabou ont les parties suivantes en commun : le codage d'une solution,la solution initiale, définition d'un voisin et la gestion de la liste Tabou.

Codage d'une solution Afin d'implémenter l'approche tabou, nous avons choisit d'utiliser un codage binaire des solutions. Soit un ordonnancement s de n travaux et m machines, la solution (ordonnancement) est une matrice binaire C(s) de n colonnes et m lignes tel que :

$$C(s) = \begin{cases} 1, & \text{si le travail } J_i \text{ est ordonnancé sur la machine } M_j\,; \\ 0, & \text{sinon.} \end{cases}$$

Exemple :

Considérons un ordonnancement s à 5 travaux et 2 machines dans lequel les séquencements des travaux sont, respectivement, (J_0, J_4, J_2) sur M_1 et (J_1, J_3) sur M_2, nous avons :

$$C(s) = \begin{pmatrix} 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 \end{pmatrix}$$

Le makespan correspond à la date de fin de traitement sur la dernière machine en exécution et le plus grand retard correspond à la plus grande différence entre la date de fin de traitement de chaque travail et sa date échue.

Solution initiale La solution initiale est obtenue par une heuristique appelée *heuristique par construction progressive*, notée H. Elle construit un ordonnancement de façon itérative. Elle commence par trier les machines par vitesse croissante. Les travaux sont ensuite triés dans deux listes différentes : la liste L_1 contient les travaux ordonnancés par date de début au plus tôt, r_i, croissante. La liste L_2 contient les travaux ordonnancés par date de fin au plus tard, d_i, croissante. En cas d'égalité si le tri est effectué par r_i croissant, les ex-aequo sont triés par d_i croissant et si le tri est effectué par d_i croissant les ex-aequo sont triés par r_i croissant.

A chaque itération H sélectionne le travail $L_1[1]$ de plus petit r_i et essaye de lui trouver une machine. Ensuite elle prend le travail $L_2[1]$ du plus petit d_k et elle vérifie s'il peut être ordonnancé sans dépasser sa date de fin impérative à la suite de l'affectation du travail $L_1[1]$. Si l'ordonnancement de $L_1[1]$ ne cause pas de dépassement alors H continue son fonctionnement en supprimant le travail $L_1[1]$ de la liste des travaux non ordonnancés et en essayant de trouver une machine pour le nouveau travail du plus petit r_i. Dans le cas contraire, cette heuristique essaye de trouver d'abord une affectation pour le travail $L_2[1]$ ensuite elle s'intéresse au travail $L_1[1]$. Une fois l'un des travaux ordonnancé, les deux listes L_1 et L_2 sont mises à jour. L'algorithme de cette heuristique est présenté dans la figure 4.13. L'exemple suivant illustre le fonctionnement de cette heuristique.

```
1   /*L1 : liste des travaux trier par ordre croissant des ri, L2 liste des travaux trier par ordre croissant des di*/
2   /*LV liste des machines trier par ordre croissant des Vj, Cij : date de fin du travail Ji sur la machine Mj*/
3   /*Ω : liste des travaux non ordonnancé, Cσ[j] : date de fin du dernier travail sur la machine Mj*/
4   Pour j = 1 à m faire
5       Cσ[j] = 0
6   FinPour
7   Pour i = 1 à n faire
8       Ci = ∞
9   FinPour
10  TantQue Ω ≠ ∅
11      existe = false, j = 1
12      /*ordonnancer le travail L1[1] sur une machine qui le finit au plus tôt*/
13      TantQue j ≤ m Faire
14          C_{L1[1]j} = max(r_{L1[1]}, Cσ[j]) + pL1[1] / LV[j]
15          SiC_{L1[1]} > C_{L1[1]j}
16          Alors C_{L1[1]} = C_{L1[1]j}
17          FinSi
18          j = j + 1
19      FinTantQue
20      Si C_{L1[1]} > d_{L1[1]}
21      Alors Fin :l'heuristique n'a pas trouvé de solution
22      Sinon
23          /*Vérifier si le travail L2[1] peut être ordonnancé */
24          TantQue (existe = false et j ≤ m)
25              C_{L2[1]j} = max(r_{L2[1]}, Cσ[j]) + pL2[1] / LV[j]
26              Si C_{L2[1]j} ≤ d_{L2[1]}, Alors existe = true
27              Sinon j = j + 1
28              FinSi
29          FinTantQue
30      FinSi
31      Si eixste = true Alors Ω = Ω − L1[1]
32      Sinon/* ordonnancer le travail L2[1]*/
33          j = 1
34          TantQue j ≤ m Faire
35              C_{L2[1]j} = max(r_{L2[1]}, Cσ[j]) + pL1[1] / LV[j]
36              SiC_{L2[1]} > C_{L2[1]j}
37              Alors C_{L2[1]} = C_{L2[1]j}
38              FinSi
39              j = j + 1
40          FinTantQue
41          Si C_{L2[1]} > d_{L2[1]}
42          Alors Fin :l'heuristique n'a pas trouvé de solution
43          Sinon Ω = Ω − L2[1]
44          FinSi
45      FinSi
46  FinTantQue
```

FIGURE 4.6 – L'heuristique H pour la solution initiale

Exemple

– Les données :

On considère un problème pour lequel le nombre de travaux $n = 8$ et le nombre de machine $m = 2$.
Les caractéristiques des travaux sont :

i	1	2	3	4	5	6	7	8
r_i	1	2	3	4	5	6	7	8
p_i	2	4	8	4	6	2	4	2
d_i	4	7	10	10	15	9	15	15

Les vitesses des machines sont données par :

j	1	2
V_j	1	2

– Préliminaire :

$L_1 = \{J_1, J_2, J_3, J_4, J_5, J_6, J_7, J_8\}$, $L_2 = \{J_1, J_2, J_6, J_3, J_4, J_5, J_7, J_8\}$
Les temps opératoires des travaux sur les deux machines dépendent des vitesses.

i	1	2	3	4	5	6	7	8
p_{i1}	2	4	8	4	6	2	4	2
p_{i2}	1	2	4	2	3	1	2	1

Dans cet exemple nous utilisons les notations suivantes : $C_{\sigma[1]}$ et $C_{\sigma[2]}$ désignent respectivement les dates de fin du dernier travail sur la machine M_1 et M_2. C_i fait référence à la date de fin réelle du travail J_i.

– Itération 1 :

$L_1[1] = J_1$ et $L_2[1] = J_1$

$C_1 = max(r_1, C_{\sigma[1]}) + P_{11} = max(1, 0) + 2 = 3$ sur M_1
$C_1 = max(r_1, C_{\sigma[2]}) + P_{12} = max(1, 0) + 1 = 2$ sur M_2
Le travail J_1 est affecté sur la machine qui l'exécute au plus tôt donc il sera sur la machine M_2.

Mise à jour de L_1 et L_2 :
$L_1 = \{J_2, J_3, J_4, J_5, J_6, J_7, J_8\}$
$L_2 = \{J_2, J_6, J_3, J_4, J_5, J_7, J_8\}$

– Itération 2 :

$L_1[1] = J_2$ et $L_2[1] = J_2$

$C_2 = max(r_2, C_{\sigma[1]}) + P_{21} = max(2, 0) + 4 = 6$ sur M_1
$C_2 = max(r_2, C_{\sigma[2]}) + P_{22} = max(2, 2) + 2 = 4$ sur M_2

Le travail J_2 est affecté sur la machine M_2.

Mise à jour de L_1 et L_2 :
$L_1 = \{J_3, J_4, J_5, J_6, J_7, J_8\}$
$L_2 = \{J_6, J_3, J_4, J_5, J_7, J_8\}$

– Itération 3 :

$L_1[1] = J_3$ et $L_2[1] = J_6$

$C_3 = max(r_3, C_{\sigma[1]}) + P_{31} = max(3, 0) + 8 = 11$ sur M_1
$C_3 = max(r_3, C_{\sigma[2]}) + P_{32} = max(3, 4) + 4 = 8$ sur M_2

Ordonnancement du travail J_6
$C_6 = max(r_6, C_{\sigma[1]}) + P_{61} = max(6, 11) + 2 = 13$ sur M_1
$C_6 = max(r_6, C_{\sigma[2]}) + P_{62} = max(6, 8) + 1 = 9$ sur M_2

Le travail J_3 peut être ordonnancé sur la machine M_2 parce que le travail J_6 peut être ordonnancé sur M_2 sans dépasser sa date de fin impérative.

Mise à jour de L_1 et L_2 :

$L_1 = \{J_4, J_5, J_6, J_7, J_8\}$
$L_2 = \{J_6, J_4, J_5, J_7, J_8\}$

– Itération 4 :

$L_1[1] = J_4$ et $L_2[1] = J_6$

$C_4 = max(r_4, C_{\sigma[1]}) + P_{41} = max(4, 0) + 4 = 8$ sur M_1
$C_4 = max(r_4, C_{\sigma[2]}) + P_{42} = max(4, 8) + 2 = 10$ sur M_2

Ordonnancement du travail J_6
$C_6 = max(r_6, C_{\sigma[1]}) + P_{61} = max(6, 8) + 2 = 10$ sur M_1
$C_6 = max(r_6, C_{\sigma[2]}) + P_{62} = max(6, 10) + 1 = 11$ sur M_2

Si nous affectons le travail J_4 en premier il ne sera plus possible d'affecter le travail J_6 sans dépasser sa date de fin impérative. Donc nous commençons d'abord par trouver une machine pour le travail J_6 ensuite nous ordonnançons le travail J_4.

$C_6 = max(r_6, C_{\sigma[1]}) + P_{61} = max(6, 0) + 2 = 8$ sur M_1
$C_6 = max(r_6, C_{\sigma[2]}) + P_{62} = max(6, 8) + 1 = 9$ sur M_2

Le travail J_6 est ordonnancé sur la machine M_1.

Mise à jour de L_1 et L_2 :
$L_1 = \{J_4, J_5, J_7, J_8\}$
$L_2 = \{J_4, J_5, J_7, J_8\}$

– Itération 5 :

$L_1[1] = J_4$ et $L_2[1] = J_4$

$C_4 = max(r_4, C_{\sigma[1]}) + P_{41} = max(5, 8) + 4 = 12$ sur M_1
$C_4 = max(r_4, C_{\sigma[2]}) + P_{42} = max(5, 8) + 2 = 10$ sur M_2

Le travail J_4 est ordonnancé sur la machine M_2.

Mise à jour de L_1 et L_2 :
$L_1 = \{J_5, J_7, J_8\}$

$L_2 = \{J_5, J_7, J_8\}$

– Itération 6 :

$L_1[1] = J_5$ et $L_2[1] = J_5$

$C_5 = max(r_5, C_{\sigma[1]}) + P_{51} = max(5, 8) + 6 = 14$ sur M_1
$C_5 = max(r_5, C_{\sigma[2]}) + P_{52} = max(5, 10) + 3 = 13$ sur M_2

Le travail J_5 est ordonnancé sur la machine M_2.

Mise à jour de L_1 et L_2 :
$L_1 = \{J_7, J_8\}$
$L_2 = \{J_7, J_8\}$

– Itération 7 :

$L_1[1] = J_7$ et $L_2[1] = J_7$

$C_7 = max(r_7, C_{\sigma[1]}) + P_{71} = max(7, 8) + 4 = 12$ sur M_1
$C_7 = max(r_7, C_{\sigma[2]}) + P_{72} = max(7, 13) + 2 = 15$ sur M_2

Le travail J_7 est ordonnancé sur la machine M_1.

Mise à jour de L_1 et L_2 :
$L_1 = \{J_8\}$
$L_2 = \{J_8\}$

– Itération 8 :

$L_1[1] = J_8$ et $L_2[1] = J_8$

$C_8 = max(r_8, C_{\sigma[1]}) + P_{81} = max(8, 12) + 2 = 14$ sur M_1
$C_8 = max(r_8, C_{\sigma[2]}) + P_{82} = max(8, 13) + 1 = 14$ sur M_2

Le travail J_8 peut être ordonnancé sur la machine M_1 ou sur la machine M_2, on l'ordonnace sur la machine M_1 .

Mise à jour de L_1 et L_2 :

$L_1 = \emptyset$

$L_2 = \emptyset$

La liste Tabou La liste Tabou contient les solutions déjà explorées par l'approche tabou, elle est générée selon la stratégie FIFO (First In First Out), on élimine la plus vieille solution tabou et on insère la nouvelle solution. En général, la liste tabou doit être maintenue à une longueur minimale permettant d'éviter un cyclage.

La taille de la liste Tabou est prise à dix (10), cette valeur est suffisante et fréquemment utilisée dans la littérature car elle évite le cyclage et le blocage du processus de recherche.

La mémorisation des solutions s'avère trop coûteuse en temps de calcul et espace mémoire d'où l'intérêt de mémoriser des caractéristiques relatives aux solutions. Dans notre travail, nous mémorisons les numéros des travaux interchangés et leurs machines sur lesquelles ils sont ordonnancés.

Définition d'un voisinage La définition d'un voisin se fait en utilisant deux heuristiques. La première, Swap-Move, interchange les travaux entre les machines sachant qu'ils sont ordonnancés sur deux machines différentes. Les nouveaux séquencements se font en respectant les dates de disponibilités.

La deuxième heuristique est l'heuristique Insert-Move, Cette heuristique permet d'ajouter un travail J_i à une solution partielle σ_j sur une machine M_j donnée. Elle commence par examiner les travaux de cette solution partielle par date de début effective croissante, i.e. t_k croissant, on rappelle que dans cette solution tous les travaux sont calés à gauche. Elle identifie ensuite les différents blocs et les temps morts qui existent entre ces blocs. Un bloc est un ensemble de travaux dont la fin de chaque travail coïncide avec le début du suivant, i.e. si J_k précède immédiatement J_l alors $C_k = t_l$ (cf. figure 4.8). Ensuite la méthode détermine les décalages maximum qu'elle peut effectuer sur les blocs pour pouvoir placer le travail J_i entre deux blocs consécutifs.
Le décalage maximum qu'on détermine tient compte des autres blocs présents sur la machine, et comme le décalage s'effectue toujours vers la droite, les blocs qui précèdent le bloc dont on calcule le décalage maximum n'interviennent pas dans ce calcul. Étant donné que l'ordonnancement partiel σ_j est un ordonnancement "calé à gauche", le premier travail J_k de chaque bloc commence à la date r_k. Dans cette partie nous présentons une formulation mathématique du décalage maximum, et nous donnons le fonctionnement de la méthode d'insertion.
On note b le nombre de blocs de σ_j. Soient B_p le bloc numéro p, π_p la durée du temps mort suivant le bloc B_p, et δ_p^{max} le décalage maximum que le bloc pourra subir sans tenir compte des autres blocs. On a : $\delta_p^{max} = min_{k \in B_p} (d_k - C_k)$. Le décalage maximum du bloc B_p en tenant compte des blocs qui suivent B_p est noté par δmax.

1 /*i^*, j^*, k^*, l^* = stockent le meilleur mouvement */

2 /Interchanger le travail $J_{i^*}(k^*)$ ordonnancé sur la machine M_{i^*}*/

2 /avec le travail $J_{j^*}(l^*)$ ordonnancé sur la machine M_{j^*}*/

3 /*$Best - position(J_i(k), J_j(l), M_i, M_j)$: calcul le meilleur swap*/

3 /*entre le travail $J_i(k)$ sur M_i et le travail $J_j(l)$ sur M_j */

4 $Etape1 : i^* = j^* = k^* = l^* = 0$; $Best - value = \infty$

5 $Etape2$: i=0 ;

6 $Etape3$: Si(i=m) alors, arrêter.

7 | Sinon, poser j=i+1 ;

8 $Etape4$: Si($j > m$) alors, aller à l'étape 10.

9 | Sinon, poser k=1 ;

10 $Etape5$: Si le travail $J_i(k)$ n'est pas en position k sur la machine M_i, aller à l'étape 9 ;

11 | Sinon, poser l=1 ;

12 $Etape6$: Si le travail $J_j(l)$ n'est pas en position l sur la machine M_j, aller à l'étape 8 ;

13 | Sinon, poser $value = Best - position(J_i(k), J_j(l), M_i, M_j)$;

14 | | If ($value < Best - value$) alors ;

15 | | | $Best - value = value$; $i^* = i$; $j^* = j$; $k^* = k$; $l^* = l$;

16 $Etape7$: poser $l = l + 1$, aller à l'étape 6 ;

17 $Etape7$: poser $k = k + 1$, aller à l'étape 5 ;

18 $Etape7$: poser $j = j + 1$, aller à l'étape 4 ;

19 $Etape7$: poser $i = i + 1$, aller à l'étape 3 ;

FIGURE 4.7 – L'heuristique swap-move

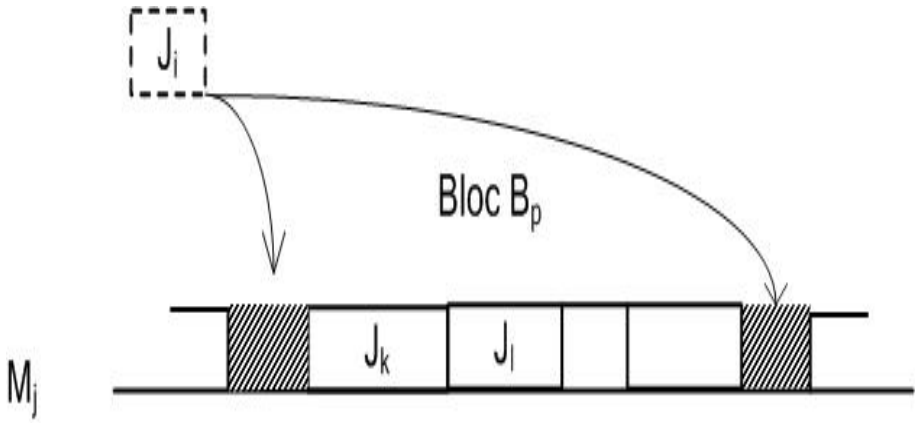

FIGURE 4.8 – Un bloc de l'ordonnancement partiel σ_j

.

Proposition 4.2.2. *Le décalage maximum δmax d'un bloc B_p est défini par :*

$\delta max = \sum_{u=p}^{b-1} \pi_u + min_{1 \leq u \leq b-1} \left(\delta_u^{max} - \sum_{v=u}^{b-1} \pi_v ; \delta_b^{max} \right)$

Preuve

On supposera, sans perte de généralité, que $p = 1$.

- Pour b=1, il n'y qu'un seul bloc, on a donc la formule suivante : $\delta max = \delta_1^{max}$.
- Pour b=2, on distingue deux cas :

  1. Si $\delta_1^{max} \leq \pi_1$, alors $\delta max = \delta_1^{max}$.
  2. Si $\delta_1^{max} > \pi_1$, alors $\delta max = \pi_1 + min\left(\delta_1^{max} - \pi_1; \delta_2^{max}\right)$.

  Notons que, si $\delta_1^{max} \leq \pi_1$, la valeur $\delta_1^{max} - \pi_1$ est négative. De plus, $\forall u \ \delta_u^{max} \geq 0$ et on a $min\left(\delta_1^{max} - \pi_1; \delta_2^{max}\right) = \delta_1^{max} - \pi_1$. Donc, si $\delta_1^{max} \leq \pi_1$, $\delta max = \pi_1 + \delta_1^{max} - \pi_1 = \delta_1^{max}$. On peut déduire que dans les deux cas de figure :

$$\delta max = \pi_1 + min\left(\delta_1^{max} - \pi_1; \delta_2^{max}\right)$$

- Pour b=3, on distingue trois cas :

  1. Si $\delta_1^{max} \leq \pi_1$, alors $\delta max = \delta_1^{max}$.
  2. Si $\delta_1^{max} > \pi_1$ et $min\left(\delta_1^{max} - \pi_1; \delta_2^{max}\right) \leq \pi_2$, on se retrouve dans le cas ou b=2, i.e, $\delta max = \pi_1 + min\left(\delta_1^{max} - \pi_1; \delta_2^{max}\right)$.
  3. Si $\delta_1^{max} > \pi_1$ et $min\left(\delta_1^{max} - \pi_1; \delta_2^{max}\right) > \pi_2$, le troisième bloc risque de diminuer le décalage du premier bloc. On a donc de la même façon que dans le cas b=2, $\delta max = \pi_1 + \pi_2 + min\left(min\left(\delta_1^{max} - \pi_1; \delta_2^{max}\right) - \pi_2; \delta_3^{max}\right)$

  On peut déduire là encore l'expression mathématique suivante :

$$\delta max = \pi_1 + \pi_2 + min\left(\delta_1^{max} - \pi_1 - \pi_2; \delta_2^{max} - \pi_2; \delta_3^{max}\right)$$

- Par analogie on en déduit l'expression mathématique suivante :

$$\delta max = \sum_{u=1}^{b-1} \pi_u + min_{1 \leq u \leq b-1}\left(\delta_u^{max} - \sum_{v=u}^{b-1} \pi_v, \delta_b^{max}\right) \ \square$$

Une fois que les blocs ont été identifiés, et que toutes les données les concernant ont été récoltées, la méthode peut réellement commencer. En premier lieu on essaye de placer le travail candidat avant le premier bloc. Si le travail ne peut pas être inséré, on décale le premier bloc vers la droite et on tente toujours de mettre le travail en question avant le premier bloc. Si on n'arrive toujours pas à le placer, alors on essaye de l'insérer entre le premier et le deuxième bloc. Si le travail n'est toujours pas inséré on déplace le deuxième bloc toujours vers la droite, et on fait une deuxième tentative, ainsi de suite pour tous les blocs $p = 1, ..., b$. Si le travail concerné n'a pas été placé entre deux blocs, on essaye de le placer après le dernier bloc. Cette méthode d'insertion peut être implémentée en $O(n)$. L'algorithme de la méthode est donnée dans la figure 4.9.

Exemple L'exemple suivant illustre le fonctionnement de la méthode d'insertion d'un travail à séquence fixée.

- Les données.

  On considère un problème pour lequel le nombre de travaux $n = 6$ et le nombre de machine $m = 2$.

  Les caractéristiques des travaux sont :

i	1	2	3	4	5	6
r_i	0	3	3	0	9	6
p_i	2	4	8	3	6	2
d_i	4	8	10	15	18	22

  Les vitesses des machines sont données par :

j	1	2
V_j	1	2

  Soit J_4 le travail que nous voulons ordonnancer dans la séquence $\sigma_1 = \{1, 2, 5, 6\}$ sur la machine M_1.

- Initialisation.
  - Identification des blocs

    Nombre de blocs : $b = 3$

    $B_1 = \{J_1\}$, $B_2 = \{J_2\}$, $B_3 = \{J_5, J_6\}$

  - Identification des temps morts.

    $\pi_1 = 1$, $\pi_2 = 2$

- On essaie de placer le travail J_4 avant le premier bloc.

  Le travail J_4 ne peut pas être placé avant le bloc B_1 parce que :

  $Min(t_{\sigma_j[1]}, d_4) - r_4 = min(0, 15) - 0 = 0 < \frac{p_4}{V_1} = 4$.

  Donc nous essayons de décaler le premier bloc.

- Calcul du décalage des trois blocs et du décalage maximum

  - $\delta_1^{max} = min(d_1 - C_1) = 4 - 2 = 2$

1 /* $t_{\sigma_j[i]}$: date début réelle du premier travail du bloc i*/
2 /* δ_i^{max} : décalage du i^{eme} bloc, δ_i^* : décalage nécéssaire du i^{eme} bloc */
3 /* π_i :la longuer du temps mort suivant le i^{eme} bloc, b : le nombre de bloc */
4 Si $Min\left(t_{\sigma_j[1]}, d_k\right) - r_k \geq \frac{p_k}{V_j}$
5 Alors /*le travail peut être placé avant le premier bloc*/
6 $t_k = r_k$
7 /*La méthode a trouvé une solution, on retourne 1*/
8 Sinon /*le travail J_k ne peut pas être placé pas avant le premier bloc*/
9 /*On essaie de décaler le premier bloc pour pouvoir placer le travail*/
10 Calcul du δ_1^{max}
11 /*On essaie de placer le travail avant le bloc décalé au maximum*/
12 Si $Min\left(\pi_1 + \delta_1^{max}, d_k\right) - r_k \geq \frac{p_k}{V_j}$
13 Alors /*le travail J_k peut être placé avant le premier bloc décalé
14 $t_k = r_k$; $\delta_1^* = t_k + \frac{p_k}{V_j} - t_{\sigma_j[1]}$; $t_{\sigma_j[1]} = t_k$
15 /*La méthode a trouvé une solution, on retourne 1*/
16 FinSi
17 FinSi
18 /*on essaie de placer le travail J_k entre le bloc i et le bloc $i+1$*/
19 Pour i de 1 à $b-1$ Faire
20 j = Dernier Travail Du Bloc i
21 Si$Min\left(t_{\sigma_j[i+1]}, d_k\right) - Max\left(t_j + \frac{p_j}{V_j}, r_k\right) \geq \frac{p_k}{V_j}$
22 Alors /*le travail peut être placé entre le bloc $i-1$ et le bloc i*/
23 $t_k = Max\left(t_j + \frac{p_j}{V_j}, r_k\right)$
24 La méthode a trouvé une solution, on retourne 1
25 Sinon /*le travail ne peut pas être placé entre le bloc $i-1$ et le bloc i*/
26 /* On essaie de déplacer le bloc i pour pouvoir placer le travail J_k*/
27 Calcul du décalage δ^{max} du i^{eme} bloc
28 /*On essaie de placer le travail avant de décalé au maximum */
29 Si$Min\left(t_{\sigma_j[i]} + \delta_{i+1}^{max}, d_k\right) - Max\left(t_j + \frac{p_j}{V_j}, r_k\right) \geq \frac{p_k}{V_j}$
30 Alors /*le travail peut être placé avant le bloc $i+1$ décalé*/
31 $t_k = Max\left(t_j + \frac{p_j}{V_j}, r_k\right)$
32 décalage nécessaire= $t_k + \frac{p_k}{V_j} - t_{\sigma_j[i]}$
33 Mise à jour de la date de début du bloc concerné et de celles des blocs suivants
34 La méthode a trouvé une solution, on retourne 1
35 FinSi
36 FinSi
37 FinPour
28 /* On essaie de placer le travail après le dernier bloc */
29 j =DernierTravailDuBloc b
30 Si $d_k - Max\left(t_j + \frac{p_j}{V_j}, r_k\right) \geq \frac{p_k}{V_j}$
31 Alors le travail peut être placé après le dernier travail placé
32 $t_k = Max\left(t_j + \frac{p_j}{V_j}, r_k\right)$
33 La méthode a trouvé une solution, on retourne 1
34 FinSi
35 La méthode n'a pas pu trouver de solution, on retourne 0

FIGURE 4.9 – Procedure Insert-Move

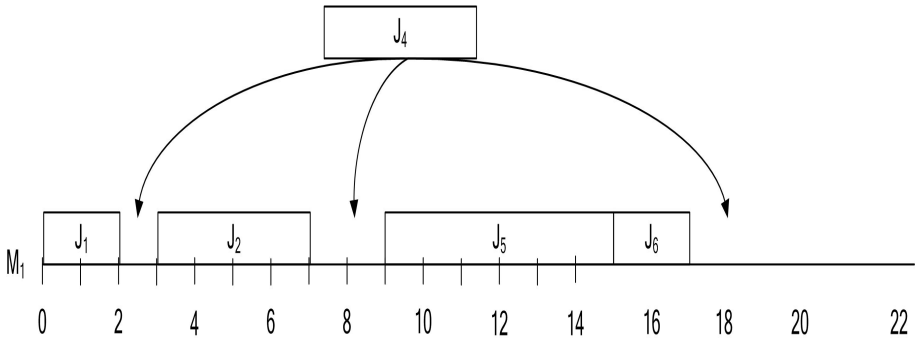

FIGURE 4.10 – Insertion du travail J_4 dans la séquence σ_1

.

- $\delta_2^{max} = min(d_2 - C_2) = 8 - 7 = 1$

- $\delta_3^{max} = min(d_5 - C_5, d_6 - C_6) = min((18 - 15), (22 - 18)) = min(3, 4) = 3$

- $\delta^{max} = \pi_1 + \pi_2 + min(\delta_1^{max} - \pi_1 - \pi_2; \delta_2^{max} - \pi_2; \delta_3^{max})$
  $= 1 + 2 + min(2 - 1 - 2; 1 - 2; 2) = 3 + min(-1; -1; 3) = 3 - 1 = 2.$
  Donc on arrive toujours pas à placer le travail J_4 avant le bloc B_1.
- On essaie de placer le travail J_4 entre le bloc B_1 et le bloc B_2.

  $Min(t_2, \tilde{d_4}) - Max(t_1 + \frac{p_1}{V_1}, r_4) = Min\,(3, 15) - Max\,(0 + 2, 0) = 3 - 2 = 1$
  $\frac{p_4}{V_1} = 3.$
  Nous avons $Min(t_2, d_4) - Max(t_1 + \frac{p_1}{V_1}, r_4) < \frac{p_4}{V_1}$ donc le travail J_4 ne peut pas être ordonnancé entre B_1 et B_2.

- On décale le bloc B_2.

  $Min(t_2 + \delta_2^{max}, d_4) - Max(t_1 + \frac{p_1}{V_1}, r_4) = Min\,(3 + 1, 15) - Max\,(0 + 2, 0) = 4 - 2 = 2.$
  Nous avons également $Min(t_1 + \delta^{max}, d_4) - Max(t_1 + \frac{p_1}{V_1}, r_4) < \frac{p_4}{V_1}$ donc le travail J_4 ne peut toujours pas être ordonnancé entre B_1 et B_2.

- On essaie de placer le travail J_4 entre le bloc B_2 et le bloc B_3.

  $Min(t_5, d_4) - Max(t_2 + \frac{p_2}{V_1}, r_4) = Min\,(9, 15) - Max\,(3 + 4, 0) = 9 - 7 = 2$
  $\frac{p_4}{V_1} = 3.$
  Nous avons $Min(t_5, d_4) - Max(t_2 + \frac{p_2}{V_1}, r_4) < \frac{p_4}{V_1}$ donc le travail J_4 ne peut pas être ordonnancé entre B_2 et B_3.

– On décale le bloc B_3 $Min(t_5 + \delta^{max}, d_4) - Max(t_2 + \frac{p_2}{V_1}, r_4) = Min\,(9+3, 15) - Max\,(3+4, 0) = 11 - 7 = 4$.

Nous avons $Min(t_5, d_4) - Max(t_2 + \frac{p_2}{V_1}, r_4) < \frac{p_4}{V_1}$ donc le travail J_4 ne peut pas être ordonnancé entre B_2 et B_3.

$t_4 = max(t_2 + \frac{p_2}{V_1}, r_4) = max(3+4, 0) = 7$.
décalage nécessaire$= t_4 + \frac{p_4}{V_1} - t_2 = 7 + 3 - 10 = 0$.

Donc le travail J_4 va être placé après le travail J_6.

4.4.2.2 Evaluation du voisin Dans le cadre d'une approche multicritère, on peut appliquer différentes techniques pour la détermination d'optima de Pareto. Nous nous sommes intéressé à trois méthodes pour la détermination de solutions non-dominées : une marche aléatoire (notée par la suite TS_{rw}), une combinaison linéaire des critères (notée par la suite TS_{cl}) et approche epsilon-contrainte (TS_ϵ).

A. Approche par Marche Aléatoire TS_{rw} Le principe de la marche aléatoire (Random Walk (RW)) est bien connu dans la littérature et couramment utilisé dans des algorithmes réputés WSAT [Selman et al., 1994].
La marche aléatoire consiste à réaliser de temps en temps un mouvement qui n'est plus guidé par la fonction d'évaluation et constitue donc un schéma de diversification. Intégrer la marche aléatoire dans un algorithme Tabou standard permet d'obtenir une recherche plus diversifiée.

L'algorithme procède comme suit : Soit $q \in [0, 1]$, cette valeur reste fixe pendant toute la recherche.À chaque itération, $rw \in [0, 1]$ est choisie aléatoirement, si $rw > q$ l'algorithme TS_rw exécutera un mouvement classique càd on choisit la meilleure solution trouvée. Dans le cas contraire, l'algorithme choisit un mouvement aléatoirement sans prendre en considération la liste tabou ou le mouvement interdit et continuer son exécution.

Le mouvement aléatoire se fait comme suit : On choisit deux machines aléatoirement, on détermine les travaux ordonnancés sur chaque machine et on sélectionne aléatoirement un travail sur chacune d'elles et en interchangeant ces deux travaux, un nouveau ordonnancement (solution) est crée.

La détermination du front pareto Au fur et à mesure que la recherche avance, l'approche sauvegarde toutes les meilleures solutions de chaque itérations. À la fin de la recherche, on procède à l'élimination des solutions dominées et on détermine une approximation du front Pareto.

B. Approche par Combinaison Linéaire TS_{cl} On évalue un voisin par une combinaison linéaire des deux critères, on considère le makespan (C_{max}) et le retard algèbrique (L_{max}).

Algorithme 2. Algorithme tabou avec marche aléatoire (TS_{rw})

entrées : s_{init} une configuration réalisable, L le nombre d'itérations, q le seuil de perturbation

sorties : s la nouvelle configuration réalisable

```
s := s_init
for i = 0 to L do
    générer aléatoirement un nombre rw ∈ [0, 1]
    if random value rw < q then
        choisir un mouvement MV au hasard
    else
        choisir le meilleur mouvement MV autorisé
    end if
    Mettre à jour la liste tabou en fonction de MV
    Effectuer le mouvement MV dans s
    mettre à jour l'ensemble des solutions non dominées en fonction de s
End for
```

FIGURE 4.11 – Algorithme tabou avec marche aléatoire (TS_{rw})

Un des problèmes posés par une telle approche est l'ordre des grandeurs des critères. De plus, il peut être impossible d'estimer à priori l'ordre de grandeur de chaque critère. Pour surmenter cette difficulté, les coefficients de la combinaison linéaire sont normalisés, cette normalisation est inspérée par [Deb et al., 2000] pour le calcul de la `distance de crowding`.

L'évaluation d'un voisin (S) est donnée par la formule suivante :

$$Z(S) = \frac{\alpha C_{max}}{max(1, max(C_{max}) - min(C_{max}))} + \frac{\beta L_{max}}{max(1, max(L_{max}) - min(L_{max}))}$$

Avec

- α, β : les coefficients choisit par le dicideur.
- $C_{max}(S)$ et $L_{max}(S)$ le makespan et le retard algèbrique du voisin S.

On note S^i_{best} le voisin choisi à l'itération i et k le numéro de l'itération courante de l'approche tabou. On calcule les valeurs maximales et minimales des critères par la formule suivante :

$$Y = S^i_{best}, \quad \forall i, i < k$$

$$max(C_{max}) = max_{y \in Y} C_{max}(y)$$

$$min(C_{max}) = min_{y \in Y} C_{max}(y)$$

$$max(L_{max}) = max_{y \in Y} L_{max}(y)$$

$$min(L_{max}) = min_{y \in Y} L_{max}(y)$$

La diversification Le problème des méthodes de la recherche locales et particulièrement la recherche tabou est qu'elles tombent souvent dans des optimums locaux, cela empêche l'exploration d'autres zones de recherche, là où peut être une meilleure solution existe. Pour remédier à ce problème, un opérateur de diversification a été utilisé, inspéré par le principe d'oscillation stratgique. Ce dernier consite à autoriser l'exploration de solution non faisable pendant une courte période de temps.

La diversité que nous avons implémentée fonctionne selon le principe suivant : Après un certain nombre d'itération sans amélioration, la recherche recommence avec la meilleure solution connue S^+ et on tire aléatoirement de nouveaux coefficients $\grave{\alpha}$ et $\grave{\beta}$ pour la combinaison linéaire. En utilisant cette technique, certaine solutions qui n'auraient jamais étaient explorées avec les coefficients initiaux peuvent être explorées. La recherche continue avec ces nouveaux coefficients jusqu'à ce qu'une meilleure solution que S^+ (avec les coefficients $\grave{\alpha}$ et $\grave{\beta}$) soit trouvée ou jusqu'à ce que le critère d'arrêt soit atteint. Dans le premier cas, l'approche continue avec la nouvelle solution et les coefficients initiaux, sinon c'est la fin de la recherche.

Détermination d'un front de Pareto À la fin de la recherche, l'approche fournit une approximation du front pareto.

L'exemple suivant illuste le fonctionnement d'une itération de l'approche tabou par combinaison linéaire des deux critères, le makespan C_{max} et le retard algèbrique L_{max}. Nous avons vu le déroulement de l'heuristique H qui fournit la solution initiale et à partir de cette solution qui devient la solution courante, l'approche construit les voisins et choisit le meilleur d'entre eux en utilisant la combinaison linéaire de C_{max} et L_{max}.

Exemple

- Les données :

  On considère un problème pour lequel le nombre de travaux $n = 8$ et le nombre de machine $m = 2$.

  Les caractéristiques des travaux sont :

i	1	2	3	4	5	6	7	8
r_i	1	2	3	4	5	6	7	8
p_i	2	4	8	4	6	2	4	2
d_i	4	7	10	10	15	9	15	15

TABLE 4.1 – Les caractéristiques des travaux de l'exemple

Les vitesses des machines sont données par :

j	1	2
V_j	1	2

TABLE 4.2 – Le tableau des vitesses des machines

Les temps opératoires des travaux sur les deux machines dépendent des vitesses.

i	1	2	3	4	5	6	7	8
p_{i1}	2	4	8	4	6	2	4	2
p_{i2}	1	2	4	2	3	1	2	1

TABLE 4.3 – Les temps opératoires des travaux

Dans cet exemple nous utilisons les notations suivantes : $C_{\sigma[1]}$ et $C_{\sigma[2]}$ désignent respectivement les dates de fin du dernier travail sur la machine M_1 et M_2. $c_i[i]$ fait référence à la date de fin réelle du travail J_i.

L'heuristique H fournit la solution suivante :

Les travaux (J_6, J_7, J_8) sont ordonnancés dans cet ordre sur la machine M_1 et $(J_1, J_2, J_3, J_4, J_5)$ sont ordonnancés dans cet ordre sur la machine M_2, avec $C_{max} = 15$ et $L_{max} = 0$

La machine qui fournit max c_i est M_1 et M_2 fournit min c_i. Le nombre de voisin est de trois, il est égale au nombre de travaux ordonnacé sur la machine M_1 et chaque voisin est obtenu en interchangeant le dernier travail ordonnancé sur M_2 avec chacun des travaux ordonnac sur M_1.

– Le voisin V_1 : interchanger (J_5, J_6).

  Trier les travaux de M_1 par ordre croissant des r_i.

$\{J_5, J_7, J_8\}$ sur M_1

$c_i[5] = Max(r_i[5], C_{\sigma[1]}) + P_{51} = Max(5, 0) + 2 = 11$
$l_i[5] = c_i[5] - d_i[5] = 11 - 15 = -4$

Mise à jour de $C_{\sigma[1]}$
$C_{\sigma[1]} = 11$

$c_i[7] = Max(r_i[7], C_{\sigma[1]}) + P_{71} = Max(7, 11) + 2 = 13$
$l_i[7] = c_i[7] - d_i[7] = 13 - 15 = -2$

Mise à jour de $C_{\sigma[1]}$
$C_{\sigma[1]} = 13$

$c_i[8] = Max(r_i[8], C_{\sigma[1]}) + P_{81} = Max(8, 13) + 2 = 15$
$l_i[8] = c_i[8] - d_i[8] = 15 - 15 = 0$

Mise à jour de $C_{\sigma[1]}$
$C_{\sigma[1]} = 15$

Trier les travaux de M_2 par ordre croissant des r_i.

$\{J_1, J_2, J_3, J_4, J_6\}$ sur M_2

$c_i[1] = Max(r_i[1], C_{\sigma[2]}) + P_{12} = Max(1, 0) + 1 = 2$
$l_i[1] = c_i[1] - d_i[1] = 2 - 4 = -2$

Mise à jour de $C_{\sigma[2]}$
$C_{\sigma[2]} = 2$

$c_i[2] = Max(r_i[2], C_{\sigma[2]}) + P_{22} = Max(2, 2) + 2 = 4$
$l_i[2] = c_i[2] - d_i[2] = 4 - 7 = -3$

Mise à jour de $C_{\sigma[2]}$
$C_{\sigma[2]} = 4$

$c_i[3] = Max(r_i[3], C_{\sigma[2]}) + P_{32} = Max(3, 4) + 4 = 8$
$l_i[3] = c_i[3] - d_i[3] = 8 - 10 = -2$

Mise à jour de $C_{\sigma[2]}$
$C_{\sigma[2]} = 8$

$c_i[4] = Max(r_i[4], C_{\sigma[2]}) + P_{42} = Max(4, 8) + 2 = 10$
$l_i[4] = c_i[4] - d_i[4] = 10 - 10 = 0$

Mise à jour de $C_{\sigma[2]}$
$C_{\sigma[2]} = 10$

$c_i[6] = Max(r_i[6], C_{\sigma[2]}) + P_{62} = Max(6, 10) + 1 = 11$
$l_i[6] = c_i[6] - d_i[6] = 11 - 9 = 2$

Mise à jour de $C_{\sigma[2]}$
$C_{\sigma[2]} = 11$

Les valeurs du voisin V_1 :

$$\begin{cases} C_{max} = 15 \\ L_{max} = 2 \end{cases}$$

– Le voisin V_2 : interchanger (J_5, J_7).
Trier les travaux de M_1 par ordre croissant des r_i.

$\{J_5, J_6, J_8\}$ sur M_1

Pour le travail J_5 , son c_i et son d_i sont de même valeur que dans le voisin V_1. et
$C_{\sigma[1]} = 11$

$c_i[6] = Max(r_i[6], C_{\sigma[1]}) + P_{61} = Max(6, 11) + 2 = 13$
$l_i[6] = c_i[6] - d_i[6] = 13 - 15 = -2$

$c_i[8] = Max(r_i[8], C_{\sigma[1]}) + P_{81} = Max(8, 13) + 2 = 15$

$l_i[8] = c_i[8] - d_i[8] = 15 - 15 = 0$

Mise à jour de $C_{\sigma[1]}$
$C_{\sigma[1]} = 15$

Trier les travaux de M_2 par ordre croissant des r_i.

$\{J_1, J_2, J_3, J_4, J_7\}$ sur M_2

Pour les travaux J_1 jusqu'à J_4, leurs c_i et leurs d_i sont de même valeur que dans le voisin V_1. et $C_{\sigma[2]} = 10$

$c_i[7] = Max(r_i[7], C_{\sigma[2]}) + P_{72} = Max(7, 10) + 2 = 12$
$l_i[7] = c_i[7] - d_i[7] = 12 - 15 = -3$

Mise à jour de $C_{\sigma[2]}$
$C_{\sigma[2]} = 12$

Les valeurs du voisin V_2 :

$$\begin{cases} C_{max} = 15 \\ L_{max} = 0 \end{cases}$$

– Le voisin V_3 : interchanger (J_5, J_8).
Trier les travaux de M_1 par ordre croissant des r_i.

$\{J_5, J_6, J_7\}$ sur M_1

Pour les travaux J_5 et J_6, leurs c_i et leurs d_i sont de même valeur que dans le voisin V_1. et $C_{\sigma[1]} = 13$

$c_i[7] = Max(r_i[7], C_{\sigma[1]}) + P_{71} = Max(7, 13) + 4 = 17$
$l_i[7] = c_i[7] - d_i[7] = 17 - 15 = 2$

Mise à jour de $C_{\sigma[1]}$
$C_{\sigma[1]} = 17$

Trier les travaux de M_2 par ordre croissant des r_i.

$\{J_1, J_2, J_3, J_4, J_8\}$ sur M_2

Pour les travaux J_1 jusqu'à J_4, leurs c_i et leurs d_i sont de même valeur que dans le voisin V_1. et $C_{\sigma[2]} = 10$

$c_i[8] = Max(r_i[8], C_{\sigma[2]}) + P_{82} = Max(8, 10) + 1 = 11$
$l_i[8] = c_i[8] - d_i[8] = 11 - 15 = -4$

Mise à jour de $C_{\sigma[2]}$
$C_{\sigma[2]} = 11$

Les valeurs du voisin V_3 :

$$\begin{cases} C_{max} = 17 \\ L_{max} = 2 \end{cases}$$

– Evaluation d'un voisin :

L'évaluation d'un voisin (S) est donnée par la formule suivante :

$$Z(S) = \frac{\alpha C_{max}}{max(1, max(C_{max}) - min(C_{max}))} + \frac{\beta L_{max}}{max(1, max(L_{max}) - min(L_{max}))}$$

Comme on est dans la première itération, la fomule devient alors :

$$Z(S) = \alpha C_{max} + \beta L_{max}$$

Soit $\alpha = 0.4$ et $\beta = 1 - \alpha = 0.6$

Pour le voisin V_1 :

$$\begin{cases} C_{max} = 15 \\ L_{max} = 2 \end{cases}$$

$Z_1(S) = 0.4 * 15 + 0.6 * 2 = 7.2$

Pour le voisin V_2 :

$$\begin{cases} C_{max} = 15 \\ L_{max} = 0 \end{cases}$$

$Z_2(S) = 0.4 * 15 + 0.6 * 0 = 6$

Pour le voisin V_3 :

$$\begin{cases} C_{max} = 17 \\ L_{max} = 2 \end{cases}$$

$Z_3(S) = 0.4 * 17 + 0.6 * 2 = 8$

Le meilleur voisin est le voisin V_2 qui correspond au minimum des Z(S). Donc

$$\begin{cases} C_{max} = 15 \\ L_{max} = 0 \end{cases}$$

devient la solution courante pour la prochaine itération et ce même processus est répété jusqu'à ce que le critère d'arrêt soit atteint.

C. Approche par Epsilon-contrainte TS_ϵ On suppose qu'on cherche à optimiser le (C_{max} sous la contrainte $L_{max} < \epsilon$). Pour l'évaluation on distingue trois cas :

1. Il existe au moins un voisin qui respecte la borne ϵ. On considère alors que le meilleur voisin est celui qui minimise C_{max} parmi ceux qui vérifient $L_{max} < \epsilon$.
2. Aucun voisin ne respecte la borne ϵ et il en existe au moins un qui a C_{max} meilleur que celui de la solution courante. Alors on considère que le meilleur voisin est celui qui minimise le L_{max} tout en étant au moins aussi bon sur le C_{max} que la solution courante.
3. Aucun voisin ne respecte la borne ϵ et aucun n'améliore le C_{max} de la solution courante. Alors, on considère que le meilleur voisin est celui qui minimise le L_{max}.

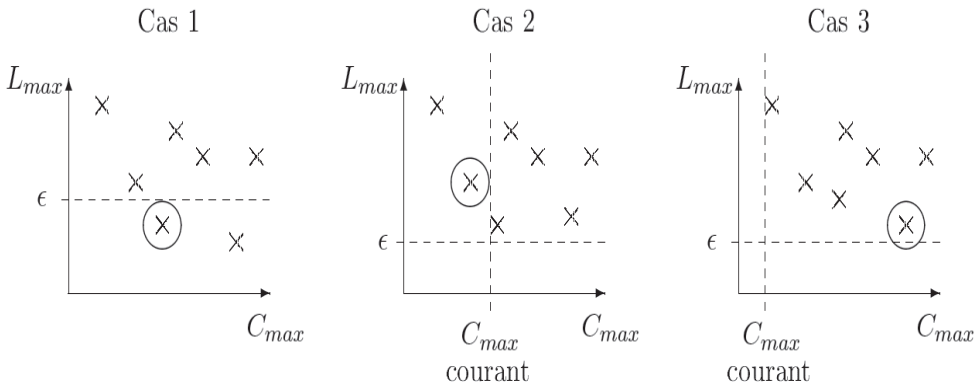

FIGURE 4.12 – Choix du meilleur voisin dans le cas epsilon-contrainte

.

4.2.3 Approches hybrides pour le problème $Q/r_i, d_i/C_{max}, L_{max}$

Dans la littérature, plusieurs problèmes d'optimisation multi-objectif en général et des problèmes d'ordonnancement multicritère en particulier, ont été résolus à l'aide des méthodes approchées. Chaque approche a montré son efficacité pour un problème ou pour un autre mais sachant que les résultats obtenus pour la plupart des problèmes ne sont pas optimals, sur ceux, des méthodes hybrides combinant

```
        /*Pré-conditions : TS_ε(val, s) lance l'approche
        TS_ε avec comme objectif L_max < val et
        en partant de la solution s. si s = ∅, alors
        la solution initiale est générée par l'heuristique H
1       ε = ∞
2       S ← TS_ε(ε, ∅)
3       TantQue (S ≠ ∅) do
4           Mettre à jour l'ensemble des solutions non-dominées avec S
5           ε = L_max(S)
6           S ← TS_ε(ε, S)
7           Si (S = ∅) then
8               S ← TS_ε(ε, ∅)
9           FinSi
10      FinTantQue
```

FIGURE 4.13 – Détermination d'un front de Pareto avec l'approche epsilon-contrainte

deux ou plusieurs méthodes ont vu le jour. Elles essayent de tirer profit des avantages de chacune des méthodes utilisées ou bien de combler certaines de ces lacunes.

Dans notre travail, on s'intéresse à l'hybridation séquentielle, respectivement, de deux algorithmes génétiques (NSGA-II et SPEA) et la recherche tabou par epsilon-contrainte.

Dans les algorithmes génétiques, avoir une bonne population initiale conditionne la rapidité de la convergence vers l'optimum. Alors, nous avons choisit d'utiliser un principe pour hybrider un algorithme génétique et l'approche par epsilon-contrainte consistant à remplacer la génération aléatoire de la population initiale par la recherche tabou. Cette dernière fournit un ensemble de solutions qui formera la population initiale pour l'algorithme génétique et sur lequel les opérateurs génétiques (sélection, croisement et mutation) sont appliqués.

0 /* TS_ϵ : Recherche Tabou */

1 /* P_0, $Q_0 = \emptyset$: les deux populations initiales */

2 /* N : la taille de la population */

3 /* tmax : Le nombre maximum de générations*/

4 Lancer TS_ϵ : la population initiale P_0 contient les individus générés par TS_ϵ

5 t=0 ;

6 TantQue $(t < tmax)$

7 $R_t = P_t \cup Q_t$;

8 Calcul des différents fronts F^j de la population R_t

9 avec un algorithme de "ranking",

10 qui permet de calculer le rang de Pareto de chaque individu ;

11 $P_{t+1} = P_t$; $j = 1$;

12 TantQue $(|P_{t+1}| + |F^j| < N)$

13 $P_{t+1} = P_{t+1} \cup F^j$;

14 $j = j + 1$

15 FinTantQue

16 Appliquer la sélection basée sur la distance de crowding sur F^j

17 pour compléter P_{t+1} ;

18 $i = 1$; $Q_{t+1} = \emptyset$;

19 TantQue $(i \leq N/2)$

20 Appliquer la séléction basée sur le crowded tournoi sur P_{t+1}

21 pour sélectionner $\mathcal{C}$ et $\mathcal{C}'$;

22 Appliquer l'opérateur de croisement avec la probabilité p_N^C

23 sur $\mathcal{C}$ et $\mathcal{C}'$ pour créer $\mathcal{O}$ et $\mathcal{O}'$;

24 Appliquer l'opérateur de mutation avec la probabilité p_N^M

25 sur $\mathcal{O}$ et $\mathcal{O}'$;

26 $Q_{t+1} = Q_{t+1} \cup \{\mathcal{O}\} \cup \{\mathcal{O}'\}$;

27 $i = i + 1$;

28 FinTantQue

29 $t = t + 1$;

30 FinTantQue

31 Sortie :Front F_1 de l'ensemble $P_{tmax} \cup Q_{tmax}$

FIGURE 4.14 – Algorithme mémétique TS-NSGA-II

0 /* TS_ϵ : Recherche Tabou */

1 /* P_0 : Population initiale */

2 /* $\bar{P}$: Archive externe */

3 /* tmax : Le nombre maximum des générations */

4 Lancer TS_ϵ : la population initiale P_0 contient les individus générés par TS_ϵ

5 $t = 0$)

6 TantQue ($t < tmax$)

7 Affecter une fitnesse pour chaque individu de $P_t \cup \bar{P}$

8 Appliquer la sélection par tournoi binaire avec ces valeurs de fitnesse,

9 l'opérateur de croisement et l'opérateur de mutation pour créer P_{t+1}

10 à partir de $P_t \cup \bar{P}$

11 Mise à jour de $\bar{P}$ (suppression des individus dominés)

12 $t = t + 1$

13 FinTantQue

14 Sortie : $\bar{P}$

FIGURE 4.15 – Algorithme mémétique TS-SPEA

CHAPITRE 5

EXPÉRIMENTATIONS NUMÉRIQUES

Après la mise au point des méthodes de résolution pour notre problème, il ne reste plus qu'à discuter sur les résultats de ces différentes méthodes. Des métriques d'évaluation ont été utilisées afin de pouvoir déterminer la ou les méthodes performantes.

5.1 La génération des données

Lors de la résolution des problèmes d'ordonnancements, les méthodes cherchent à calculer la meilleure approximation du front Pareto à partir des données générées aléatoirement (on appelle aussi des instances). La génération aléatoire des données est un point crucial dans la comparaison des différentes méthodes.

La génération des instances pour notre problème d'ordonnancement bicritère se fait aléatoirement à l'aide d'un générateur conçu exclusivement pour pouvoir couvrir le plus possible de configuration de r_i p_i d_i et V_j.

Le nombre de travaux n prend ses valeurs dans l'ensemble $\{10, 20, 30, 40, 50, 60, 70, 80, 90\}$. Le nombre de machines m prend ses valeurs dans l'ensemble $\{2, 4, 6\}$. Pour maîtriser la distribution des dates échues d_i, et les dates de début au plus tôt r_i, nous utilisons deux paramètres L et R. Le paramètre L permet de centrer la distribution des d_i et celle des r_i, tandis que R contrôle leur variance. Pour chaque problème de taille $(n; m)$, on teste différentes valeurs du paramètre R : $R \in \{0.2; 0.4; 0.6; 0.8; 1.0; 1.2; 1.4; 1.6\}$. L est fixé dynamiquement en fonction des données générées. Les vitesses V_j des machines sont générées aléatoirement selon une loi uniforme sur l'intervalle $[1; 10]$. Les temps opératoires p_i sont générés aléatoirement selon une loi uniforme sur l'intervalle $[10; 100]$. On note par $\overline{V}$ la somme des vitesses générées, i.e. $\overline{V} = \sum_{j=1}^{m} V_j$. V_{max} correspond à la vitesse maximum, i.e. $V_{max} = max_{j=1,\ldots,m}(V_j)$ et V_{min} correspond à la vitesse minimum, i.e. $V_{min} = min_{j=1,\ldots,m}(V_j)$. De même, $\overline{P}$ désigne la somme des temps opératoires générés, i.e. $\overline{P} = \sum_{i=1}^{n} p_i$. Le paramètre L prend

automatiquement une valeur suffisamment grande définie par $L = \frac{\overline{P}}{V_{min}}$. La date échue d_i pour chaque travail J_i est générée aléatoirement entre $\left(L - \frac{R \times \overline{P}}{2\overline{V}}\right)$ et $\left(L + \frac{R \times \overline{P}}{2\overline{V}}\right)$ en utilisant une loi uniforme. De la même façon les dates de début au plus tôt r_i sont générées dans l'intervalle $\left[d_i - 3\frac{p_i}{V_{max}}; d_i - \frac{p_i}{V_{max}}\right]$. Dans le cas où $r_i < 0$, on pose $d_i = d_i - r_i$ et $r_i = 0$. Pour chaque triplet $(n; m; R)$, 30 instances sont générées ce qui nous conduit à un total de 240 instances par taille de problème $(n; m)$.

5.2 Les métriques d'évaluation

L'évaluation de l'efficacité des méthodes se fait en comparant les différents résultats calculés lors de la phase de l'optimisation, sachant que les résultats sont des ensembles, la détermination du meilleur ensemble n'est pas toujours facile voire impossible. Des mesures (ou métrique) ont été introduite pour évaluer l'efficacité des méthodes approchées dans le cas multicritère [Jaszkiewicz, 2004]. Différentes mesures existe dans la littérature, dans notre travail, nous nous intéressons aux mesures qui fournissent des informations concernant la comparaison entre une approximation et la meilleure approximation connue, ou même l'ensemble exacte ZE (approximation calculée par une méthode) et des mesures qui fournissent des informations concernant la répartition des solutions le long de l'approximation. Dans la suite de cette section, nous allons définir quatre mesures utilisées dans notre mémoire.

Soient A,B deux méthodes approchées et Soient ZE_A, respectivement ZE_B, l'approximation calculée par l'algorithme A, respectivement par l'algorithme B et soient $ZE_A^* \subseteq ZE_A$, respectivement $ZE_B^* \subseteq ZE_B$, l'ensemble des vecteurs de critères non dominés par ZE_B, respectivement ZE_A. Nous définissons également $ZE^* \subseteq ZE_A^* \cup ZE_B^*$, la meilleur approximation connue, c'est à dire ZE'^* contient les vecteurs de critères non dominés qui appartiennent à $ZE_A \cup ZE_B$.

Mesure de qualité : Cette mesure désigne le pourcentage des solutions dans l'approximation Z et qui sont également dans ZE^*. Nous avons

$$Q_1(Z) = 100 \times \frac{|ZE^* \cap Z|}{|Z|}.$$

Pour chaque couple (n, m), dans les 240 instances générées, nous calculons la valeur moyenne de $Q_1(Z_A)$ et $Q_1(Z_B)$.

Mesure de distance : Cette mesure que nous notons $Q_2(Z)$, permet de calculer la distance moyenne du front Z par rapport au front de référence ZE^*. Nous avons

$$Q_2(Z) = \frac{1}{|ZE^*|} \sum_{z \in ZE^*} \min_{z' \in Z}(d(z, z'))$$

avec $d(z, z')$ la distance euclidienne entre z et z' dans $\mathbb{R}^2$. Également, pour chaque couple (n, m) nous calculons la valeur moyenne de $Q_2(Z_A)$ et $Q_2(Z_B)$.

Mesure de quantité : Cette mesure consiste à comparer le front ZE_A, respectivement ZE_B, au front de référence ZE^*. Soit $Q_1(Z)$ le pourcentage des solutions potentiellement non dominées dans Z et qui sont également dans ZE^*. Nous avons

$$Q_3(Z) = 100 \times \frac{|ZE^* \cap Z|}{|ZE^*|}.$$

Pour chaque couple (n, m), nous calculons la valeur moyenne de $Q_3(Z_A)$ et $Q_3(Z_B)$.

Mesure de répartition : Cette mesure consiste à estimer la répartition des solutions le long de l'approximation. Nous considérons $d'(z)$ comme la distance absolue de $z \in Z$, et elle est définie par $d'(z) = min_{z' \in Z, z' \neq z} \sum_{k=1}^{2} |z_k - z'_k|$. Ensuite nous considérons $\bar{d'}$, la distance moyenne des d' sur l'ensemble d'approximation Z : $\bar{d'} = \frac{1}{|Z|} \sum_{z \in Z} d'(z)$. La mesure de répartition de l'ensemble est définie par :

$$Q_4(Z) = \sqrt{\frac{1}{|Z|-1} \sum_{z \in Z} (\bar{d'} - d'(z))^2}$$

5.3 Comparaison des méthodes

Les instances utilisées dans les tests numériques de toutes les méthodes vues dans le chapitre précédent à savoir les deux algorithmes génétique NSGA-II et SPEA, les trois versions de recherche tabou (l'approche tabou basée sur la marche aléatoire (TS_{rw}), approche par combinaison linéaire des critères (TS_{cl}) et approche par $\epsilon - contrainte$ (TS_ϵ)) ainsi que les deux hybridations (TS-NSGA-II et TS-SPEA) sont les même que celles décrites dans la section précédente.

Pour l'évaluation de l'efficacité des méthodes étudiées, nous utiliserons deux aspect : Le premier est le temps d'exécution requis pour retourner une estimation de l'ensemble des optima de Pareto stricts. Le second aspect auquel nous nous intéressons est la qualité de l'approximation calculée par les méthodes, nous utiliserons quatres mesures (mesure de qualité, mesure de distance, mesure de quantité et mesure de répartition) que nous avons définies précédemment.
Notre travail commence par la comparaison entre les trois versions de recherche tabou (TS_{rw}, TS_{cl} et TS_ϵ), la plus performante par rapport aux métriques d'évaluation sera évalué aux deux algorithmes génétiques NSGA-II et SPEA et enfin nous comparons les deux algorithmes mémétiques.
Afin de pouvoir évoluer correctement nos méthodes approchées, nous avons choisit de réaliser nos expérimentations sur des instances générées aléatoirement que nous avons décrites précédemment et qui peuvent présenter au mieux toutes les instances possibles.

5.3.1 Comparaison de TS_{rw}, TS_{cl} et TS_ϵ

Dans cette section, nous comparons les trois versions de recherche tabou. La première est la recherche tabou basée sur une marche aléatoire notée TS_{rw}, la deuxième est la recherche tabou par

combinaison linéaire des critères notée TS_{cl} et la dernière est l'approche tabou par epsilon-contrainte notée TS_{ϵ}.

Résultats expérimentaux par rapport aux métriques

Les tableaux 5.1, 5.2 et 5.3 récapitulent les différents résultats de l'expérimentation.

Dans le tableau 5.1 : la première colonne présente le nombre de travaux n, la seconde correspond au nombre de machines m, la troisième, quatrième et cinquième colonnes désignent la mesure de qualité pour l'approche TS_{rw}, l'approche TS_{cl} et l'approche TS_{ϵ} notées respectivement $Q_{1-TS_{rw}}$, $Q_{1-TS_{cl}}$ et $Q_{1-TS_{\epsilon}}$. Enfin, les dernières colonnes corespondent au mesure de distance pour les trois approches notées respectivement $Q_{2-TS_{rw}}$, $Q_{2-TS_{cl}}$ et $Q_{2-TS_{\epsilon}}$.

Dans le tableau 5.2 : la première colonne présente le nombre de travaux n, la seconde correspond au nombre de machines m, la troisième, quatrième et cinquième colonnes désignent la mesure de quantité pour l'approche TS_{rw}, l'approche TS_{cl} et l'approche TS_{ϵ} notées respectivement $Q_{3-TS_{rw}}$, $Q_{3-TS_{cl}}$ et $Q_{3-TS_{\epsilon}}$. Enfin, les dernières colonnes corespondent au mesure de répartition pour les trois approches notées respectivement $Q_{4-TS_{rw}}$, $Q_{4-TS_{cl}}$ et $Q_{4-TS_{\epsilon}}$

D'après les résultats expérimentaux présentés dans les tableaux 5.1 et 5.2, nous constatons que TS_{ϵ} est préférable à TS_{rw} et TS_{cl} par rapport à la mesure de qualité Q_1 et de quantité Q_2 avec une différence respective qui dépasse 60% pour Q_1 et de 50% pour Q_3. Par exemple, pour la mesure de qualité Q_1, la différence entre TS_{ϵ} et TS_{rw} est de 66.94% et entre TS_{ϵ} et TS_{cl} est de 61.12% et pour la mesure de quantité Q_2, la différence entre TS_{ϵ} et TS_{rw} est de 62.74% et entre TS_{ϵ} et TS_{cl} est de 53.8%.
Nous remarquons également que TS_{rw} est préférable à TS_{ϵ} et TS_{cl} avec une différence respective de 9.29% et 37.85% par rapport à la mesure de distance Q_2 et une différence respective de16.01% et 17.05% par rapport à la mesure de répartition Q_4. Donc, on peut déduire que l'approche epsilon-contrainte TS_{ϵ} présente de meilleurs résultats.

5.3.2 Evaluation expérimentale de NSGA-II, SPEA et TS_{ϵ}

Dans cette section, nous comparons les trois méthodes décrites dans le chapitre précédent à savoir NSGA-II, SPEA et la recherche tabou par ϵ-contrainte notée TS_{ϵ}. Les résultats de la résolution du problème $Q/r_i, d_i/C_{max}, L_{max}$ par ces trois méthodes sont présentés tableaux 5.4, 5.5, 5.6, 5.7

n	m	$Q_{1-TS_{rw}}$	$Q_{1-TS_{cl}}$	$Q_{1-TS_{\epsilon}}$	$Q_{2-TS_{rw}}$	$Q_{2-TS_{cl}}$	$Q_{2-TS_{\epsilon}}$
10	2	9.11%	31.58%	69.26%	59.72%	24.17%	3.20%
	4	41.45%	46.49%	52.98%	11.84%	2.71%	12.12%
	6	45.76%	47.08%	52.78%	2.92%	0.00%	17.08%
20	2	4.52%	13.92%	86.08%	52.56%	41.28%	0.74%
	4	19.89%	36.2%	63.35%	45.81%	13.66%	8.44%
	6	40.74%	45.17%	54.28%	10.83%	0.83%	19.58%
30	2	3.65%	6.22%	93.99%	48.66%	46.73%	0.86%
	4	15.56%	26.39%	73.47%	43.88%	26.91%	5.46%
	6	27.58%	38.79%	59.47%	34.41%	11.41%	10.43%
40	2	1.63%	3.01%	96.99%	48.40%	49.50%	0.02%
	4	13.11%	20.69%	79.03%	45.76%	33.40%	5.84%
	6	18.82%	34.19%	64.51%	48.64%	16.36%	8.76%
50	2	2.71%	3.54%	97.92%	48.02%	49.02%	0.04%
	4	11.58%	16.12%	83.88%	46.12%	37.63%	1.67%
	6	17.46%	28.51%	71.49%	43.37%	23.29%	5.42%
60	2	1.39%	1.94%	98.06%	49.29%	49.46%	0.00%
	4	9.7%	13.27%	86.73%	46.30%	39.12%	0.83%
	6	14.91%	22.42%	76.6%	45.42%	29.99%	2.50%
70	2	0.52%	1.67%	98.33%	50.13%	49.18%	0.28%
	4	8.66%	10.86%	89.14%	43.54%	44.78%	0.84%
	6	13.27%	18.89%	81.11%	42.38%	35.95%	2.08%
80	2	0.62%	1.17%	98.83%	48.52%	51.46%	0.02%
	4	8.61%	10.5%	89.5%	42.71%	43.53%	1.67%
	6	14.21%	17.79%	82.07%	42.52%	34.98%	5.42%
90	2	0.97%	1.53%	98.89%	48.40%	50.34%	0.43%
	4	8.83%	10.01%	89.99%	44.14%	45.02%	1.25%
	6	10.84%	15.35%	84.65%	44.89%	35.61%	2.08%
Moyenne		13.56%	19.38%	80.5%	42.19%	32.90%	4.34%

TABLE 5.1 – Les valeurs des mesures d'évaluation Q_1 et Q_2

n	m	$Q_{3-TS_{rw}}$	$Q_{3-TS_{cl}}$	$Q_{3-TS_{\epsilon}}$	$Q_{4-TS_{rw}}$	$Q_{4-TS_{cl}}$	$Q_{4-TS_{\epsilon}}$
10	2	18.37%	56.90%	88.16%	28.11%	20.69%	13.28%
	4	88.13%	97.50%	92.50%	1.04%	4.79%	12.51%
	6	97.08%	100%	91.39%	0.42%	0.42%	13.33%
20	2	9.79%	24.84%	88.39%	34.42%	20.65%	12.43%
	4	44.79%	76.74%	91.53%	17.99%	13.59%	18.01%
	6	89.38%	99.17%	90.61%	1.67%	2.08%	18.76%
30	2	7.11%	9.62%	90.42%	39.63%	21.57%	8.80%
	4	36.46%	53.21%	93.16%	25.88%	18.15%	15.14%
	6	61.94%	86.22%	91.35%	13.25%	8%	22.09%
40	2	3.33%	5%	91.84%	41.93%	17.59%	9.23%
	4	29.83%	40.26%	94.20%	34.17%	21.5%	13.08%
	6	43.24%	73.33%	91.04%	23.66%	10.49%	18.35%
50	2	3.96%	4.41%	93.16%	46.65%	16.47%	9.38%
	4	25.49%	30.71%	94.92%	37.16%	13.28%	17.48%
	6	40.90%	59.26%	92.71%	31.05%	17.68%	18.77%
60	2	3.13%	2.86%	92.92%	43.42%	18.97%	7.20%
	4	22.92%	25.60%	96.28%	45.31%	13.71%	15.56%
	6	35.28%	47.17%	94.14%	38.49%	14.39%	15.05%
70	2	1.25%	2.50%	93.46%	41.97%	22.85%	8.10%
	4	20.24%	19.66%	96.65%	36%	20.71%	18.71%
	6	31.32%	37.48%	97.07%	35.29%	15.81%	20.45%
80	2	1.46%	1.46%	93.40%	41.82%	21.55%	11.63%
	4	22.85%	21.82%	95.75%	39.91%	17.35%	16.90%
	6	35.21%	37.03%	94.41%	38.32%	14.57%	19.19%
90	2	1.67%	2.43%	92.92%	35.53%	21.29%	9.85%
	4	20.28%	18.05%	97.19%	43.11%	15.94%	16.78%
	6	26.39%	30.70%	96.74%	37.98%	17.41%	13.77%
Moyenne		30.46%	39.40%	93.20%	31.64%	15.61%	14.59%

TABLE 5.2 – Les valeurs des mesures d'évaluation Q_3 et Q_4 pour les trois approches

Les algorithmes génétiques NSGA-II et SPEA dépond d'un ensemble de paramètres qui influence sur leurs efficacités. Ces paramètres sont la probabilité de croisement, la probabilité de mutation, la taille de la population (nombre d'individus) et le nombre d'itérations, le tableau suivant résume les valeurs de ces paramètres pour les deux algorithmes génétiques.

NSGA-II		SPEA	
Parameter	Value	Parameter	Value
p_N^C	0.9	p_S^C	0.8
p_N^M	0.05	p_S^M	0.5
N	$3 \times n$	N	$3 \times n$
t_{max}	50	t_{max}	50

TABLE 5.3 – Les paramètres de NSGA-II et SPEA

Résultats expérimentaux par rapport aux temps d'exécution

Pour l'approche par ϵ-contrainte (TS_ϵ), le nombre d'itérations est fixé à 30 et la taille de la liste tabou est fixé à 10.

Dans le tableau 5.4 et 5.6 la première colonne présente le nombre de travaux n, la seconde correspond au nombre de machines m, la troisième, quatrième et cinquième colonnes désignent le temps minimum pour la méthode NSGA-II, la méthode SPEA et la méthode TS_ϵ notés respectivement $t_{min-NSGA-II}$,$t_{min-SPEA}$ et t_{min-TS_ϵ}. La sixième , septième et huitième colonnes présentent les temps moyens des trois méthodes notés $t_{moy-NSGA-II}$,$t_{moy-SPEA}$ et t_{moy-TS_ϵ}. Enfin, les trois dernières colonnes désignent les temps maximum des trois méthodes notés $t_{max-NSGA-II}$,$t_{max-SPEA}$ et t_{max-TS_ϵ}. Tous les temps sont calculés en seconde.

n	m	$t_{min-NSGA-II}$	$t_{min-SPEA}$	t_{min-TS_ϵ}	$t_{moy-NSGA-II}$	$t_{moy-SPEA}$	t_{moy-TS_ϵ}
	2	0.734	0.188	0	0.8131	0.236	0.009
10	4	0.7341	0.39	0	1.5935	0.4776	0.0011
	6	0.7342	0.547	0	2.3718	0.7432	0.0012
	2	5.937	1.406	0	6.5927	1.5931	0.0024
20	4	11.14	2.687	0	13.5153	3.1615	0.0022
	6	18.266	4.125	0	21.116	4.9087	0.0017
	2	21.843	4.64	0	26.1945	5.0913	0.0052
30	4	48.359	9.016	0	61.3505	10.1598	0.0036
	6	70.484	13.484	0	101.0253	15.5653	0.0025
	2	58.922	10.329	0	83.3071	11.4454	0.011
40	4	137.032	20.5	0	211.0669	22.635	0.005
	6	254.688	30.937	0	366.4149	34.0753	0.004
	2	142.391	20.531	0	219.2192	22.6821	0.02
50	4	382.703	40.938	0	590.7713	45.86	0.0088
	6	677.906	62.453	0	1077.524	69.411	0.0078
	2	360.016	37.391	0.015	524.9791	40.0377	0.0326
60	4	1087.844	72.844	0	1470.8645	80.1741	0.0442
	6	–	114.594	0	–	123.3283	0.0704

TABLE 5.4 – Temps de calcul pour les trois méthodes

Le tableau suivant est la suite du tableau des temps de calcul pour les trois méthodes.

n	m	$t_{max-NSGA-II}$	$t_{max-SPEA}$	t_{max-TS_ϵ}
10	2	0.89	0.281	0.015
	4	1.719	0.594	0.015
	6	2.594	0.984	0.015
20	2	7.422	1.875	0.015
	4	15.829	3.937	0.015
	6	24.719	6.328	0.015
30	2	32.781	5.89	0.015
	4	78.578	12.109	0.015
	6	133.350	21.015	0.015
40	2	103.407	12.813	0.031
	4	286.266	25.125	0.015
	6	503.813	38.297	0.015
50	2	298.453	25.593	0.046
	4	893.1411	52.703	0.093
	6	1644.0621	78.718	0.156
60	2	707.2501	43.734	0.062
	4	1758.4681	87.094	0.234
	6	–	138.17	0.312

TABLE 5.5 – Temps de calcul pour les trois méthodes (suite)

D'après les résultats expérimentaux présentés dans les tableaux 5.4 et 5.5, on constate que la méthode TS_ϵ est la plus rapide que NSGA-II et SPEA et la différence est très significative cela revient au nombre d'individu (ensemble d'individus) que traitent les deux algorithmes génétiques contrairement à la TS_ϵ qui traite un seul individu à la fois, nous remarquons également que NSGA-II prend un temps considérable pour résoudre une instance par exemple, une instance de 50 travaux et 6 machines le temps moyen d'exécution est de 1077.52 secondes cela fait presque 18 minutes, SPEA met 69.39 s tandis que la recherche tabou TS_ϵ la résoud en 0.02 secondes. De plus, si on compare les deux méthode génétique, on peut constater que SPEA est meilleure en temps d'exécution que NSGA-II.

Dans le tableau 5.6 : la première colonne présente le nombre de travaux n, la seconde correspond au nombre de machines m, la troisième, quatrième et cinquième colonnes désignent le nombre d'instances non résolus dont le temps dépasse la limite de temps de 1800 seconde (30 mn) pour la méthode NSGA-II, la méthode SPEA et la méthode TS_ϵ notés respectivement $NNR_{NSGA-II}$, NNR_{SPEA} et NNR_{TS_ϵ}.

n	m	$NNR_{NSGA-II}$	NNR_{SPEA}	NNR_{TS_ϵ}
10	2	0	0	0
	4	0	0	0
	6	0	0	0
20	2	0	0	0
	4	0	0	0
	6	0	0	0
30	2	0	0	0
	4	0	0	0
	6	0	0	0
40	2	0	0	0
	4	0	0	0
	6	0	0	0
50	2	0	0	0
	4	0	0	0
	6	1	0	0
60	2	0	0	0
	4	14	0	0
	6	80	0	0

TABLE 5.6 – Nombre d'instances non résolues

Le tableau 5.6 présente les résultats expérimentaux comparant le nombre d'instances dont la résolution dépassent 1800 secondes. Nous remarquons que les instances 10,20,30,40,50 travaux (sauf une instance de 50 travaux et 6 machines dont le temps de résolution dépasse 1800s) sont toutes résolues par les trois méthodes dans la limite de temps fixé. Par contre, à partir de 60 travaux et 4 machines nous avons un certain pourcentage d'instances dont le temps dépasse 1800 secondes. A savoir 17.5% et 100% respectivement pour les instances 60 travaux et 4 machines et 60 travaux et 6 machines pour la méthode NSGA-II.

Résultats expérimentaux par rapport aux métriques

Nous avons vu au début de ce chapitre un autre aspect de comparaison entre les méthodes autre que le temps d'exécution, il s'agit de la qualité de l'approximation calculés par les trois algorithmes. Les tableaux 5.7 et 5.8 récapitulent les différents résultats de l'expérimentation.

Dans le tableau 5.7 : la première colonne présente le nombre de travaux n, la seconde correspond au nombre de machines m, la troisième, quatrième et cinquième colonnes désignent la mesure de qualité pour la méthode NSGA-II, la méthode SPEA et la méthode TS_ϵ notées respectivement $Q_{1-NSGA-II}$, Q_{1-SPEA} et Q_{1-TS_ϵ}. Enfin, les dernières colonnes corespondent au mesure de ditance pour les trois méthodes notées respectivement $Q_{2-NSGA-II}$, Q_{2-SPEA} et Q_{2-TS_ϵ}.

n	m	$Q_{1-NSGA-II}$	Q_{1-SPEA}	Q_{1-TS_ϵ}	$Q_{2-NSGA-II}$	Q_{2-SPEA}	Q_{2-TS_ϵ}
	2	18.7%	2.57%	95.56%	44.05%	55.54%	0.41%
10	4	0%	0.14%	99.86%	50.19%	49.81%	0%
	6	0%	0%	100%	50.01%	49.99%	0%
	2	04.10%	1.53%	94.38%	37.99%	61.20%	0.81%
20	4	01.63%	0.49%	97.88%	45.93%	54.07%	0%
	6	0%	0%	100%	49.42%	50.58%	0%
	2	08.54%	0.35%	91.11%	30.80%	69.12%	0.08%
30	4	04.41%	0.49%	95.10%	36.24%	63.76%	0%
	6	0.45%	0.21%	99.34%	44.22%	55.78%	0%
	2	20.45%	0%	79.55%	27.06%	72.93%	0.01%
40	4	07.90%	0%	92.10%	30.89%	69.11%	0%
	6	4.20%	0%	95.80%	36.48%	63.52%	0%
	2	32.5%7	0%	67.43%	24.41%	75.58%	0%
50	4	25.68%	0%	74.32%	26.72%	73.28%	0%
	6	28.37%	0%	71.63%	32.34%	67.66%	0%
	2	34.69%	3.75%	65.31%	21.38%	78.61%	7.5%
60	4	21.46%	0%	78.54%	26.33%	73.67%	0%
	6	0%	6.25%	91.25%	0%	13.62%	23.75%
Moyenne		11.55%	0.88%	88.29%	36.15%	60.99%	1.81%

TABLE 5.7 – Les valeurs des mesures d'évaluation Q_1 et Q_2

Dans le tableau 5.8 : la première colonne présente le nombre de travaux n, la seconde correspond au nombre de machines m, la troisième, quatrième et cinquième colonnes désignent la mesure de quantité pour la méthode NSGA-II, la méthode SPEA et la méthode TS_ϵ notées respectivement $Q_{3-NSGA-II}$, Q_{3-SPEA} et Q_{3-TS_ϵ}. Enfin, les dernières colonnes corespondent au mesure de répartition pour les trois méthodes notées respectivement $Q_{4-NSGA-II}$, Q_{4-SPEA} et Q_{4-TS_ϵ}

n	m	$Q_{3-NSGA-II}$	Q_{3-SPEA}	Q_{3-TS_ϵ}	$Q_{4-NSGA-II}$	Q_{4-SPEA}	Q_{4-TS_ϵ}
	2	4.58%	3.82%	87.74%	0%	19.97%	27.12%
10	4	0%	0.42%	92.50%	0%	0%	14.17%
	6	0%	0%	91.39%	0%	0%	13.33%
	2	6.39%	2.31%	88.39%	4.57%	28.98%	19.79%
20	4	2.71%	1.25%	91.53%	2.07%	10.37%	25.05%
	6	0%	0%	90.61%	0%	2.5%	19.17%
	2	9.69%	0.42%	90.42%	14.04%	25.5%	15.05%
30	4	6.67%	1.25%	93.16%	6.58%	30.3%	16.87%
	6	0.83%	0.21%	91.35%	1.24%	17.08%	22.93%
	2	20.21%	0%	91.84%	33.06%	18.03%	14.33%
40	4	9.44%	0%	94.20%	13.32%	26.89%	21.04%
	6	5.63%	0%	91.04%	10.38%	24.15%	20.47
	2	29.31%	0%	93.16%	27.42%	11.96%	13.54%
50	4	29.08%	0%	94.5%	23.68%	19.73%	18.25%
	6	34.31%	0%	91.42%	12.09%	28.89%	23.45%
	2	27.73%	3.75%	90.63%	18.74%	11.22%	13.79%
60	4	24.22%	0%	94.95%	19.69%	21.19%	30.34%
	6	0%	8.75%	89.79%	0%	32.50%	31.25%
Moyenne		12.40%	1.23%	91.59%	10.99%	18.29%	20%

TABLE 5.8 – Les valeurs des mesures d'évaluation Q_3 et Q_4 pour les trois méthodes

La constation que l'on peut effectuer est qu'expérimentalement l'approche tabou par epsilon-contrainte domine les deux algorithmes génétiques NSGA-II et SPEA par rapport aux mesure de qualité Q_1, de quantité Q_2 et de répartition Q_4 avec une différence respective de 76.74% pour Q_1, 79.19% pour Q_3 et 9.01% pour Q_4 par rapport à NSGA-II et avec une différence respective de 87.41% pour Q_1,

59.18% pour Q_3 et 1.71% pour Q_4 par rapport à l'algorithme génétique SPEA. Par contre pour la mesure de distance Q_2, on remarque que SPEA est préférable à NSGA-II et TS_ϵ avec une différence respective de 24.84% et 59.86%.

A travers ces résultats on peut donc en conclure que l'approche tabou par epsilon-contrainte TS_ϵ est la plus performante que les deux algorithmes génétiques NSGA-II et SPEA par rapport à la plus part des mesures d'évaluation et au temps d'exécution.

5.3.3 Evaluation expérimentales des deux hybridations

La première hybridation combine entre l'algorithme génétique NSGA-II et la recherche tabou par epsilon-contrainte notée $TS - NSGA - II$ et la seconde hybridation combine l'algorithme génétique SPEA et la recherche tabou par epsilon contrainte notée $TS - SPEA$. Chacun des algorithmes génétiques tire sa population initiale de l'algorithme tabou.

Résultats expérimentaux par rapport aux temps d'exécution

Dans le tableau 5.9 : la première colonne présente le nombre de travaux n, la seconde correspond au nombre de machines m et la troisième quatrième colonnes désignent le temps minimum pour l'hybridation $TS-NSGA-II$ et l'hybridation $TS-SPEA$ notés respectivement $t_{min-TS-NSGA-II}$ et $t_{min-TS-SPEA}$. La cinquième et sixième colonnes présentent les temps moyen des deux méthodes notés $t_{moy-TS-NSGA-II}$ et $t_{moy-TS-SPEA}$. Enfin les deux dernières colonnes désignent les temps maximum notés $t_{max-TS-NSGA-II}$ et $t_{max-TS-SPEA}$.

Dans le tableau 5.10 : la première colonne présente le nombre de travaux n, la seconde correspond au nombre de machines m, la troisième, quatrième colonnes désignent le nombre d'instances non résolus dont le temps dépasse la limite de temps de 900 seconde (15 mn) pour la méthode TS-NSGA-II et la méthode TS-SPEA notées respectivement $NNR_{TS-NSGA-II}$ et $NNR_{TS-SPEA}$.

D'après les résultas expérimentaux présentés dans les tableaux 5.9 et 5.10, on constate qu'à partir de 70 travaux et 4 machines on commence à avoir des instances dont le temps dépasse les 900s et cela pour les deux algorithmes mémétiques $TS-NSGA-II$ et $TS-SPEA$. Nous remarquons également que la méthode $TS - NSGA - II$ présente de meilleurs résultats par rapport au temps moyen mais la différence n'est pas très significative, elle est de 7.1629s.

Résultats expérimentaux par rapport aux métriques

Dans cette section, nous comparons les deux hybridations en utilisant les quatres métriques. Les tableaux suivants résument les différents résultats obtenus lors des tests effectués sur ces deux hybridations.

Dans le tableau 5.11 : la première colonne présente le nombre de travaux n, la seconde correspond au nombre de machines m, la troisième, quatrième colonnes désignent la mesure de qualité pour la méthode mémétique TS-NSGA-II et la méthode mémétique TS-SPEA notées respectivement $Q_{1-TS-NSGA-II}$ et $Q_{1-TS-SPEA}$. Enfin, les dernières colonnes corespondent au mesure de distance pour les deux méthodes notées respectivement $Q_{2-TS-NSGA-II}$, $Q_{2-TS-SPEA}$.

		$TS-NSGA2$	$TS-SPEA$	$TS-NSGA2$	$TS-SPEA$	$TS-NSGA2$	$TS-SPEA$
n	m	t_{min}	t_{min}	t_{moy}	t_{moy}	t_{max}	t_{max}
	2	0.015	0	0.0938	0.0932	0.485	0.485
10	4	0.016	0.016	0.1116	0.1126	0.875	0.875
	6	0.013	0.031	0.1535	0.1542	1.297	1.312
	2	0.015	0.015	1.154	1.1525	3.344	3.36
20	4	0.046	0.078	0.8643	0.9012	6.469	6.515
	6	0.093	0.093	0.572	0.5374	3.109	2.281
	2	0.077	0.016	6.5321	6.5331	29.406	29.64
30	4	0.109	0.109	4.7623	4.9921	27.016	27.156
	6	0.125	0.125	4.188	3.9644	47.828	48
	2	0.063	0.031	14.14	14.1381	34	34
40	4	0.188	0.188	15.9797	16.7313	71.64	71.64
	6	0.186	0.171	14.1197	13.3698	166.077	166.499
	2	0.108	0.062	32.5078	32.9499	67.344	67.343
50	4	0.188	0.187	45.4659	45.0678	230.969	230.968
	6	0.484	0.453	36.7809	36.7621	296.624	297.484
	2	0.6250	0.093	72.4030	72.4059	133.594	133.625
60	4	0.0930	0.2340	129.4870	129.5051	556.734	557.609
	6	0.359	0.3430	46.3410	46.6543	653.624	653.593
	2	1.1550	0.1710	122.5468	124.1971	266.469	267.562
70	4	0.3590	0.2810	200.9030	203.53	824.767	826.516
	6	0.5770	0.5460	134.4438	129.3052	798.484	800.953
	2	0.1560	0.1400	235.9369	238.9545	664.687	664.671
80	4	0.8590	0.5160	284.4771	288.9926	797.156	799.922
	6	0.64	0.5780	108.5403	99.0813	892.015	892.015
	2	0.7650	0.1560	308.6382	312.9807	665.562	665.578
90	4	1.2810	0.8280	337.1301	331.9638	892.515	896.562
	6	0.5150	0.3750	585.781	395.625	891.047	896.875

TABLE 5.9 – Temps de calcul pour les méthodes TS-NSGA-II et TS-SPEA

n	m	$NNR_{TS-NSGA-II}$	$NNR_{TS-SPEA}$
10	2	0	0
	4	0	0
	6	0	0
20	2	0	0
	4	0	0
	6	0	0
30	2	0	0
	4	0	0
	6	0	0
40	2	0	0
	4	0	0
	6	0	0
50	2	0	0
	4	0	0
	6	0	0
60	2	0	0
	4	0	0
	6	0	0
70	2	0	0
	4	1	1
	6	6	6
80	2	0	0
	4	8	8
	6	14	14
90	2	0	0
	4	11	11
	6	18	18

TABLE 5.10 – Nombre d'instances non résolues

Dans le tableau 5.12 : la première colonne présente le nombre de travaux n, la seconde correspond au nombre de machines m, la troisième, quatrième colonnes désignent la mesure de quantité pour la méthode mémétique TS-NSGA-II et la méthode mémétique TS-SPEA notées respectivement $Q_{3-TS-NSGA-II}$ et $Q_{3-TS-SPEA}$. Enfin, les dernières colonnes corespondent au mesure de répartition pour les deux méthodes notées respectivement $Q_{4-TS-NSGA-II}$, $Q_{4-TS-SPEA}$.

La constatation que l'on peut effectuer est qu'expérimentalement l'algorithme mémétique $TS - SPEA$ combinant l'approche tabou par epsilon-contrainte et l'algorithme génétique SPEA domine l'algorithme mémétique TS-NSGA-II combinant l'approche tabou par epsilon-contrainte et l'algorithme génétique NSGA-II par rapport aux mesures de qualité Q_1, de quantité Q_3 et de répartition Q_4 avec une différence respective de 25.86% pour Q_1, 21.25% pour Q_3 et 18.21% pour Q_4. Par contre, pour la mesure de distance Q_2 on remarque que TS-NSGA-II est préférable à TS-SPEA avec une différence de 13.29%.
On peut donc en conclure qu'en règle générale l'hybridation TS-SPEA est la plus performante que l'hybridation TS-NSGA-II.

5.3.4 Conclusion sur les comparaisons des méthodes entre-elles

De ces expérimentations, on peut dire qu'aucune méthode ne domine strictement les autres. On peut cependant remarquer que l'approche tabou par epsilon-contrainte donne en règle générale les meilleurs résultats. En deuxième position vient l'algorithme mémétique TS-SPEA puis l'algorithme mémétque TS-NSGA-II.

n	m	$Q_{1-TS-NSGA-II}$	$Q_{1-TS-SPEA}$	$Q_{2-TS-NSGA-II}$	$Q_{2-TS-SPEA}$
10	2	70.62%	93.54%	18.75%	5%
	4	85.63%	93.75%	13.75%	6.25%
	6	85.00%	96.25%	15%	3.75%
20	2	53.93%	88.81%	25%	7.50%
	4	73.54%	88.13%	21.25%	11.25%
	6	88.75%	93.75%	11.25%	6.25%
30	2	61.25%	85.63%	12.50%	16.25%
	4	59.35%	86.38%	31.25%	13.75%
	6	81.25%	93.13%	16.25%	8.75%
40	2	44.94%	87.25%	30%	12.50%
	4	51.46%	89.17%	40%	11.25%
	6	75.62%	86.04%	18.75%	13.75%
50	2	44.46%	84.17%	27.5%	16.25%
	4	43.62%	87.63%	42.5%	12.50%
	6	58.13%	91.88%	35%	5%
60	2	38.68%	86.73%	33.33%	16.67%
	4	40.35%	84.65%	44.76%	16.49%
	6	63.54%	87.08%	28.09%	13.16%
70	2	35.5%	92.83%	35.83%	6.67%
	4	52.05%	68.84%	37.97%	30.38%
	6	61.15%	77.37%	28.38%	18.92%
80	2	41.73%	80.60%	22.08%	21.67%
	4	58.56%	65.74%	29.17%	27.78%
	6	61.49%	85.98%	30.30%	12.12%
90	2	47.75%	59.96%	32.5%	37.50%
	4	57.17%	66.01%	28.99%	28.99%
	6	57.39%	89.92%	35.48%	6.45%
Moyenne		59%	84.86%	27.62%	14.33%

TABLE 5.11 – Les valeurs des mesures d'évaluation Q_1 et Q_2

n	m	$Q_{3-TS-NSGA-II}$	$Q_{3-TS-SPEA}$	$Q_{4-TS-NSGA-II}$	$Q_{4-TS-SPEA}$
10	2	81.25%	96.88%	0%	15%
	4	86.25%	93.75%	0%	1.25%
	6	85%	96.25%	0%	0%
20	2	73.54%	93.96%	1.25%	22.50%
	4	78.75%	89.38%	0%	12.50%
	6	88.75%	93.75%	0%	0%
30	2	86.88%	89.17%	1.25%	32.50%
	4	68.75%	88.88%	2.50%	13.75%
	6	83.75%	92.92%	0%	6.25%
40	2	70%	91.94%	1.25%	28.75%
	4	60%	91.25%	1.74%	23.26%
	6	81.25%	87.50%	0%	10%
50	2	48.75%	89.77%	0%	23.75%
	4	50.42%	91.35%	1.25%	26.25%
	6	60.63%	95%	2.11%	11.64%
60	2	41.25%	90.71%	1.25%	21.25%
	4	40.63%	87.46%	1.25%	26.25%
	6	68.13%	88.13%	0%	15%
70	2	40.21%	97.35%	1.42%	37.33%
	4	50.63%	72.05%	0%	26.58%
	6	70.95%	83.45%	0%	17.57%
80	2	47.08%	85.15%	4.96%	20.04%
	4	70.83%	69.21%	0%	26.39%
	6	69.70%	88.38%	0%	16.67%
90	2	57.92%	66.04%	0%	37%
	4	70.29%	68.12%	1.71%	35.97%
	6	64.52%	91.94%	0%	8.06%
Moyenne		66.52%	87.77%	0.81%	19.02%

TABLE 5.12 – Les valeurs des mesures d'évaluation Q_3 et Q_4

	Q_1	Q_2	Q_3	Q_4
TS_{rw}	13.56%	42.19%	30.46%	31.64%
TS_{cl}	19.38%	32.9%	39.4%	15.61%
TS_{ϵ}	88.29%	4.34%	93.2%	14.59%
$NSGA-II$	11.55%	35.15%	12.4%	10.99%
$SPEA$	0.8%	60.99%	1.23%	18.29%
$TS-NSGA-II$	59%	27.62%	66.52%	0.81%
$TS-SPEA$	84.86%	14.33%	87.77%	19.02%

TABLE 5.13 – Comparaison des méthodes entre-elles

CONCLUSION

Dans ce mémoire nous avons abordé et étudié le problème d'ordonnancement bicritère à machines parallèles uniformes. L'objectif est de minimiser simultanément la date de fin de l'ensemble des travaux et leur retard algébrique. Ce problème se note $Q/r_i, d_i/C_{max}, L_{max}$. Nous avons proposé de le résoudre par différentes approches, elles sont basées sur l'énumération d'une approximation de l'ensemble des solutions de meilleurs compromis appelés également *optima de Pareto stricts*. Nous nous somme intéressé par deux types d'approches : le premier type utilise comme solution de départ une population d'individus (ensemble de solutions), il s'agit des algorithmes évolutionnaires NSGA-II et SPEA et le second type concerne les approches utilisant une seule solution au départ (recherche locale). Enfin, nous avons combiné les deux types pour former les méthodes hybrides.

Dans le premier chapitre, nous avons donné des généralités sur les problèmes d'ordonnancement, leurs concepts fondamentaux à savoir leurs contraintes, leurs objectifs et leurs classifications. Des notions sur la théorie de la complexité ont été aussi présentées.

Le chapitre 2 est dédié aux problèmes multi-objectif en général et plus particulièrement aux problèmes d'ordonnancement multicritère. Nous avons définit des notions liées à l'optimalité et nous avons présenter des méthodes de résolution des problèmes d'ordonnacement.

Le chapitre 3 est consacré à l'état de l'art du problème d'ordonnacement bicritère que nous avons traité dans ce mémoire. Nous nous somme intéressé à la minimisation simultanément de makespan et de retard algèbrique du problème d'ordonnancement à machines parallèles uniformes $Q/r_i, d_i/C_{max}, L_{max}$.

Dans le chapitre 4, nous avons résolu notre problème d'ordonnacement par deux algorithmes génétiques NSGA-II et SPEA. Lors de la résolution du problème traité par ces deux algorithmes génétiques, nous avons remarqué que le temps d'exécution est en extrème croissance, cela revient au nombre important d'individus que traitent les algorithmes. Alors, nous avons opté pour une méthode traitant moins d'individu, la recherche tabou semble la mieux adapté. Nous avons développé trois

versions de recherche tabou pour la détermination d'optima de Pareto : une marche aléatoire, une combinaison linéaire des critères et une approche epsilon-contrainte. Et pour terminer, nous avons résolu notre problème par deux hybridations combinant une méthode de recherche tabou avec chacun des algorithmes génétiques (NSGA-II et SPEA).

Le chapitre 5 est dédié à l'évaluation et à la comparaison entre les différentes méthodes décrites dans le chapitre précédent. La première comparaison a été effectué entre les trois versions de la recherche tabou, Nous avons constaté que l'approche tabou par epsilon-contrainte est avéré la plus performante que les deux algorithmes génétiques. La seconde comparaison est entre les deux algorithmes génétiques et l'approche epsilon-contrainte. Enfin, nous avons comparé les deux algorithmes mémétique dont la population initiale est générée par l'approche tabou. Nous avons conclu que notre approche epsilon-contrainte donnait les meilleurs résultats en un temps très satisfaisant.

Les premières perspectives de recherche que nous pouvons donner concernent les améliorations de nos algorithmes vus dans ce mémoire. Nous pouvons améliorer les deux algorithmes génétiques notamment au niveau de l'efficacité (le temps de calcul de résolution) et pour la recherche tabou, nous pouvons envisager d'autres techniques pour la détermination d'optima de Pareto. Enfin, dans ce mémoire, nous avons développé une hybridation combinant un algorithme génétique et une méthode de recherche tabou, cette dernière intervient au début de l'agorithme génétique pour construire la population initiale. Dans la littérature, nombreux chercheurs ont substitué la méthode tabou à l'opérateur de mutation. Donc, il est intéressant de développer cette hybridation pour notre problème.

BIBLIOGRAPHIE

[Back et al., 1997] T. Back, D.B. Fogel, Z. Michalewicz, and T. Baeck, *Handbook of Evolutionary Compu- tation*, Institute of Physics Publishing and Oxford University Press, 1997.

[Blazewicz et al., 1996] J. Blazewicz, K.H. Ecker, E. Pesch, G. Schmidt and J. Weglarz, *Scheduling Computer and Manufacturing Process*, ed Springer-Verlag, Berlin, Heidelberg, 1996.

[Blazewicz et al., 1996] J. Blazewicz, K. Ecker, E. Pesch, G. Schmidt and J. Weglarz, *Scheduling Computer and Manufacturing Processes*, Computers and Springer-Verlag, Berlin, Heidelberg, 1996.

[Bouibede, 2007] K. Bouibed, *Enumération en ordonnancement multicritère : application à un problème bicritère à machines parallèles*, Université François Rabelais Tours, 2007.

[Bouibede et al., 2012] Bouibede, K., Hettak, D., Boudhar, M. : *A tabu search for the enumeration of Pareto optima for uniform parallel machines scheduling with two criteria.* In Congrès des Mathématimaticiens Algériens (CMA2012), Annaba (Algérie), 2012.

[Brucker, 2004] P. Brucker, *Scheduling Algorithms.* Springer, Berlin. (2004)

[Dell'Amico et Martello, 1995] M. Dell'Amico and S. Martello : *Optimal Scheduling of Tasks on Identical Parallel Processors.* Journal on Computing, 7, 2, 181-200, 1995.

[Carlier, 1987] J. Carlier. *Scheduling jobs with release dates and tails on identical parallel machines to minimize the makespan.* European Journal of Operational Research, 29 :298–306, 1987.

[Carlier et Pinson, 1998] J. Carlier and E. Pinson. *Jackson's pseudo preemptive schedule for the P—ri, qi—Cmax.* Annals of Operations Research, 83 :41–58, 1998.

[Collette et Siarry, 2003] Y. Collette and P. Siarry. *Multiobjective optimization.* Springer-Verlag. 2003

[Colorni et al., 1991] A. Colorni, M. Dorigo, and V. Maniezzo. *Distributed optimization by ant colonies.* In Proceedings of European Conference on Artificial Life, pages 134-142, 1991.

[Colorni et al., 1992] A. Colorni, M. Dorigo, and V. Maniezzo. *An investigation of some proper- ties of an ant algorithm.* In Proceedings of the Parallel Problem Solving from Nature Conference, pages 509-520, 1992.

[Cook, 1971] S.-A. Cook. *The complexity of theorem proving procedures.* In Proceedings of the third Annual ACM Symposium on Theory of Computing. Association for Computing Machinery, pages 151–158, New York (USA), 1971

[De Jong, 1975] K.A. De Jong, *An analysis of the behaviour of a class of genetic adaptive systems*, PhD thesis, University of Michigan, 1975.

[Deb, 2002] K. Deb, *Multi-Objective Optimization using Evolutionary Algorithms*, Wiley, 2002.

[Deb et al., 2000] K. Deb, S. Agrawal, A. Pratap, and T. Meyarivan. *A fast and elitist multiobjective genetic algorithm for multi-objective optimization : NSGA-II. In Proceedings of the Parallel Problem Solving from Nature VI (PPSN-VI)*, pages 849-858, 2000.

[Ehrgott, 2005] M. Ehrgott. *Multicriteria Optimization.* Springer (Heidelberg). 2005.

[Ehrgott et Gandibleux, 2000] M. Ehrgott and X. Gandibleux : *A survey and Annotated Bibliography of Multiobjective Combinatorial Optimization.* OR Spektrum, 22. 425-460, 2000.

[Evans, 1984] G. Evans. *An overview of techniques for solving multiobjective mathematical programs.* volume 3O, pages 1268–1282. 1984

[$Fanjul - Peyro$ et Ruiz, 2010] L. Fanjul-Peyro and R. Ruiz : *Iterated greedy local search methods for unrelated parallel machine scheduling.* European Journal of Operational Reaserch, 207 , 55-69, 2010.

[Fogel, 2000] D. Fogel, *Evolutionary Computation : Toward a New Philosophy of Machine Intelligence (second edition)*, IEEE Press, 2000.

[Garey et Johnson, 1979] M. R. Garey and D. S. Johnson, *Computers and intractability : a guide to the theory of NP-completeness*, W. H. Freeman and Company, New York, 1979.

[Geoffrion, 1968] A. M. Geoffrion, *Proper efficiency and the theory of vector maximization*, Journal of Mathematical Analysis and Applications, 22 :618–630, 1968.

[Glover, 1986] F. Glover, *Future paths for integer programming and links to artificial intelligence*, Computers and Operations Research, 1986.

[Goldberg, 1989] D.E. Goldberg, *Genetic algorithms for search, optimization, and machine learning. Reading*, MA : Addison-Wesley, 1989.

[Graham et al., 1979] R.L. Graham, E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan, *Optimisation and approximation in deterministic sequencing and scheduling*, Annals of discrete Mathematics 287–326, 1979.

[Haimes et al., 1975] Y. Haimes, W. A. Hall, and H. T. Freedman, *Multiobjective Optimization in Water Resource Systems*, Elsevier Scientific Publishing, Amsterdam, 1975.

[Haimes et al., 1971] Y. Haimes, L. Ladson and D. Wismer, *On a bicriterion formulation of the problems of integrated system identification and system optimization*, IEEE Transactions on Systems, Man and Cybernetics, 1 :296–297, 1971.

[Haouari and Gharbi, 2003] M. Haouari and A. Gharbi. *An improved max-flowbased lower bound for minimizing maximum lateness on identical parallel machines.* Operations Research Letters, 31 :49–52. 2003.

[Holland, 1975] J.H. Holland, *Adaptation in natural and artificial systems*, PhD thesis, University of Michigan Press, 1975.

[Ishibuchi et Murata, 1996] H. Ishibuchi and T. Murata, *Multi-objective genetic local search algorithm*, In Pro- ceedings of IEEE International Conference on Evolutionary Computation (ICEC'96), pages 119-124. IEEE, 1996.

[Jaszkiewicz, 2004] A. Jaszkiewicz, *Evaluation of multiple objective metaheuristics*, In Gandibleux, X., Sevaux, M., Sorensen, K., T'kindt, V. (Eds) : Multiple Objective Metaheuristics, Lecture Notes in Economics and Mathematical Systems, 535 :1–26, 2004.

[Klein et Hannan, 1982] D. Klein and E. Hannan, , *An algorithm for the multiple objective integer linear programming problem*, European Journal of Operational Research, (9) :378–385, 1982.

[Koulamas et Kyparisis, 2000] C. Koulamas and G.J. Kyparisis, *Scheduling on uniform parallel machines to minimize maximum lateness*, Operations Research Letters 26, 175-179, 2000.

[Lancia, 2000] G. Lancia. *Scheduling jobs with release dates and tails on two unrelated parallel machines to minimize makespan.* volume 120, pages 277–288. 2000.

[Martello et al., 1997] S. Martello, F. Soumis and P. Toth : *Knapsack Problems : Algorithm and Computer Implementations.* John Wiley and Sons, Chichester, England. 1997.

[Mc Cormick et Pinedo, 1995] S. T. Mc Cormick, and M. L. Pinedo, *Scheduling n independant jobs on m uniform machines with both flowtime and makespan objectives : a parametric analysis*, ORSA Journal on Computing, 7(l) :63-77, 1995.

[Morrison et De Jong, 2001] W. Morrison and K.A De Jong, *Measurement of population diversity. In Artificial Evolution*, pages 31-41. Springer, 2001.

[Parks et Miller, 1998] G.T. Parks and I. Miller, *Selective breeding in a multiobjective genetic algorithm*, In Proceedings of the Fifth International Conference on Parallel Problem Solving from Nature (PPSN-V), pages 250-259. Springer, 1998.

[Pareto, 1896] V. Pareto, *Cours d'Economie Politique*, Rouge, Lausane, Swizerland, 1896.

[Roy, 1985] B. Roy, *Méthodologie multicritère d'aide à la décision (in french).* Economica, Paris, 1985.

[Ruiz et Stutzle, 2007] R. Ruiz and T. Stutzle, *A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem*,European Journal of Operational Reaserch, 177 (3), 2033-2049, 2007.

[Schwefel, 1984] H-P. Schwefel, *Numerical optimization of computer models*, Wiley, Chichester, 1984.

[Selman et al., 1994] Bart Selman, Henry A. Kautz, and Bram Cohen, *Noise strategies for improving local search*, In AAAI, volume 1, pages 337-343, 1994.

[Sen et al., 1988] T. Sen, M. Raiszadeh, and P. Dileepan, *A branch and bound approach to the bi-criterion scheduling problem involving total flowtime and range of lateness*,Management Science, 34(2) :254–260, 1988.

[Srinivas et Deb, 1995] N. Srinivas and K. Deb, *Multiobjective optimization using nondominated sorting in genetic algorithms, evolutionary computation*, The Massachusetts Institute of Technology, 2(3) :221–248, 1995.

[Stewart et White, 1991] B. Stewart and C. White, *Multiobjective a**, Journal of the ACM, 38(4) :775–814, 1991.

[Talbi, 2002] Talbi, E.-G., *A Taxonomy of Hybrid Metaheuristics*, Journal of Heuristics, 2002. 8 :pp. 541-564.

[Tamaki et al., 1994] H. Tamaki, M. Mori, M. Araki, Y. Mishima, and H. Ogai, *Multicriteria optimization by genetic algorithms : A case of scheduling in hot rolling process*, In Proceedings of APORS'94, pages 374-381. World Scientific Publishing, 1994.

[Tercinet, 2004] F. Tercinet. *Méthodes arborescentes pour la résolution des problèmes d'ordonnancement, conception d'un outil d'aide au développement*. PhD thesis. 2004.

[Tercinet et al., 2004] F. Tercinet, C. Lenté and E. Néron. *Mixed satisfiability tests for multiprocessor scheduling with release dates and deadlines*. volume 32. Oper. Res. Let. 2004.

[TKindt et Billaut, 2006] V. T'Kindt and J.-C. Billaut,, *Multicriteria Scheduling : Theory Models and Algorithms*, Springer-Verlag (Heidelberg), 2006.

[Tuzikov et al., 1998] A. Tuzikov, M. Makhaniok, and R. Manner, *Bicriterion scheduling of identical processing time jobs by uniform processors*, Computers and Operations Research, 25(l) :31-35, 1998.

[Ulungu et Teghem, 1994] E.L. Ulungu and J. Teghem : *Multi-objective combinatorial optimization : a survey*. Journal of Multi-Criteria Decision Analysis, 3. 83-104, 1994.

[Van de Velde, 2005] S.L. Van de Velde : *Duality-based algorithms for scheduling unrelated parallel machines*. ORSA J. on Computing, 3, 1, 1-21, 2005.

[White, 1982] D. White, *The set of efficient solutions for multiple-objectives shortest path problems*, Computers and Operations Research, 9 :101–107, 1982.

[Zitzler et al., 2000] E. Zitzler, K. Deb, and L. Thiele, *Comparison of multiobjective evolutionary algorithms : empirical results*, Evolutionary Computation Journal, 8(2) :125-148, 2000.

[Zitzler et Thiele, 1998] E. Ziztler and L. Thiele, *An evolutionary algorithm for multiobjective optimization : the strength pareto approach*, Technical report, Swiss Federal Institute of Technology, 1998.

[Zitzler et Thiele, 1999] E. Zitzler and L. Thiele. *Multiobjective evolutionary algorithms : a comparative case study and the strength Pareto approach*, IEEE Transactions on Evolutionary Computation, 3 :257-271, 1999.

Résumé

Les problèmes d'ordonnancement ont fait l'objet de nombreuses études dans la littérature. Mais la plupart d'entres elles portaient sur la résolution de problèmes à critère unique, malgré que la réalité est bien différente et que les domaines d'application font intervenir différents critères. Dans un contexte appliqué un problème d'ordonnancement est par nature multicritère ([Roy, 1985]). La résolution de ce type de problème consiste à trouver un ensemble de points correspondant aux meilleurs compromis possibles entres les différents critères, ces points sont appelés *optima de Pareto*. De nombreuses méthodes de résolution ont vu le jour pour résoudre des problèmes d'ordonnancement multicritères. Certaines de ces méthodes sont basées sur la détermination d'un seul optimum de Pareto. D'autre sont basées sur l'énumération de tous les optima de Pareto. Un panorama de telles méthodes est disponible dans [TKindt et Billaut, 2006]. Dans le cadre de notre étude nous avons proposé sept approches : trois versions de recherche tabou basées chacune sur une technique pour la détermination des solutions non-dominées : une marche aléatoire, une combinaison linéaire des critères et une approche epsilon-contrainte, deux algorithmes génétiques et deux algorithmes mémétiques combinant la recherche tabou et un algorithme génétique pour la résolution du problème d'ordonnancement bicritère à machines parallèles $Q/r_i, d_i/C_{max}, L_{max}$.

Mots-Clés

Ordonnancement multicritère, Optima de Pareto, recherche tabou, algorithmes génétiques, algorithmes mémétiques

Abstract

Numerous scheduling problems were the subject of many studies in the literature. But the majority of them are related to the solution of problems with a single criterion, although real life is quite different since problems often involves multiple criteria [Roy, 1985]. The solution of this kind of problem consists in finding a set of points corresponding to the best tradeoffs between different criteria. These points are called Pareto optima. Numerous solution methods are present in the literature to solve multicriteria scheduling problems. Some of these methods are based on the determination of only one Pareto optimum. Others are based on the enumeration of all the Pareto optima. A synthesis of these methods is presented in [TKindt et Billaut, 2006]. In this work, different meta heuristics were developed : tree versions of tabu search algorithms, two genetic algorithms and two hybrid algorithms to solve the uniform parallel machines scheduling problem $Q/r_i, d_i/C_{max}, L_{max}$.

Key-words

Multicriteria scheduling, Pareto optima, tabu search algorithms, genetic algorithms, hybrid algorithms .