optimiser j et k et $k4.446$

N° d'ORDRE :18/2016 – M/MT

REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE

MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE LA RECHERCHE SCIENTIFIQUE

UNIVERSITE DES SCIENCES ET DE LA TECHNOLOGIE HOUARI BOUMEDIENE

FACULTE DE MATHEMATIQUES

MEMOIRE

Présenté pour l'obtention du diplôme de MAGISTER
En : MATHEMATIQUES
Spécialité : Recherche Opérationnelle

Par : HADJ BENALI LINDA KHOULOUD

Sujet

Approches de Résolution du Problème d'Ordonnancement d'Atelier de Production

Soutenu publiquement, le 09/05/2016. Devant le jury composé de :

M. R. OUAFI	**Professeur**	**à l'USTHB**	**Président**
M. M. AIDER	**Professeur**	**à l'USTHB**	**Directeur de mémoire**
M. M. MOULAI	**Professeur**	**à l'USTHB**	**Examinateur**
Mme.C. ADICHE	**Maître de Conférences/B**	**à l'UMBB**	**Invitée**

Résumé

Nous nous intéressons au problème d'ordonnancement noté $1/r_i/\begin{cases}\sum_{i=1}^{n}\omega_i C_i \\ \sum_{i=1}^{n}\omega_i T_i\end{cases}$

Ce problème est décrit comme suit : Soit $tâche = \{tâche_1, \ldots, tâche_n\}$ un ensemble de n tâches, où chaque tâche est caractérisée par une date de disponibilité r_i, une durée opératoire p_i et un poids ω_i. Les tâches doivent toutes être exécutées sur une machine, qui ne peut effectuer plus d'une tâche en même temps. De plus, l'exécution d'une tâche ne peut se faire que sur un unique intervalle de temps, c'est-à-dire qu'on ne peut interrompre l'exécution d'une tâche (elles sont non préemptives). On note C_i la date de fin d'exécution de la $tâche_i$ et T_i le retard absolu dans un ordonnancement donné. Une solution du problème $1/r_i/\begin{cases}\sum_{i=1}^{n}\omega_i C_i \\ \sum_{i=1}^{n}\omega_i T_i\end{cases}$ est donc un ordre de passage des opérations sur la machine ; qui satisfait toutes les contraintes (disponibilité des tâches et disjonction de la machine), et qui minimise les deux quantités $\sum_{i=1}^{n}\omega_i C_i$ et $\sum_{i=1}^{n}\omega_i T_i$.

Mots clés : Ordonnancement à machine unique, date de disponibilité, la somme pondérée des dates de fin d'exécution, la somme pondérée des retards absolus.

Table des matières

Introduction générale

Les problèmes d'ordonnancement des ateliers de production sont largement étudiés depuis de nombreuses décennies. Plusieurs de ces problèmes ont été démontrés NP-difficiles au sens fort [42]. Autrement dit, on ne connaît pas d'algorithme polynomial qui peut les résoudre. L'utilité de l'ordonnancement dans le sous-système décisionnel du système de production et sa nature difficile ont créé un défi entre les chercheurs du domaine et les a motivé à proposer divers approches de résolution.

Les définitions d'ordonnancement proposées dans la littérature sont nombreuses. Nous en citons celles proposées par Gotha [29] : Ordonnancer un ensemble de tâches, c'est programmer leur exécution en leur allouant les ressources requises et en fixant leurs dates de début.

Un ordonnancement est composé de manière générale d'un ensemble de tâches soumises à certaines contraintes, et dont l'exécution de ces tâches nécessite des ressources. Notons qu'une tâche est par définition [29] un travail élémentaire nécessitant un certain nombre d'unités de temps et de ressources. La résolution du problème d'ordonnancement consiste à organiser ces tâches, c'est à dire à programmer dans le temps l'exécution de ces tâches, en leur allouant les ressources requises et en respectant les contraintes imposées, dans le but d'atteindre certains critères.

Nous avons quatre types de problème d'ordonnancement : Ordonnancement à une machine, ordonnancement Flow-Shop, ordonnancement Flow-Shop avec permutation et ordonnancement Flow-Shop sous les contraintes de blocage. Nous commençons par un état de l'art sur les travaux récents de ces quatre problèmes d'ordonnancement, ensuite nous résumant leur problématique et à la fin nous concluons par le choix d'un de ces quatre problèmes d'ordonnancement pour l'étudier dans notre projet de mémoire de magister.

Les travaux présentés dans l'état de l'art (chapitre 1) comportent généralement deux types de méthodes : des méthodes exactes basées sur la séparation et évaluation et des méthodes approchées basées généralement sur des métaheuristiques.

Des benchmarks de la littérature ou des instances générées aléatoirement ont été utilisées pour la validation de l'efficacité de ces méthodes.

En ce qui concerne la partie que nous souhaitons traiter dans le projet de mémoire de magister, nous nous envisageons d'étudier le problème à une seul machine. L'étude portera sur la modélisation et la proposition d'une heuristique ou métaheuristique de résolution.

Le problème d'ordonnancement à une seul machine qui nous intéresse consiste à minimiser les deux quantités $1/r_i/\begin{cases} \sum_{i=1}^{n} \omega_i C_i \\ \sum_{i=1}^{n} \omega_i T_i \end{cases}$.

Ce présent mémoire est organisé en quatre chapitres :

Le premier chapitre présente quelques généralités sur les problèmes d'ordonnancement, les types d'ateliers ainsi que les paramètres nécessaires pour caractériser les problèmes d'ordonnancement spécifiques sont aussi évoqués. Ensuite, nous présentons des notions de base sur la théorie de la complexité tout en présentant la complexité de certains problèmes d'ordonnancement.

Dans **le deuxième chapitre**, nous commençons par présenter quelques notions et définitions de base

de l'optimisation multiobjectif, pour ensuite montrer qu'il faut choisir une méthode d'optimisation pour résoudre le problème multiobjectif. Cette dernière étape est déjà délicate en optimisation mono−objectif, alors, on conçoit que l'optimisation multiobjectif soit d'une difficulté plus grande encore.

Le troisième chapitre est consacré dans un premier temps à la formulation mathématique du problème d'ordonnancement à une machine traité. Notons d'une part que la formulation mathématique proposée pour la résolution exacte du problème est donnée sous forme de programmes linéaires en variables mixtes (entières et bivalentes). D'autre part, l'utilisation d'un logiciel commercial pour la résolution exacte de ce programme mathématique.

Le quatrième chapitre, présente l'approche métaheuristique de résolution proposée. Nous décrivons les fondements théoriques de l'algorithme du recherche taboue utilisé pour la résolution du problème, et nous montrons comment l'algorithme est adapté pour le problème d'ordonnancement.

Dans **le cinquième chapitre**, une série d'expérimentations est menée autour de la méthode approchée que nous avons adoptée afin de juger sur l'efficacité de sa résolution en un temps raisonnable.

Pour la validation de notre approche recherche taboue, nous avons utilisé le logiciel ***lekin***, qui utilise principalement des règles de priorité et deux approches de résolution pour le problème d'ordonnancement mono-objectif $1/r_i/\sum_{i=1}^{n}\omega_i C_i$ et $1/r_i/\sum_{i=1}^{n}\omega_i T_i$.
Ensuite, nous avons utilisé quelque instances pour la résolution bi-objectif, en concaténant les deux fonctions objectifs vu précédemment.

A la fin de ce mémoire, nous concluons en établissant une synthèse des travaux réalisés dans ce mémoire et proposons un certain nombre de perspectives de recherche.

Chapitre 1

Généralités sur l'ordonnancement

1.1 Introduction

Dans la première section de ce chapitre, nous présentons quelques généralités sur les problèmes d'ordonnancement et donnons une brève description des différentes organisations d'ateliers qui caractérisent des problèmes d'ordonnancement spécifiques. Ensuite, nous présentons des notions de base sur la théorie de la complexité, notamment, ce qui concerne certains problèmr d'ordonnancement. Puis, nous présentons différentes approches de résolution et détaillerons celle que nous proposons pour la résolution du problème d'ordonnancement étudié.

1.2 Concepts de base en ordonnancement

L'ordonnancement est un champ d'investigation qui a suscité l'intérêt de beaucoup de chercheurs. De nombreux problèmes sont identifiés ainsi que des techniques de résolution sont développées pour les résoudre.

On retrouve dans la littérature plusieurs ouvrages consacrés à l'ordonnancement. Nous citons Baker [3], French [23], Carlier & Chrétienne [7], Mortan & Pintico [39], Chrétienne et al. [9], Chu & Proth [10], Esquirol & Lopez [22], Lopez & Roubelat [36], Pinedo [42] et T'kindt & Billaut [48]. Plusieurs définitions de l'ordonnancement y sont données, nous reprenons celle proposée dans [7] :

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

Un problème d'ordonnancement consiste à organiser dans le temps la réalisation d'un ensemble de tâches, compte tenu des contraintes temporelles et celles portant sur l'utilisation et la disponibilité des ressources requises par les tâches [36].

La théorie de l'ordonnancement traite de modèles mathématiques mais analyse également des situations réelles fort complexes ; aussi le développement de méthodes utiles ne peut-il être que le fruit de contacts entre la théorie et la pratique.

On va s'intéresser particulièrement aux problèmes d'ordonnancement d'ateliers, dans les quels, quatre notions fondamentales interviennent, ce sont les travaux (jobs, ou tâches), les ressources, les contraintes et les objectifs. Chacune de ces notions est définie dans ce qui suit.

1.2.1 Les tâches

Une tâche qu'on note i est, par définition, une entité élémentaire de travail localisée dans le temps par une date de début t_i ou une date de fin C_i, dont la réalisation nécessite l'utilisation de ressources pour une durée $p_i = C_i - t_i$ [22].

Elle est généralement caractérisée par trois paramètres [48] : sa durée opératoire p_i (processing time), sa date de disponibilité r_i (release time) et sa date d'échéance d_i (due date). Les tâches sont souvent liées entre elles par des relations d'antériorité. On dit alors qu'elles sont interdépendantes ; dans le cas contraire, elles sont dites indépendantes. Dans certains problèmes, les tâches peuvent être *préemptives*, ce qui signifie qu'elles peuvent être exécutées par morceaux par une ou plusieurs ressources ; dans le cas contraire elles sont dites *non préemptives*.

1.2.2 Les ressources

Une ressource est un moyen destiné à être utilisé pour la réalisation d'au moins une tâche, et est disponible en quantités limitées [22].

Différents types de ressources peuvent être distinguées :

• **Ressources renouvelables**

Une ressource est dite renouvelable, si après avoir été utilisée par une ou plusieurs tâches, elle est à nouveau disponible en même quantité comme : les hommes, les machines, l'espace, etc.

• **Ressources consommables**

Une ressource est consommable, lorsque elle devient non disponible après avoir été utilisée ; comme la matière première, l'énergie, etc [22].

• **Ressources disjonctives**

Une ressource est dite disjonctive (ou non partageable), si elle ne peut être affecté qu'à une seule tâche à la fois, (robot manipulateur)

• **Ressource cumulative**

Une ressource cumulative (partageable) peut être utilisée par plusieurs tâches simultanément (équipe d'ouvriers, poste de travail, ...) [22].

1.2.3 Les contraintes

Les contraintes expriment des restrictions sur les valeurs que peuvent prendre conjointement une ou plusieurs variables de décision [22]. Donc les contraintes représentent les limites imposées par l'environnement, tandis que l'objectif est le critère d'optimisation.

Plusieurs types de ces contraintes doivent être respectés. On distingue :

1. **Les contraintes de localisation temporelle** Sont issues des impératifs de gestion, et relatives aux dates limites des tâches ou du projet entier. On a surtout :
   – La date de disponibilité r_i (avant laquelle la tâche ne peut pas commencer).
   – La date d'échéance d_i(avant laquelle la tâche doit être achevée).
2. **Les contraintes d'enchaînement** Nous qualifions de contrainte d'enchaînement ou de succession, une contrainte qui lie le début ou la fin de deux activités par une relation linéaire [4]. Ce sont des contraintes imposées généralement par la cohérence technologique (les gammes opératoires dans le cas d'ateliers) qui décrivent des positionnements relatifs devant être respectés entre les tâches. D'autres contraintes plus spécifiques entre deux tâches ou plus, telles que : la synchronisation, la simultanéité, le recouvrement,... sont également imposées dans certains systèmes [22].
3. **Les contraintes de ressources** Une contrainte de ressources représente le fait que les activités utilisent une certaine quantité de ressources, tout au long de leur exécution [4]. Ces contraintes sont essentiellement soit [22] :

– Des contraintes d'utilisation des ressources qui expriment la nature, la quantité et les caractéristiques d'utilisation de ces ressources.
– Des contraintes de disponibilité des ressources qui déterminent les quantités des ressources disponibles au cours du temps.

On peut distinguer les contraintes suivant qu'elles sont strictes ou non. Les contraintes strictes sont des obligations à respecter. Alors que, les contraintes dites "relâchables" (appelées aussi préférences) peuvent éventuellement ne pas être satisfaites [35].

1.2.4 Les objectifs

Les facteurs les plus importants dans l'évaluation d'une solution d'un problème d'ordonnancement sont l'utilisation efficace des ressources, le délai global à respecter, la minimisation des encours ou le respect des dates échues [29]. Plusieurs objectifs sont considérés dans la littérature, nous en citons quelques-uns :

- La durée totale de l'ordonnancement (Makespan) $C_{max} = \max_{1 \preceq i \preceq n} C_i$, où C_i désigne la date de fin de la tâche i et n est le nombre de tâches.
- La moyenne des dates de fin des tâches $\overline{C}=\frac{1}{n}\sum_{1 \preceq i \preceq n} C_i$
- La moyenne pondérée des dates de fin des tâches $\overline{C^W} = \frac{1}{n}\sum_{1 \preceq i \preceq n} W_i C_i$ où W_i est un poids positif associé à la tâche i.
- Le plus grand retard algébrique $L_{max} = \max_{1 \preceq i \preceq n} L_i$,où $L_i=(C_i - d_i)$ représente le retard algébrique par rapport aux dates limites fixées.
- La moyenne des retards algébriques $\overline{L} = \frac{1}{n}\sum_{1 \preceq i \preceq n} L_i$
- La somme pondérée des retards algébriques $\overline{L^W} = \frac{1}{n}\sum_{1 \preceq i \preceq n} W_i L_i$.
- Le plus grand retard absolu $T_{max} = \max_{1 \preceq i \preceq n} T_i$,où $T_i=\max_{1 \preceq i \preceq n}(0,C_i - d_i)$ représente le retard absolu (tardiness) par rapport aux dates limites fixées.
- La moyenne des retards absolus $\overline{L} = \frac{1}{n}\sum_{1 \preceq i \preceq n} T_i$.
- La somme pondérée des retards absolus $\overline{T^W} = \frac{1}{n}\sum_{1 \preceq i \preceq n} W_i T_i$.
- Le plus grande avance absolue $E_{max} = \max_{1 \preceq i \preceq n} E_i$,où $E_i=\max_{1 \preceq i \preceq n}(0,d_i - C_i)$ représente l'avance absolue (earliness) par rapport aux dates limites fixées.
- La moyenne des avances absolues $\overline{E} = \frac{1}{n}\sum_{1 \preceq i \preceq n} E_i$
- La somme pondérée des avances absolues $\overline{E^W} = \frac{1}{n}\sum_{1 \preceq i \preceq n} W_i E_i$.
- Le plus grand temps de séjour dans l'atelier ou la durée de flot, noté $F_{max} = \max_{1 \preceq i \preceq n} F_i$, où $F_i=C_i-r_i$ représente le temps de séjour de i dans l'atelier.
- La moyenne des durées de flot $\overline{F} = \frac{1}{n}\sum_{1 \preceq i \preceq n} F_i$.
- La somme pondérée des durées de flot, $\overline{F^W} = \frac{1}{n}\sum_{1 \preceq i \preceq n} W_i F_i$.
- Le nombre de tâches en retard $\sum_{1 \preceq i \preceq n} U_i$, où U_i est un indicateur de retard tel que : $U_i=1$ si $L_i>0$ et $U_i=0$ sinon.
- La somme pondérée des tâches en retard, $\overline{U^W} = \frac{1}{n}\sum_{1 \preceq i \preceq n} W_i U_i$.

1.2.5 Les problèmes d'ordonnancement d'ateliers

La diversification des systèmes de production se caractérise par l'existence d'une multitude d'organisations au niveau d'atelier. Cette classification basée sur les différentes configurations des ressources (machines) permet de déterminer théoriquement et pratiquement des problèmes d'ordonnancement différents.Généralement, trois types de problèmes d'atelier sont distingués.

- **Machine unique** tous les travaux sont appelés à être exécutés sur une même machine.
- **Machines parallèles** Les machines parallèles exécutent les mêmes traitements et donc peuvent traiter n'importe quelle tâche. Plusieurs configurations de cette organisation peuvent être distinguées [4]. Selon

la vitesse de traitement on distingue :

- **Machines parallèles identiques** qui ont toutes la même vitesse.
- **Machine parallèles uniformes**, leurs vitesses sont différentes deux à deux mais restent indépendantes des tâches.
- **Machines parallèles non liées** (différentes), pour lesquelles les vitesses sont différentes deux à deux et dépendent des tâches exécutées.

– **Machines dédiées** plusieurs machines, chacune étant spécialisée pour l'exécution de certains travaux.

Ces organisations se caractérisent par l'existence de plusieurs ressources, sur lesquelles seront exécutées plusieurs tâches. Ce sont des organisations énormément répandues dans le domaine de la production industrielle, constituant ainsi des catégories différentes d'ateliers. Trois types ont une importance particulière dans les problèmes d'ordonnancement : le Job Shop, le Flow Shop et l'Open Shop.

– **Atelier à cheminement unique (Flow Shop)** : Dans ce cas, Les tâches sont décomposées en plusieurs opérations qui doivent être exécutées sur l'ensemble des machines selon un même ordre (cas d'une chaîne de fabrication).
– **Atelier à cheminement multiples (Job Shop)** : Le problème du job shop se différencie du flow shop par un élément essentiel : le cheminement des pièces n'est pas unidirectionnel. Les opérations des tâches doivent être exécutées sur un sous-ensemble de machines disposées en série, et peuvent avoir des routages différents.
– **Atelier à cheminement libre (Open Shop)** Dans le cas d'un atelier de type open shop, les contraintes de précédence sont relâchées. Autrement dit, les opérations nécessaires à la réalisation de chaque tâche peuvent être effectuées dans n'importe quel ordre (les gammes sont libres). Ce cas se présente lorsque chaque produit à fabriquer doit subir une séquence d'opérations, mais dans un ordre totalement libre.

1.2.6 Relations entre organisations

Les organisations présentées précédemment ne sont pas indépendantes. En effet, des relations de ressemblance, de réduction et de différentiation existent entre elles. L'étude des relations qui existent entre les différents problèmes d'ordonnancement revêt un grand intérêt, dans la mesure où cela permet d'appliquer des algorithmes de résolution connus pour certaines classes de problèmes à d'autres classes qui leurs sont réductibles [48].

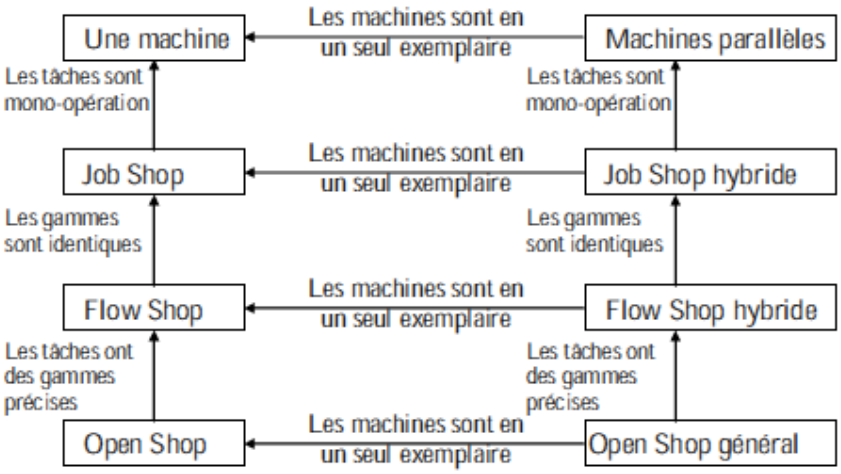


FIGURE 1.1 – Relations entre les différentes organisations [48]

1.2.7 Classification des problèmes d'ordonnancement

Etant donnée la diversité des problèmes d'ordonnancement, une classification formelle est nécessaire pour distinguer les différents problèmes. L'intérêt de telle classification réside, entre autre, dans [32] :
– Le besoin de recenser les problèmes qui sont de plus en plus divers, afin de mieux les identifier ;
– La détermination de leur complexité et de leurs outils de résolution s'ils existent.
Plusieurs classifications ont été proposées. Il ne s'agit pas de formalismes différents, mais d'une approche qui commence avec les travaux de Conway & al., reprise et améliorée chaque fois par d'autres chercheurs. La classification "standard" proposée par Graham & al. en 1979 et par Blazewicz et al. en 1983 , est couramment utilisée dans la littérature et elle comporte trois champs $\alpha|\beta|\gamma$.

1 Le premier champ α représente l'organisation des ressources et est spécifié par la concaténation de deux éléments α_1 et α_2.

- $\alpha_1 \in \{\emptyset, P, Q, R, F, J, O\}$ représente la configuration de machines utilisées :
  - $\alpha_1 = \emptyset$: problème à une seule machine.
  - $\alpha_1 \in \{P, Q, R\}$: problèmes à machines parallèles.
    - * $\alpha_1 = P$: les machines sont identiques et $p_{ij} = p_j$.
    - * $\alpha_1 = Q$: les machines sont uniformes et $p_{ij} = p_j/s_i$, où s_i est la vitesse de traitement de la machine M_i .
    - * $\alpha_1 = R$: les machines sont indépendantes et $p_{ij} = p_j/s_{ij}$, avec s_{ij} est la vitesse de traitement de la tâche J_j sur la machine M_i.
  - $\alpha_1 = F$: atelier à cheminement unique : « Flow shop ».
  - $\alpha_1 = J$: atelier à cheminements multiples : « Job shop ».
  - $\alpha_1 = O$: atelier à cheminements libres : « Open shop ».
- Le paramètre $\alpha_2 \in \emptyset, m$ permet de préciser le nombre de machines composant l'atelier.
  - $\alpha_2 = \emptyset$ le nombre de machines est variable.
  - $\alpha_2 = m$ le nombre de machines est constant et est égal à m.

2 Le champ β précise les caractéristiques des tâches et est généralement composé de sept sous-paramètres : $\beta_1, \beta_2, \beta_3, \beta_4, \beta_5, \beta_6, \beta_7$.Nous en citons quelque uns.

- $\beta_1 \in \{\emptyset, prmpt\}$ renseigne sur l'autorisation ou non de la préemption :
  - $\beta_1 = \emptyset$: la préemption n'est pas autorisée (lorsqu'une tâche commence son exécution, elle doit être terminée sans interruption).
  - $\beta_1 = prmpt$: la préemption est autorisée (l'exécution d'une opération peut être interrompue).
- β_2 : caractérise les ressources supplémentaires nécessaires à l'exécution d'un travail(outils, ressources de transport).
  - $\beta_2 = \emptyset$ indique qu'il n'y a pas de ressources complémentaires.
- $\beta_3 \in \{\emptyset, prec, tree, chain\}$ décrit un type de précédence entre travaux, i.e., le fait qu'un travail doit être exécuté avant un autre. La valeur $\emptyset$ indique que les travaux sont indépendants. Les valeurs prec, tree, chain indiquent l'existence, respectivement, d'une relation de précédence générale, d'une relation de précédence sous forme d'arbre et d'une relation de précédence sous forme de chaîne.
- $\beta_4 \in \{\emptyset, r_i\}$: spécifie si les táches sont toutes disponibles à l'état initial ou pas.
- $\beta_5 \in \{\emptyset, p_j = p, p_1 \preceq p \preceq p_2\}$ détaille les durées opératoires des différents travaux.
  - $\beta_5 = \emptyset$: les travaux ont des durées opératoires arbitraires.
  - $\beta_5 = p_j = p$: tous les travaux ont une durée opératoire égale à P.
  - $\beta_5 = p_1 \preceq p_j \preceq p_2$: les durées opératoires des travaux sont comprises entre p_1 et p_2.
- $\beta_6 \in \{d_j, \widetilde{d}_j\}$ indique les éventuelles dates d'échéance (ou dates au plus tard) des travaux.
  - $\beta_6 = \emptyset$: les travaux n'ont pas de date d'échéance.
  - $\beta_6 = d_j$: chaque travail à une date d'échéance de fin d'exécution sous peine de pénalisation.
  - $\beta_6 = \widetilde{d}_j$: chaque travail à une date d'échéance impérative (date limite) qu'il faut absolument respecter.
- $\beta_7 \in \{\emptyset, nwt\}$ indique si un temps d'attente entre les différentes opérations d'une tâche est autorisé. Ces contraintes sont très souvent introduites afin de mieux représenter les problèmes réels.
  - $\beta_7 = \emptyset$: un nombre illimité d'opérations peuvent être en attente devant une machine.
  - β_7=nwt (no-wait) on doit pouvoir imposer que toutes les opérations d'un travail soient exécutées sans temps d'attente.

3 Le champ γ correspond au critère à optimiser. (cf. chap1, § 1.2.4)

1.2.8 Complexité des problèmes d'ordonnancement

La théorie de la complexité a pour but d'analyser les coûts de résolution des problèmes d'optimisation combinatoire et permet d'établir une classification des problèmes en plusieurs niveaux de difficulté. Plusieurs ouvrages Blacewicz et al. en 1996 et Xuong en 1992 développent la théorie de la complexité. Cette théorie de la complexité repose sur la définition de classes de complexité qui permettent de classer les problèmes en fonction de la complexité des algorithmes qui existent pour les résoudre. Pour pouvoir exposer la notion de ces classes, il est nécessaire de donner quelques définitions de base.

Définition 1.1 *Un algorithme est* ***polynomial*** *si le nombre d'opérations élémentaires nécessaires pour résoudre un exemple de taille* n *est borné par un polynôme en* n. *Un algorithme est efficace si et seulement si il est polynomial [44].*

Définition 1.2 *Un problème de reconnaissance ou de décision est un problème dont la réponse est " vrai " ou " faux " [44].*

On peut associer à chaque problème d'optimisation un problème de décision, en définissant un seuil k pour la fonction objectif f, et on obtient le problème de décision " existe-il une solution réalisable S telle que $f(S) \preceq k$? " pour un problème de minimisation, et $f(S) \succeq k$? pour un problème de maximisation.

Définition 1.3 *Un algorithme non déterministe est un algorithme qui comporte l'instruction choix, celle-ci opérant sur un ensemble fini, choisit un élément de cet ensemble mais on ne spécifie pas à priori comment ce choix est effectué. De ce fait, les algorithmes non déterministes sont une construction abstraite et ne peuvent être mis en œuvre sur ordinateur [44].*

Définition 1.4 *La classe P regroupe tous les problèmes pouvant être résolus par un algorithme (déterministe) polynomial. Les problèmes de la classe P sont dits* ***faciles*** *[44].*

Définition 1.5 *La classe NP est la classe qui regroupe les problèmes de décision résolus en temps polynomial par un algorithme non déterministe [44].*

Parmi les problèmes de la classe NP, les problèmes NP-complets, ils sont équivalents entre eux et s'il existe un algorithme polynomial pour résoudre un seul de ces problèmes, alors il en existe un pour chaque problème de la classe NP. Pour décrire cette classe d'équivalence, on définit la notion de réduction polynomiale entre deux problèmes.

Définition 1.6 *Etant données deux problèmes de décision P1 et P2, on dit que P1 se réduit polynomialement à P2 s'il existe un algorithme pour P1 qui fait appel à un algorithme de résolution de P2, et qui est polynomial lorsque la résolution de P2 est comptabilisé comme une opération élémentaire [44].*

Définition 1.7 *Un problème de décision est NP-complet si tout problème de la classe NP se réduit polynomialment à lui [44].*

Définition 1.8 *Un problème d'optimisation dont le problème de décision associé est NP complet est dit* ***NP-difficile*** *[44].*

Montrer qu'un problème de décision D est NP-complet consiste à :

1. Montrer que D est dans la classe NP,
2. Choisir D' un problème NP-complet connu,
3. Construire une transformation f de D' à D,
4. Montrer que f est une réduction polynomiale.

Cette preuve révèle une très grande importance puisque une fois la démonstration faite, on peut dire que l'existence d'un algorithme polynomial pour la résolution du problème considéré est peu probable, et par la suite les efforts seront dirigés pour la recherche d'une méthode de résolution approximative.

Problème NP-complets connus

Le problème P=NP est un problème ouvert posé en 1970. On a trivialement P⊂NP car un algorithme déterministe est un algorithme non déterministe particulier. Mais la réciproque NP⊂P, que l'on résume généralement à P=NP, est l'un des problèmes ouverts les plus fondamentaux et intéressants en mathématiques et informatique théorique. En effet, s'il s'avérait que P=NP, alors on pourrait résoudre tous les problèmes NP-complets en un temps polynomial.

1.2.9 Hiérarchie de complexité entre les problèmes d'ordonnancement

Généralement, un algorithme défini pour un problème d'ordonnancement spécifique peut être appliqué à un autre problème d'ordonnancement [42].

L'intérêt d'étudier les différentes relations entre les problèmes d'ordonnancement est d'appliquer des algorithmes de résolution de certains problèmes pour en résoudre d'autres qui leurs sont réductibles. Une représentation de ces relations peut avoir plusieurs configurations, selon le type d machines, les contraintes de précédence entre les tâches, les contraintes sur les durées de traitement ou bien selon les critères d'optimalité. Les figures ci-dessous représentent ces différentes configurations. Le sens de la flèche indique une réduction polynomiale.

Réduction entre les durées de traitement Les durées opératoires des tâches influent sur la complexité des problèmes, une simple contrainte sur les durées opératoires peut rendre le problème NP-difficile ou facile. Selon la figure 1.2

Réduction entre les machines D'après le graphe, le cas le plus simple est celui où il n'y a qu'une seule machine, c'est un cas particulier de plusieurs machines parallèles identiques ou en série. La difficulté augmente si le nombre de machines est non borné et si on considère des machines uniforme ou non liées (différentes).

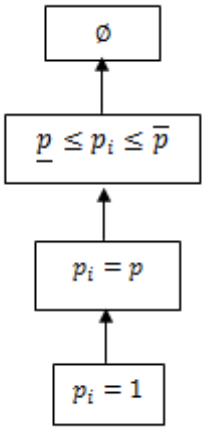
Réduction entre les contraintes de précédence Selon la figure, il est nettement claire que le cas où il n'y a pas de contraintes de précédence est plus simple que celui où les contraintes sont considérées. La complexité varie selon leur nature (graphe de précédence des tâches).

Relations entre les critères d'optimisation Les critères que doit satisfaire un ordonnancement sont variés. L'existence de plusieurs relations d'équivalence, de similitude et de réduction entre ces critères permet de déduire certains des autres. En effet, une solution optimale pour un critère donné peut s'avérer optimale pour un autre.

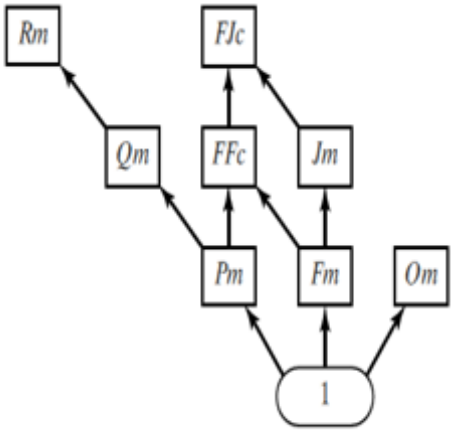
Une présentation plus large et complète concernant la hiérarchie existant entre les différents problèmes d'ordonnancement peut être trouvée dans Pinedo en 2008 et Blacewicz & al. en 1996.

1.3 Complexité des problèmes d'ordonnancement

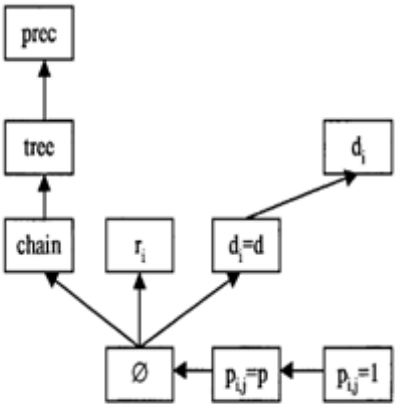
Les problèmes d'ordonnancement des ateliers de production ont été largement étudiés au cours de ces dix dernières années. Plusieurs de ces problèmes ont été démontrés NP-difficile au sens fort [42]. Autrement dit, il n'existe pas d'algorithme permettant de les résoudre de manière exacte en un temps polynomial. L'utilité de l'ordonnancement dans le sous-système décisionnel du système de production et sa nature



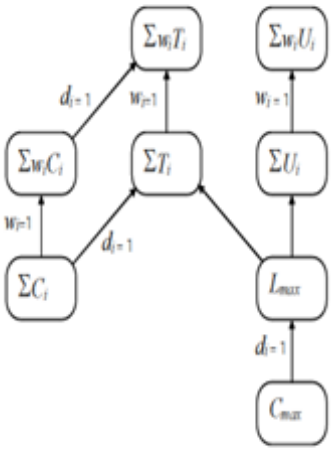
Réduction entre les durées de traitement



Réduction entre les machines *[Pinedo]*



Réduction entre les contraintes de précédence



Relations entre les critères d'optimisation

FIGURE 1.2 – Hiérarchies de complexité des problèmes d'ordonnancement déterministe

difficile ont créé un défi entre les chercheurs du domaine et les a motivé à proposer divers approches de résolution.

Les problèmes d'ordonnancement à une machine. Parmi ces travaux, il y a ceux qui ont optimisé la moyenne pondérée des dates de fin d'exécution de tâches. Nous citons à titre d'exemple les travaux de Nessah & Kacem [40] qui ont présenté une méthode arborescente. Dans leur papier, ils proposent deux nouvelles bornes inférieures dont elles peuvent respectivement être calculées en $o(n^2)$ et $o(nlogn)$, où n est le nombre de jobs. Aussi, ils présentent une heuristique efficace en utilisant les bornes proposées. L'heuristique proposée, les bornes inférieures et les propriétés de dominance sont intégrées dans un algorithme Branch and Bound qui a été testé sur des instances de grande taille. D'autres travaux récents intègrent des tâches de maintenance à ordonnancer avec les tâches d'ordonnancement, comme les travaux de Ju−Yong Lee and Yeong−Dae Kim en 2012 et les travaux de Mahin Esmaeili en 2012. Dans les travaux de Ju−Yong Lee and Yeong−Dae Kim en 2012, il y a eu l'étude de la maintenance périodique avec l'objectif de minimiser le nombre de tâches en retard. Les auteurs proposent une heuristique à deux phases dans lequel la solution initiale est obtenue d'abord par une méthode modifiée à partir de l'algorithme de Moore pour le problème sans maintenance, puis la solution est améliorée dans la deuxième phase. La performance de l'heuristique proposée est évaluée par des expérimentations sur des problèmes générés aléatoirement et les résultats obtenus montrent que leur heuristique donne des solutions proches de celles obtenues à partir d'un solveur commercial de programmation entière.

Les premiers résultats sur l'ordonnancement d'atelier Flow-Shop sont apparus dans les années cinquante. Johnson (1954) a proposé un algorithme polynomial pour l'ordonnancement Flow-Shop à deux machines sous le critère de minimisation du makespan F2| |Cmax selon la notation de Blazewiczet, al. en 1983. Dans ce même travail, il a démontré que pour les problèmes d'ordonnancement Flow-Shop à trois machines sous le même critère makespan ($F3||Cmax$), si les durées opératoires sur la deuxième machine sont uniformément inférieures à ceux de la première ou de la troisième machine, le problème peut être aussi résolu en un temps polynomial. Par contre, si cette condition n'est pas vérifiée, Garey & al.en 1976 [26] ont prouvé que ce dernier est NP-Difficile au sens fort. Dans le cas général, plusieurs méthodes (exactes ou approchées) ont été proposées pour la résolution du problème d'ordonnancement Flow-Shop. Parmi les méthodes exactes, on trouve ceux basées sur la Séparation et Evaluation (Branch and Bound). Où plusieurs travaux l'ont utilisé pour résoudre le Flow-Shop à m machines $F||Cmax$. Parmi ces travaux, nous citons : les travaux de Pottsen 1980 qui a proposé une procédure Branch and Bound avec laquelle il est possible de résoudre des instances jusqu'à 15 jobs et 4 machines. Carlier et Rebaï en 1996 ont également proposé une procédure Branch and Bound pouvant résoudre des instances jusqu'à 50 jobs et 10 machines. Plusieurs approches ont été également proposées pour le problème $F||Cmax$. Nous trouvons dans la littérature des travaux basés sur des méthodes constructives et autres basées sur des méta-heuristiques. A titre d'exemple, les algorithmes génétiques de Iyer et Saxena en 2004, Siarry et Michalewicz en 2007 et Zobolas & al. en 2009 ont des résultats qui pouvant résoudre des problèmes de grande taille (jusqu'à 200 jobs et 20 machines). Parmi les dérivées du problème d'ordonnancement de type Flow-Shop, nous détaillons dans la section suivante quelques travaux traitant le Flow-Shop avec permutation et le Flow-Shop avec blocage.

Concernant le flow shop avec permutation, divers travaux récents, Zobolas, &. al en 2009 ont proposé une méta-heuristique hybride pour la minimisation du makespan du problème d'ordonnancement Flow-shop. L'approche proposée selon les auteurs est robuste, simplement structurée et est formée de trois

composants : une procédure de génération de la population initiale qui est basée sur une heuristique constructive, un algorithme génétique pour l'évolution de la solution et une méthode de recherche par voisinage pour l'amélioration de la population. Les résultats obtenus, suite à des expérimentations effectuées sur des benchmarks, que la méthode proposée résout des instances de grande taille en un temps raisonnable. Grabowski and Wodecki en 2004, ont proposé un algorithme basé sur la recherche taboue pour le Flow-Shop avec permutation et ont testé des instances supérieur à 500 travaux et 20 machines par une comparaison avec les meilleurs résultats de la littérature. Cheng et, al. en 2001, ont étudié le problème de flow shop avec permutation en considérant trois machines et optimisant le critère makespan sous les contraintes de disponibilité des travaux. Ils ont proposé une méthode par séparation et évaluation pour sa résolution et aussi ont utilisé une procédure de décomposition pour réduire la taille des problèmes avec des dates de disponibilité assez large.

1.4 Méthodes de résolution

Les problèmes d'optimisation combinatoire notamment les problèmes d'ordonnancement sont de complexités différentes. Il n'existe pas de méthodes universelles permettant de résoudre efficacement tous les cas.

Vu la grande importance des problèmes d'ordonnancement, de nombreuses méthodes qui peuvent être classées en deux grandes catégories : les méthodes exactes et les méthodes approchées [24].

Dans cette partie, nous exposons quelques méthodes les plus connues classées en trois catégories principales : méthodes exactes, méthodes approchées constructives et approchées amélioratrices. Notant que par la suite, nous détaillons les deux méthodes, exacte et approchée, que nous proposons.

1.4.1 Méthodes exactes

Les méthodes de résolution exactes sont nombreuses et se caractérisent par le fait qu'elles permettent d'obtenir une ou plusieurs solutions dont l'optimalité est garantie.

Parmi ces méthodes, on trouve les techniques de séparation et évaluation progressive(SEP) et la programmation dynamique.

La programmation dynamique

Introduite par Bellman dans les années 50, elle consiste à décomposer un problème de dimension n en n sous-problèmes de dimension1. Le système est alors constitué de n étapes que l'on résout séquentiellement, le passage d'une étape à une autre se fait à partir des lois d'évolution du système et d'une décision. Le principe d'optimalité est basé sur l'existence d'une équation récursive permettant de décrire la valeur optimale du critère à une étape en fonction de sa valeur à l'étape précédente [18].

La modélisation mathématique d'un problème d'ordonnancement ne se limite pas à mettre en évidence l'objectif et l'ensemble de contraintes caractérisant le problème, mais ce qui est important est de le résoudre efficacement tous les cas.

Il est évident que les problèmes d'ordonnancement peuvent être modélisés par des programmes linéaires

en variables mixtes (entières et binaires). Ce type de modèles est donc très difficile à résoudre et le temps de calcul nécessaire d'une telle méthode augmente en général exponentiellement avec la taille du problème à résoudre (dans le pire des cas).

Dans la littérature, deux types de méthodes de résolution sont généralement utilisées pour résoudre les programmes linéaires en variables entières (ou mixtes) : les méthodes utilisant les coupes comme la coupe de Gomory en 1958 et les méthodes utilisant les arbres de recherche comme la séparation et évaluation de Dakin en 1965.

Méthode de séparation et évaluation

Cette méthode consiste à développer une arborescence, dont la racine représente l'ensemble de toutes les solutions réalisables, en tenant compte d'un principe de séparation, un principe d'évaluation et une stratégie de développement. Le principe de séparation consiste à partager l'ensemble des solutions réalisables en sous ensembles en fonction d'un certain critère. L'évaluation permet de calculer (dans le cas d'un problème de minimisation) une borne inférieure de la solution obtenue après la séparation. Une borne supérieure est calculée au préalable, elle est utilisée pour éviter l'exploration des nœuds dont la valeur de la borne inférieure est supérieure à la valeur de la borne supérieure. La procédure de séparation est appliquée selon plusieurs stratégies, la première en profondeur consiste à descendre dans les branches jusqu'à ce qu'on trouve un sommet qu'on peut éliminer, ensuite on remonte pour descendre dans une autre branche. La seconde stratégie le meilleur d'abord, dans ce cas on commence par séparer les sommets qui ont l'évaluation la plus petite (pour un problème de minimisation) en espérant trouver la solution optimale dans ces sommets. Enfin, on peut envisager une exploration en largeur, cette stratégie est peu utilisée pour des questions d'efficacité et d'encombrement de la mémoire [8].

1.4.2 Méthodes approchées

Une méthodes approchée ou heuristique est un algorithme qui a pour but de trouver une solution réalisable, tenant compte de a fonction économique, mais sans garantie d'optimalité, contrairement aux méthodes exactes qui trouvent toujours l'optimum si on leur laisse le temps qu'il faut. Il existe un très grand nombre d'heuristiques selon le problème à traiter. On distingue les méthodes constructives, les méthodes amélioratrices et les méthodes basées sur l'intelligence artificielle.

Méthodes constructives

Ce sont des méthodes itératives ou à chaque itération, une solution est complétée. Ces méthodes sont généralement des algorithmes gloutons car elles considèrent les éléments dans un certain ordre sans remettre en question un choix une fois qu'il est fait. Pour les problèmes d'ordonnancement on peut citer les algorithmes de liste qui consistent à établir une liste, à partir d'un critère de priorité, puis construire l'ordonnancement à partir de cette liste Ces méthodes constructives se distinguent par leur rapidité et leur grande simplicité. Mais leur défaut réside malheureusement dans la qualité des solutions obtenues.

Les méthodes amélioratrices

Ces méthodes sont initialisées par une solution réalisable obtenue par une méthode gloutonne ou générée aléatoirement, et recherchent à chaque itération une amélioration de la solution courante par des modifications locales jusqu'a ce qu'un critère d'arrêt soit satisfait et on ne peut plus améliorer la solution courante. Parmi les méthodes amélioratrices qui ont prouvé leur efficacité dans la résolution de problème d'optimisation combinatoire, nous citons le recuit simulé, la recherche taboue, et la colonie des fourmis.

Méthodes basées sur l'intelligence artificielle

Cette catégorie regroupe des méthodes qui utilisent des techniques de représentation des connaissances et de résolution de problèmes issus de l'intelligence artificielle. Cette catégorie de méthodes est vaste et peut être scindée en plusieurs sous-classes. Parmi ces sous-classes, nous citons : les systèmes à base de connaissances dotés de capacité d'apprentissage et les algorithmes génétiques [29].Pour avoir plus de détails ainsi que des exemples d'application, on peut se référé à l'ouvrage de [8], celui de Lacomme et al.

La présentation générale menée au cours de ce chapitre sur les Méta-heuristiques, étaient et restent l'objet d'une recherche intense pour la résolution de divers problèmes NP-difficiles. Elles ont révélé leur grande efficacité de fournir des solutions approchées de bonne qualité pour un grand nombre de problèmes. Dans le chapitre suivant, on va définir quelques concepts importants de l'optimisation multi-objectif utiles.

Chapitre 2

Optimisation multiobjectif

2.1 Introduction

Nous rappelons dans la première partie de ce chapitre quelques notions et définitions de base de l'optimisation multiobjectif. Nous donnerons aussi une classification des méthodes d'optimisation multiobjectif. Puis dans la partie qui suit, nous présenterons quelques une de ces méthodes. On finira par définir quelques métriques permettant l'évaluation de ces méthodes.

Le problème à résoudre peut fréquemment être exprimé sous la forme générale d'un problème d'optimisation. Un problème d'optimisation mono−objectif se définit comme la recherche du minimum ou du maximum (de l'optimum donc) d'une fonction donnée. Mathématiquement parlant, un problème d'optimisation se présentera sous la forme suivante :

$$\begin{cases} \text{minimiser } f(\overrightarrow{x}) & \text{(fonction à optimiser)} \\ \text{avec } g(\overrightarrow{x}) \preceq 0 & \text{(m contraintes d'inégalité)} \\ \text{et } h(\overrightarrow{x}) = 0 & \text{(p contraintes d'égalité)} \end{cases}$$

On a $\overrightarrow{x} \in \Re^n$. Ici, $g(\overrightarrow{x})$ et $h(\overrightarrow{x})$ représentent respectivement m contraintes d'inégalité et p contraintes d'égalité [12].

2.2 Notions de base

Fonction objectif C'est le nom donné à la fonction f (on l'appelle encore fonction de coût ou critère d'optimisation). C'est cette fonction que l'algorithme d'optimisation va devoir "optimiser" (trouver un optimum). Sauf mention explicite contraire, on supposera dans la suite que la fonction objectif est à minimiser.

Variables de décision Elles sont regroupées dans le vecteur $\overrightarrow{x}$. C'est en faisant varier ce vecteur que l'on recherche un optimum de la fonction f .

Minimum global Un "point" $\overrightarrow{x^*}$ est un minimum global de la fonction f si on a : $f(\overrightarrow{x^*}) < f(\overrightarrow{x})$ quel que soit $\overrightarrow{x}$ tel que $\overrightarrow{x^*} \neq \overrightarrow{x}$

Minimum local fort Un "point" $\overrightarrow{x^*}$ est un minimum local fort de la fonction f si on a : $f(\overrightarrow{x^*}) < f(\overrightarrow{x}$) quel que soit $\overrightarrow{x} \in V(\overrightarrow{x^*})$ et $\overrightarrow{x^*} \neq \overrightarrow{x}$, où $V(\overrightarrow{x^*})$ définit un "voisinage" de $\overrightarrow{x^*}$.

Minimum local faible Un "point" $\overrightarrow{x^*}$ est un Minimum local faible de la fonction f si on a : $f(\overrightarrow{x^*}) \leq f(\overrightarrow{x}$) quel que soit $\overrightarrow{x} \in V(\overrightarrow{x^*})$ et $\overrightarrow{x^*} \neq \overrightarrow{x}$, où $V(\overrightarrow{x^*})$ définit un "voisinage" de $\overrightarrow{x^*}$.

2.3 Optimisation multiobjectif

La formulation précédente était relative à un problème dans lequel on recherchait un optimum pour une seule fonction objectif f (dans l'expression précédente). Cependant, lorsque l'on modélise un problème, on cherche souvent à satisfaire plusieurs objectifs. Par exemple, on veut un système performant et on veut aussi que ce système consomme peu. Dans ce cas, on parle de problème d'optimisation multiobjectif (ou problème d'optimisation multicritère). Celui-ci s'écrit de la manière suivante :

$$(P)\begin{cases} \text{minimiser } f_i(\overrightarrow{x}), i = 1, \ldots, k \\ \text{avec } g(\overrightarrow{x}) \preceq 0 \\ \text{et } h(\overrightarrow{x}) = 0 \\ \text{où } \overrightarrow{x} \in \Re^n,\ g(\overrightarrow{x}) \in \Re^m, h(\overrightarrow{x}) \in \Re^p. \end{cases}$$

On appellera ce problème le **problème P**. Comme on peut le voir ici, on n'a plus un seul objectif à atteindre, mais k [12].

2.3.1 Multiplicité des solutions

Lorsque l'on cherche à obtenir une solution optimale à un problème d'optimisation multiobjectif donné, on s'attend souvent à trouver une solution et une seule. En fait, on rencontre rarement ce cas de figure. La plupart du temps, on trouve une multitude de solutions, du fait que certains des objectifs sont contradictoires.

Donc, quand on résoudra un problème d'optimisation multiobjectif, on obtiendra une grande quantité de solutions.

Ces solutions, comme on peut s'en douter, ne seront pas optimales, au sens où elles ne minimiseront pas tous les objectifs du problème.

Un concept intéressant, qui nous permettra de définir les solutions obtenues, est le compromis. En effet, les solutions que l'on obtient lorsque l'on a résolu le problème sont des solutions de compromis. Elles minimisent un certain nombre d'objectifs tout en dégradant les performances sur d'autres objectifs.

2.3.2 Dominance

Introduction et définitions

Lorsque nous avons résolu notre problème d'optimisation multiobjectif, nous avons obtenu une multitude de solutions. Seul un nombre restreint de ces solutions va nous intéresser. Pour qu'une solution soit intéressante, il faut qu'il existe une relation de dominance entre la solution considérée et les autres solutions, dans le sens suivant [12] :

Définition 2.1 *La relation de dominance On dit que le vecteur $\overrightarrow{x_1}$ domine le vecteur $\overrightarrow{x_2}$ si :*
- *$\overrightarrow{x_1}$ est au moins aussi bon que $\overrightarrow{x_2}$ pour tous les objectifs, et*
- *$\overrightarrow{x_1}$ est strictement meilleur que $\overrightarrow{x_2}$ pour au moins un objectif*

*Les solutions qui dominent les autres mais ne se dominent pas entre elles sont appelées **solutions optimales au sens de Pareto** (ou solutions **non dominées**). On définit comme suit l'optimalité locale et l'optimalité globale au sens de Pareto.*

Définition 2.2 *Optimalité locale au sens de Pareto Un vecteur $\overrightarrow{x} \in \Re^n$ est optimal localement au sens de Pareto s'il existe un réel $\delta > 0$ tel qu'il n'y ait pas de vecteur $\overrightarrow{x'}$ qui domine le vecteur $\overrightarrow{x}$ avec $\overrightarrow{x'} \in \Re^n \bigcap B(\overrightarrow{x}, \delta)$représente une boule de centre $\overrightarrow{x}$ et de rayon δ . Un vecteur $\overrightarrow{x}$ est donc optimal localement au sens de Pareto s'il est optimal au sens de Pareto sur une restriction de l'ensemble $\Re^n$ [12].*

Définition 2.3 *Optimalité globale au sens de Pareto Un vecteur $\overrightarrow{x}$ est optimal globalement au sens de Pareto (ou optimal au sens de Pareto) s'il n'existe pas de vecteur $\overrightarrow{x'}$ tel que $\overrightarrow{x'}$ domine le vecteur $\overrightarrow{x}$.*

La différence entre cette définition et celle de l'optimalité locale tient dans le fait que l'on ne considère plus une restriction de l'ensemble $\Re^n$.

2.3.3 Cônes polaires

Définition 2.4 *Un cône négatif est défini dans R^k de la manière suivante :*

$$C^- = \{\overrightarrow{x} \setminus f(\overrightarrow{x}) \in \Re^k et f(\overrightarrow{x} \leq 0\}$$

Théorème 2.1 ***Le théorème du contact*** *Un vecteur $\overrightarrow{x}$ est optimal au sens de Pareto pour un problème d'optimisation multiobjectif donné si :*

$$(C^- + \overrightarrow{x}) \bigcap S = \overrightarrow{x}$$

où S désigne l'espace des solutions réalisables.

2.3.4 Relations dérivées de la dominance

La relation de dominance n'offre pas de degrés de liberté dans sa définition. Par exemple, il n'est pas possible d'inclure dans la définition de la relation de dominance une préférence d'un objectif par rapport à un autre. C'est pour contrecarrer ce manque de flexibilité des relations dérivées de la relation de dominance ont été développées. Les solutions permettent de trouver ces relations dérivées de la dominance sont toutes optimales au sens de Pareto. La grande différence que l'on rencontre avec ces relations est que l'ensemble solutions que l'on obtient avec ces relations est un sous-ensemble de l'ensemble des solutions obtenues avec la relation de dominance de Pareto.

2.3.5 Dominance au sens de Geoffrion

Une dernière forme de dominance importante dans le monde de l'optimisation multiobjectif est la dominance au sens de Geoffrion (voir [20] et [38]). Les solutions optimales obtenues par ce type de dominance sont appelées les solutions Pareto optimales propres.

Définition 2.5 *La dominance au sens de Geoffrion Une solution $\overrightarrow{x^*} \in F$ est appelée solution Pareto optimale propre si :*

- *elle est Pareto optimale*
- *il existe un nombre $M > 0$ tel que $\forall i$ et $\forall \overrightarrow{x} \in S$ vérifiant $f_i(\overrightarrow{x}) < f_i(\overrightarrow{x^*})$, il existe un index j tel que $f_j(\overrightarrow{x^*}) < f_j(\overrightarrow{x})$ et :*

$$\frac{f_i(\overrightarrow{x^*}) - f_i(\overrightarrow{x})}{f_j(\overrightarrow{x}) - f_j(\overrightarrow{x^*})} \leq M$$

Cette relation n'est quasiment jamais utilisée telle quelle. En général, on utilise plutôt un résultat qui découle de cette définition. En effet, l'interprétation de ce théorème faite dans [20] est la suivante :

"Les solutions Pareto optimales propres ont des compromis bornés suivant leurs objectifs".

Un théorème relatif à la méthode de pondération des fonctions objectif utilisant ce résultat est le suivant :

Théorème 2.2 *Soit la méthode d'agrégation des fonctions objectif suivantes :*

$$f_{eq}(\overrightarrow{x}) = \sum_{i=1}^{k} \omega_i . f_i(\overrightarrow{x})$$

Supposons $\forall i = 1, \ldots, k,\ \omega_i > 0$ *avec* $\sum_{i=1}^{k} \omega_i = 1$ *[12].*

Si $\overrightarrow{x^*}$ *est une solution optimale obtenue en utilisant la méthode d'agrégation ci-dessus, alors cette solution est aussi Pareto optimale propre.*

La méthode de pondération des fonctions objectif, avec des poids qui respectent les relations ci-dessus, permet d'obtenir des solutions avec des compromis bornés.

2.3.6 Surface de compromis

Le petit nombre de solutions de rang 1 que l'on a sélectionnées en utilisant la règle de classement basée sur la définition de la dominance forme ce que l'on appelle **la surface de compromis** (ou **front de Pareto**). Imaginons que nous ayons un problème à deux objectifs (minimiser f_1 et minimiser f_2 sous les contraintes $g(\overrightarrow{x}) \leq 0$ et $h(\overrightarrow{x}) = 0$) :

- On appelle S l'ensemble des valeurs du couple $(f_1(\overrightarrow{x}), f_2(\overrightarrow{x}))$ quand $\overrightarrow{x}$ respecte les contraintes $g(\overrightarrow{x})$ et $h(\overrightarrow{x})$
- On appelle P la surface de compromis

On représente S et P comme suit :

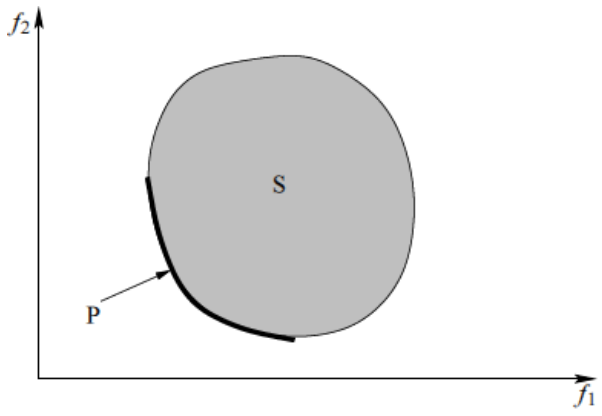


FIGURE 2.1 – Représentation de la surface de compromis [12].

Ces formes de surfaces de compromis sont typiques d'un problème d'optimisation multiobjectif sur un ensemble de solutions convexe. C'est ce type d'ensemble que l'on rencontre la plupart du temps. On observe deux points caractéristiques associés à une surface de compromis :

Point idéal Les coordonnées de ce point sont obtenues en optimisant chaque fonction objectif séparément [24].

Point "nadir" Les coordonnées de ce point correspondent aux pires valeurs obtenues par chaque fonction objectif lorsque l'on restreint l'espace des solutions à la surface de compromis. Le point idéal est utilisé dans beaucoup de méthodes d'optimisation comme point de référence. Le point nadir, lui, sert à restreindre l'espace de recherche ; il est utilisé dans certaines méthodes d'optimisation interactives [12].

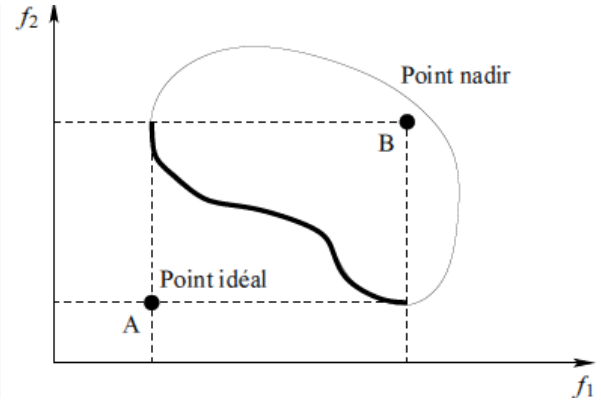


FIGURE 2.2 – Représentation du point idéal et du point nadir

Remarque Comme nous le verrons plus loin, certaines méthodes d'optimisation multiobjectif nécessitent de respecter certaines hypothèses. Le plus souvent, la méthode réclame de travailler sur un espace S des valeurs de $\overrightarrow{x}$ qui soit convexe.

Définition 2.6 *La convexité Un ensemble S est convexe si, étant donnés deux points distincts quelconques de cet ensemble, le segment qui relie ces deux points est contenu dans l'ensemble S.*

2.4 Méthodes de résolution

Les méthodes d'optimisation multicritères s'intéressent aux problèmes essayant d'optimiser simultanément plusieurs objectifs. Elles peuvent être classées selon le schéma suivant [47] :

Selon le mode de coopération entre le solveur et le décideur, elles peuvent être rangées en trois familles :

2.4.1 Méthodes à préférence à priori

L'utilisateur définit le compromis qu'il désire réaliser avant de lancer la méthode d'optimisation. On retrouve dans cette famille, toutes les méthodes agrégatives.

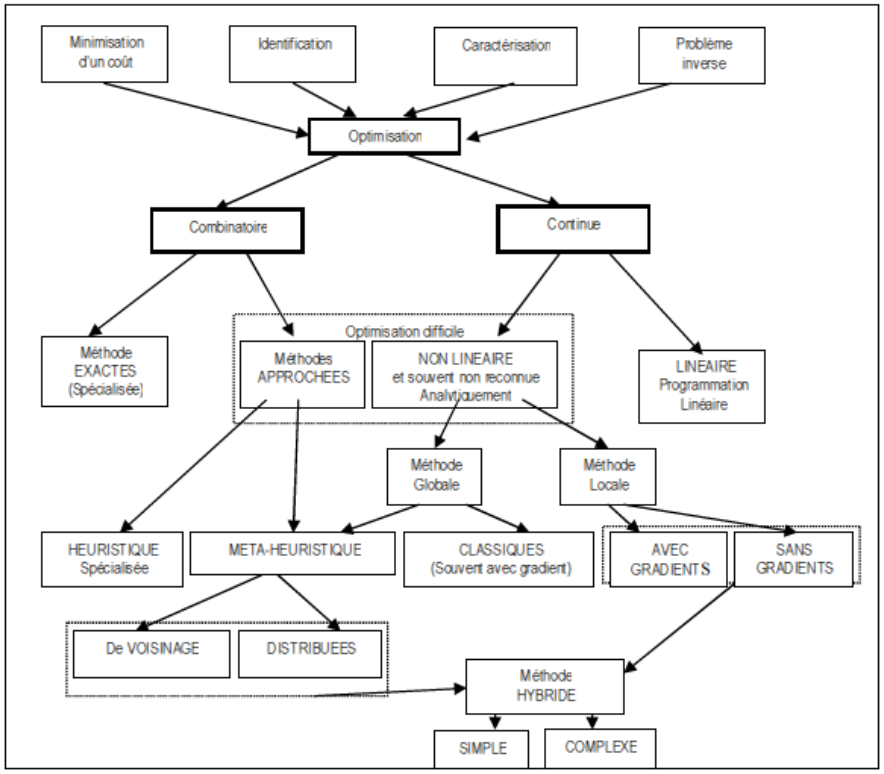


FIGURE 2.3 – Classification des méthodes d'optimisation multiobjectif

2.4.2 Méthodes à préférence progressives

L'utilisateur affine son choix de compromis au fur et à mesure du déroulement de l'optimisation. On retrouve dans cette famille, toutes les méthodes interactives.

2.4.3 Méthodes à préférence à posteriori

L'utilisateur choisit une solution de compromis parmi les solutions de la surface de compromis. Les algorithmes de résolutions des PMOs se répartissent en deux grandes catégories :

1. LES METHODES EXACTES : Ces méthodes permettent d'obtenir l'optimum global de manière exacte pour certains types de problèmes. Elles sont efficaces seulement lorsque le PMO à résoudre est bi-objectif et l'espace de recherche est de petite taille.
2. LES HEURISTIQUES :
   (a) Heuristiques spécifiques : Elles sont dédiées à la résolution d'un PMO spécifique et ne fonctionnent que sur ce type de problèmes.
   (b) Métaheuristiques : Elles sont fondées sur une idée générale. Et peuvent s'adapter à n'importe quel type de problème et donner de bons résultats. On trouve dans cette classe les algorithmes génétiques, la recherche taboue, le recuit simulé,...

   Les métaheuristiques résolvant les PMO peuvent être classées en trois approches :

   i. **Transformation des PMO en uni-objectif**

   A. **Méthodes d'agrégation**

Les méthodes agrégatives fusionnent les différentes fonctions objectives pour se ramener à un problème d'optimisation mono-objectif. Parmi ces méthodes on trouve La méthode de pondération des fonctions objectif *[Coello]*, qui est la plus simple des méthodes d'optimisation multi-objectif. Elle est d'ailleurs appelée « l'approche naïve ». La transformation que l'on effectue est la suivante :

$$f_{eq}(\overrightarrow{x}) = \sum_{i=1}^{N} \omega_i . f_i(\overrightarrow{x})$$

Supposons $\forall i = 1, \ldots, N$,$\omega_i > 0$ avec $\sum_{i=1}^{N} \omega_i = 1$ On peut définir des préférences vis à vis de certaines fonctions objectives en jouant sur les coefficients de pondération.

Discussion

Cette méthode est très efficace point de vue algorithmique, mais elle ne permet pas de trouver les solutions enfermées dans des concavités.

B. **Méthode ε-contrainte**

Ce n'est pas une méthode d'agrégation des fonctions objectives. Elle est aussi dite méthode du compromis [38]. Elle transforme un PMO en un problème d'optimisation monoobjectif de la façon suivante :

- Choisir un objectif à optimiser prioritairement,
- Choisir un vecteur de contraintes initiales ($\overrightarrow{\varepsilon}$)
- transformer le problème en gardant l'objectif prioritaire et en transformant les autres objectifs en contraintes d'inégalités comme suit :

$$\begin{cases} minimiser \overrightarrow{f}(\overrightarrow{x}) \\ \\ Avec \overrightarrow{x} \in S \end{cases}$$

où S : représente le domaine réalisable.

càd ;

$$\begin{cases} minimiser \overrightarrow{f_1}(\overrightarrow{x}) \\ \\ Avec \overrightarrow{f_2} \leq \varepsilon_2, \ldots, \overrightarrow{f_n} \leq \varepsilon_n \\ Avec \overrightarrow{x} \in S \end{cases}$$

Discussion

Cette méthode présente plusieurs inconvénients à savoir :

- la formulation des préférences utilisateur est délicate et nécessite une connaissance approfondie du problème de départ.
- Les contraintes rajoutées compliquent la résolution du problème.

C. **Programmation par but**

On définit un ensemble de buts qu'on espère atteindre pour chaque fonction objectif. L'algorithme tente de minimiser l'écart entre la solution courante et ses buts.

Discussion

Cette méthode est efficace et facile à implémenter mais la définition des buts à atteindre est souvent coûteuse. Pour plus de détails, voir [38], [11].

ii. **Approches non Pareto**

Ces approches traitent séparément les différents objectifs, elles sont sensibles au paysage du front de Pareto (convexité, continuité,...). Elles sont efficaces et faciles à implémenter seulement pour les PMO avec un nombre réduit d'objectifs.

On trouve :

Vector Evaluated Genetic Algorithm (V.E.G.A). Cette méthode est décrite dans [11] et [17].Le danger de cette méthode est d'obtenir, en fin d'optimisation, une population d'individus moyens dans tous les objectifs.

L'ordonnancement lexicographique .C'est une méthode très intuitive qui consiste à considérer les fonctions objectif les unes après les autres et à minimiser à chaque fois un problème mono-objectif, en complétant au fur et à mesure l'ensemble des contraintes [11]. L'inconvénient majeur de cette méthode est qu'elle requiert un choix de la séquence des objectifs à minimiser. Deux choix différents génèrent deux solutions distinctes.

iii. **Approches Pareto**

Ces approches utilisent la notion de dominance lors de la sélection des solutions. Elles sont capables de générer des solutions Pareto situées dans des concavités.

2.5 Métaheuristiques et optimisation multi-objectif

Les métaheuristiques sont des méthodes générales de recherche dédiées aux problèmes d'optimisation difficile. Elles reprennent des idées que l'on trouve parfois dans la vie courante. Les principales heuristiques sont : le recuit simulé, la recherche tabou et les algorithmes génétiques.

Les métaheuristiques se répartissent en trois familles :

- Les méthodes déterministes de recherche d'optimum local :
  Ces méthodes convergent rapidement, mais, la plus part du temps, elles se contentent de trouver l'optimum local.
- Les méthodes déterministes de recherche d'optimum global :
  Ces méthodes permettent de trouver un optimum global et ne reposent pas sur un processus stochastique.
- Les méthodes stochastiques de recherche d'optimum global :
  Ces méthodes reposent sur un processus stochastique pour la recherche de l'optimum global. Elles sont moins rapides que les méthodes déterministes mais elles peuvent trouver un optimum global difficile à atteindre.

2.5.1 Recuit simulé

Le recuit simulé (simulated annealing) a été inventé par les physiciens Kirkpatrick, Gelatt et Vecchi en 1983. Le recuit s'inspire des méthodes de simulation de Metropolis développées aux années 50. L'analogie s'inspire du recuit des métaux en métallurgie : un métal refroidi trop vite présente de nombreux défauts microscopiques, c'est l'équivalent d'un minimum local pour un problème d'optimisation combinatoire. Si on le refroidit lentement, les atomes se réarrangent, les défauts disparaissent, et le métal a alors une structure très ordonnée, équivalente du minimum global pour un problème d'optimisation combinatoire. Cette méthode part du même principe des

méthodes de recherche locale, et elle a l'originalité d'accepter une solution voisine moins bonne que la solution courante avec une certaine probabilité pour échapper aux optimums locaux.

Algorithme 1 Algorithme Recuit simulé

Début

DébutDéterminer la solution initiale s_0, et la température initiale T_0 ;Poser Ł=s_0 et T_i=T_0 ;Calculer $f(s_0)$;Répéter pour chaque itération $i/i = 1, \ldots, n$ (ou jusqu'à ce que T est proche)

2: Choisir $s' \in N(s_i)$;

2: Calculer $\triangle f = f(s') - f(s_i)$;

3: **Si** $\triangle f < 0$**Alors** =s_i ;

4: **Sinon**

Début

tirer p dans [0,1] suivant une distribution uniforme ;

Si $p \leq e^{(-\triangle f \backslash T)}$ **Alors** Ł=s_i ;

Sinon s_i est rejetée ;

Fin ;

5: Calculer T=γT;

6: Retourner Ł

7: **Fin**.

Fin

Pour appliquer le recuit simulé aux MOPs, il suffit de déterminer comment calculer la probabilité d'acceptation d'un individu $\overrightarrow{x}$. La première version du recuit simulé multiobjectif a été proposée par Serafini [46] qui a proposé un ensemble de règles d'acceptation. Dans le cas multiobjectif, trois situations peuvent apparaître en comparant la nouvelle solution $\overrightarrow{x'}$ avec la solution courante $\overrightarrow{x}$:

– $\overrightarrow{x'}$ domine $\overrightarrow{x}$

– $\overrightarrow{x'}$ est dominé par $\overrightarrow{x}$

– $\overrightarrow{x'}$ et $\overrightarrow{x}$ sont non dominées

Dans le premier cas la probabilité d'acceptation est naturellement égale à un, dans le second cas elle est inférieure à un, mais elle reste indéfinie (inconnue) dans le troisième cas. Actuellement, il existe plusieurs versions du recuit simulé multiobjectif.

2.5.2 Recherche taboue

Les méthodes tabou (tabu search) ont été inventées par Glover et Hansen en 1986. Leur principe est le suivant : à chaque itération le voisinage (complet ou partiel) de la solution courante est examiné et la solution minimisant l'augmentation du coût est sélectionnée. Pour éviter le phénomène de cyclage, la méthode interdit de revisiter une solution déjà visitée. Pour cela, une liste taboue contenant les solutions visitées est utilisée.

Le principe de la recherche taboue est présenté par l'algorithme suivant :

Plusieurs versions multiobjectif de la méthode Tabou ont été proposées : Gandibleux [25], Hansen [31], Ben Abdelaziz [5], Barichard

Dans la suite, on ne s'intéressera qu'à cette méthode.

Algorithme 2 Algorithme Recherche taboue

Début

1: Trouver une solution initiale s_0, et poser Ł=s_0,i=0 et $TL = \emptyset$,(TL : liste taboue)

2: Poser i=i+1 ;

3: Déterminer le sous ensemble de voisinage $N^*(s_i) \subset N(s_i)$ tel que :

$\forall s_{i+1} \in$N*(s$_i$) : $(s_i, s_{i+1}) \notin TL$;((s$_i$, s_{i+1})le mouvement de s_ià s_{i+1};

4: Remplacer s_i par s_{i+1} tel que s_{i+1} est la meilleure solution de $N^*(s_i)$;

5: Mettre à jour TL ;

6: Si la condition d'arrêt n'est pas vérifiée, alors aller à 3 ;

Fin

2.5.3 Algorithmes génétiques

Cette classe de méthodes a été inventée par Holland dans les années 70, pour imiter les phénomènes d'adaptation des êtres vivants. L'application aux problèmes d'optimisation a été développée ensuite par Goldberg [28] en 1989. On par d'une population initiale de NS solutions aléatoires, chacune étant codée par une chaîne de caractères appelée *chromosome*. Chaque chromosome est muni d'une mesure d'adaptation qui peut être la fonction objectif.

Un algorithme génétique choisit des paires de chromosomes parents, en favorisant les plus adaptés, et engendre de nouvelles solutions (enfants) en appliquant des opérateurs de croisement et de mutation. On espère ainsi que les bonnes solutions vont échanger par croisement leurs caractéristiques et engendrer des solutions encore meilleures.

Le principe de l'Algorithme génétique se résume par l'algorithme qui suit :

Algorithme 3 Principe de l'algorithme génétique

Début

Génération d'une population initiale de k individus ;

Répéter

1: Evaluation de la fonction objectif f de chaque individu ;

2: Sélection des meilleurs individus ;

3: Croisement entre deux individus sélectionnés : obtention de deux enfants issus de deux parents sélectionnés ;

4: Mutation : transformation aléatoire des gènes de certains individus de la population ;

Jusqu'à(condition d'arrêt)

Retourner la meilleure solution ;

Fin.

2.5.4 Autres métaheuristiques

D'autres Métaheuristiques ont été adapté au cas multiobjectif et ont prouvé leur succès.

- Les colonies de fourmis *[Mariano]*, *[Iredi]*,...
- L'Optimisation par essaim de particules : *[Moore and Champman]*,...
- Les réseaux de neurones artificiels : *[Yoo and Hajela]*, *[Cui et al.]*, *[Coello and Cruz]*
- Les algorithmes culturelles : Coello and Landa.

2.6 Conclusion

Comme nous avons pu le voir tout au long de cete partie, l'optimisation multiobjectif n'est pas une tâche facile.

Tout commence par la définition d'un optimum multiobjectif. Une définition générale et bien établie est celle de Pareto. Mais cette définition étant trop générale, des variantes ont été introduites : la dominance au sens de Geoffrion, optimalité maximale, etc.

Ensuite vient le mode de résolution. En fonction de la connaissance que l'on a du problème, on peut :

- partir à la recherche d'une unique solution si les préférences du décideur sont bien déterminées et si celui-ci ne souhaite pas choisir parmi un éventail de possibilités ;
- préciser les préférences du décideur en cours d'optimisation ;
- approcher la surface de compromis par un "nuage" de solutions.

On a aussi une grande latitude quant au choix du traitement du problème multiobjectif. On peut se ramener à une seule et unique fonction objectif ou alors traiter vectoriellement le problème.

Pour finir, il faut choisir une méthode d'optimisation pour résoudre le problème multiobjectif. Cette dernière tâche est déjà délicate en optimisation monobjectif, alors, on conçoit que l'optimisation multiobjectif soit d'une difficulté plus grande encore.

Chapitre 3

Modélisation mathématique

3.1 Introduction

Le problème d'ordonnancement traité est noté par : $1/r_i/ \begin{cases} \sum_{i=1}^{n} \omega_i C_i \\ \sum_{i=1}^{n} \omega_i T_i \end{cases}$ statique est un des problèmes d'ordonnancement les plus traités dans la littérature. Il est dit statique car toutes les données du problème sont connues à priori. La nature fortement combinatoire du problème étudié présente des difficultés importantes pour la recherche de l'optimalité de ses solutions.

Nous montrons que le problème d'ordonnancement à une machine que nous traitant peut être représenté en un programme linéaire en variables mixtes (entières et bivalentes)

3.2 Modélisation du problème

Cette section est consacrée à la formulation mathématique des différentes contraintes ainsi que les fonctions objectif à minimiser par un programme linéaire que nous proposons pour le problème d'ordonnancement.

3.2.1 Description des données et des variables

n : le nombre de tâches ;

tâche : ensemble des tâches $\{tâche_1, \ldots, tâche_n\}$;

p_i : durée de traitement de la tâche $tâche_i$, $p_i \in N$ $(\forall i = 1, \ldots, n)$;

r_i : la date de disponibilité de la tâche $tâche_i$;

ω_i : poids de la tâche $tâche_i \in N$ $(\forall i = 1, \ldots, n)$;

d_i : date d'échéance de la tâche $tâche_i$;

M : un nombre entier trés grand ;

On définit les variables :

t_i : date de début d'exécution de la tâche $tâche_i$,$t_i \geq 0$, $(\forall i = 1, \ldots, n)$

$$x_{ij} = \begin{cases} 1 & \text{si la tâche } tâche_i \text{ précède la tâche } tâche_j \quad \forall i = 1, \ldots, n, \forall j = 1, \ldots, n, i \neq j \\ 0 & \text{sinon} \end{cases}$$

3.2.2 Description des contraintes

Cette section est consacrée à la formulation des différentes contraintes ainsi que la fonction objectif à minimiser,

Contraintes disjonctives

Ces contraintes spécifient que deux tâche ($tâche_i$,$tâche_j$) ne peuvent pas être exécutées durant le même intervalle de temps.

Les contraintes s'écrivent dans le cas où la tâche $tâche_i$ est exécutée avant la tâche $tâche_j$ par :

$$t_j\text{-}t_i \geq p_i, \forall i, j = 1, \ldots, n \qquad \text{(a)},$$

Par contre, dans le cas où la tâche $tâche_j$ est exécutée avant la tâche $tâche_i$ les contraintes s'écrivent :

$$t_i\text{-}t_j \geq p_j, \forall i, j = 1, \ldots, n \qquad \text{(b)},$$

Il est nécessaire d'assurer qu'une et une seule des deux contraintes (a) et (b) soit satisfaite, et cela se traduit par le respect des inéquations suivantes :

Pour deux tâches différentes $tâche_i$ et $tâche_j$ traitées sur une machine, soit la tâche $tâche_i$ précède la tâche $tâche_j$ ou bien l'inverse, donc on aura :

$$\begin{cases} t_i + p_i \leq t_j + (1 - x_{ij})M & \forall i, j = 1, \ldots, n, i \neq j \\ x_{ij} + x_{ji} = 1 & \forall i, j = 1, \ldots, n, i \neq j \end{cases}$$

Où M est un entier très grand qui est fixé à n'importe quelle borne inférieure du problème, (exemple : M=$\sum_{i=1}^{n}(\omega_i + t_i + p_i)$

Contraintes temporelles

Le temps d'exécution de la tâche $tâche_i$ doit être supérieur ou égale à sa date de disponibilité, donc :

$$t_i \geq r_i \quad \forall i = 1, \ldots, n$$

3.2.3 Critères à optimiser

On cherche à minimiser :

i. la somme pondérée des dates de fin d'exécution des tâches

$$\text{Min} z_1 = \sum_{i=1}^{n} \omega_i (t_i + p_i)$$

ii. la somme pondérée des retards absolus des tâches

$$\text{Min} z_2 = \sum_{i=1}^{n} \omega_i \ \max_{1 \preceq i \preceq n}(0, \text{C}_i \text{ -d}_i)$$

3.2.4 Modèle

On obtient le modèle suivant :

$$\begin{cases} \min z_1 = \sum_{i=1}^{n} \omega_i (t_i + p_i) \\ \min z_2 = \sum_{i=1}^{n} \omega_i \max_{1 \preceq i \preceq n} (0, C_i - d_i) \\ \\ \begin{array}{ccccll} t_i + p_i & \leq & t_j + (1 - x_{ij})M & \forall i, j = 1, \ldots, n, i \neq j & (1) \\ x_{ij} + x_{ji} & = & 1 & \forall i, j = 1, \ldots, n, i \neq j & (2) \\ t_i & \geq & r_i & \forall i = 1, \ldots, n & (3) \\ t_i & \geq & 0 & \forall i = 1, \ldots, n & \\ x_{ij} & \in & \{0, 1\} & \forall i, j = 1, \ldots, n, i \neq j & \end{array} \end{cases}$$

Taille du modèle Notons que la dimension du programme linéaire se traduit par le nombre de variables et de contraintes utilisées.

Notre modèle comporte :

– n variables entières

– $n(n-1)$ variables bivalentes

Le nombre de contraintes est :

– $n(n-1)$ contraintes de types (1),

– $\frac{n(n-1)}{2}$ contraintes de types (2),

– n contraintes de types (3),

Donc : $\frac{n(3n-1)}{2}$ contraintes.

Le modèle mathématique permet de trouver la solution exacte du problème posé pour des instances de taille réduite. Le modèle proposé peut être résolu par un des deux solveurs : Lingo ou Cplex.

Le cas n'est pas aussi simple pour un problème de taille importante.

Exemple illustratif Rappelons que notre modèle mathématique possède n variables entières et $n(n-1)$ variables bivalentes et $\frac{n(3n-1)}{2}$ contraintes.

Considérons l'exemple suivant pour un problème mono-objectif minimiser la somme pondérée des dates de fin d'exécution des tâches.

n= 20 ;

R=[55,88,13,90,512,135,11,738,934,751,479,888,1118,262,457,688,1907,1698,2006,2209] ;

P=[80,27,80,47,29,47,95,64,60,82,4,65,48,33,86,49,5,85,95,44] ;

W=[4,10,8,4,8,6,6,4,2,10,9,7,8,7,4,1,7,7,2,7] ;

Résultat en Cplex : // solution (optimal) with objective 97614

t_i = [342 93 13 120 512 167 214 833 962 751 479 897 1118 309 541 688 1907 1698 2006 2209] ;

X =[00001001111110111111]
[10011111111111111111]
[11011111111111111111]
[10001111111111111111]
[00000011110110111111]
[10001011111111111111]
[10001001111111111111]
[00000000100110001111]
[00000000000010001111]
[00000011001100011111]
[00001001110110111111]
[00000000100010001111]
[00000000000000001111]
[10001001111110111111]
[00000001110110011111]
[00000001110110001111]
[00000000000000000011]

[00000000000000001011]
[00000000000000000001]
[00000000000000000000] ;

Le modèle testé comporte 20 variables entiéres et 380 variables bivalentes et 590 contraintes.

Pour tester l'efficacité du modèle il est préférable de générer des instances de différentes taille est de les tester.

Chapitre 4

Méthode de résolution

4.1 Introduction

Un modèle mathématique sous forme d'un programme linéaire en variables mixtes (entières et bivalentes) est proposé pour la résolution du problême d'ordonnancement d'atelier à une seule machine. Ce type de modèle mathématique est connu d'être très difficile à résoudre et demande beaucoup de temps de calcul. Dans la littérature, deux types de méthodes de résolution sont généralement utilisées pour résoudre les programmes linéaires en variables entière (ou mixtes) : les méthodes utilisant les coupes comme la coupe de Gomory et les méthodes utilisant les arbres de recherche comme la séparation et évaluation.

Les deux solveurs IBM ILOG CPLEX 12 et Extended Lingo 8.0 utilisent ces deux types de méthodes de résolution. L'objectif de cette partie est d'effectuer des tests en utilisant des instances pour justifier le choix d'un de ces solveurs pour la résolution du modèle mathématique proposé .

4.2 Résolution du modèle mathématique associé

4.2.1 Pourquoi choisir CPLEX

ILOG CPLEX (plus communément appelé "CPLEX") est un outil informatique d'optimisation. Son nom fait référence au langage C et à l'algorithme du simplexe.

Il est composé d'un exécutable (CPLEX Interactif) et d'une bibliothèque de fonctions pouvant s'interfacer avec différents langages de programmation (CPLEX Callable Library) : C, C charp, Java et Python. IBM ILOG CPLEX Optimization permet de développer et de déployer rapidement des applications d'aide à la décision analytiques basées sur la technologie de l'optimisation avec des fonctions de débogage, de test, de mise au point et de génération d'applications. Il est utilisé face à des algorithmes robustes et hautement performants pour résoudre les problèmes d'optimisation de grande taille propres aux organismes, ceci avec la vitesse nécessaire pour les applications interactives actuelles.

Cplex 12.1 d'IBM Ilog permet de Résoudre les problèmes de programmation linéaire, de programmation mixte en nombres entiers, de programmation quadratique et de programmation quadratique mixte en nombres entiers avec les optimiseurs IBM ILOG CPLEX.

4.2.2 Pourquoi choisir Lingo

LINGO est un outil complet conçu pour formuler rapidement, facilement et efficacement les problèmes d'optimisation de modèles linéaires, non linéaires, quadratiques, de cônes du second degré et stochastiques. LINGO met à votre disposition : un langage puissant et un environnement complet pour construire et éditer vos modèles, le tout complété d'un jeu de solveurs ultra-performants.

LINGO est livré avec un jeu de solveurs pour l'optimisation linéaire, non-linéaire (convexe ou non convexe), quadratique, sous contraintes, et en nombre entier. Vous n'avez même pas à vous préoccuper du choix du solveur : en effet, LINGO interprète lui-même vos formulations et sélectionne automatiquement le solveur adapté à chaque problème.

Vous pouvez construire et résoudre vos problèmes d'optimisation directement dans LINGO, ou appeler LINGO depuis une application personnalisée grâce aux DLL et aux liens OLE inclus. Vous pouvez même faire appel à LINGO depuis une macro Excel ou une base de données.

4.2.3 Etude comparative entre les solveurs Cplex et Lingo

Générations des instances

Les différents paramètres du problème d'ordonnancement (date de disponibilité R, durée opératoire P et le poids W) sont générés aléatoirement. Pour chaque instance le nombre de tâches (n) prends ses valeurs dans l'ensemble 5, 10, 15, 20, 25,30. Les durées opératoires (P) sont générées aléatoirement selon la loi uniforme dans l'intervalle [1 100]. Les poids (W) par la loi uniforme dans l'intervalle [1 10] et les dates de disponibilité (R) sont générées par la loi uniforme dans l'intervalle [0 , 50.5*n*a] où $\alpha \in$0.2 ; 0.4 ; 0.6 ; 0.8 ; 1 ; 1.25 ; 1.50 ; 1.75 ; 2.0 ;3.0 [40].

Déroulement des tests

Dans ces tests, nous utilisant la résolution du problème d'ordonnancement mono-objectif qui consiste à minimiser la somme pondérée des dates de fin d'exécution des tâches afin d'en déduire le solveur le plus efficace pour la résolution du programme mathématique.

La première étape consiste à fixer le nombre de tâches n, et pour chaque n fixé on génère trois instances (n=5, n= 10, n=15, n= 20, n=25 et n=30). Les deux premières colonnes des deux tableaux 1 et 2 indiquent respectivement le nom de l'instance et sa taille. La colonne 3 (resp. colonne 5) de chaque tableau présente le coût de la solution optimale de l'instance résolue par le solveur Cplex (resp par le solveur Lingo). La colonne 4 (resp. colonne 6) indique le temps de déroulement du solveur Cplex (resp du solveur Lingo). Dans le tableau 2, le caractère (-) indique que le solveur a échoué à donner une solution optimale après un déroulement de 3 heures.

Analyse des résultats

D'après les résultats obtenus dans les deux tableaux 1 et 2, nous remarquons que les deux solveurs (Cplex et Lingo) ont résolus de manières optimales les instances de petites tailles (5, 10 et 15 tâches) en un temps très raisonnable (moins de 42.31 sec) (voir tableau1). Pour les instances de taille 20, 25 et 30 tâches, le solveur Cplex a résolu toutes les instances de manière optimale en un temps qui varie entre 0.27 sec et 10086.40 sec. Par contre, solveur Lingo n'a résolu aucune de ces instances après 3 heures d'éxécution.

De ces tests nous concluons que le programme mathématique proposé pour la résolution du problème d'ordonnancement d'atelier de production à une seule machine peut être mieux résolu en utilisant le solveur Cplex.

Le modèle mathématique permet de trouver la solution exacte du problème posé pour des instances de taille réduite. Le modèle proposé peut être résolu par un des deux solveurs : Lingo ou Cplex.

La version d'évaluation de LINGO est limitée par la résolution des instances à moins de 5 tâches. Cependant le solveur Cplex nous a permis de résoudre, en temps raisonnable, des instances de tailles supérieur à 20 tâches.

Instances	Nombre de tâches	Cplex		Lingo	
		Solution optimale	CPU (sec)	Solution optimale	CPU (sec)
Instance 01	5	3348	0.07	3348	0.01
Instance 02	5	3288	0.02	3288	0.01
Instance 03	5	4279	0.03	4279	0.01
Instance 04	10	12164	2.50	12164	5.16
Instance 05	10	10632	1.30	10632	0.48
Instance 06	10	9786	0.64	9786	0.05
Instance 07	15	95001	0.11	95001	0.01
Instance 08	15	106009	42.31	106009	14.37
Instance 09	15	50090	0.16	50090	0.02

FIGURE 4.1 – Les instances de 01 à 09 résolues par les deux solveurs

Instances	Nombre de tâches	Cplex		Lingo	
		Solution optimale	CPU (sec)	Solution optimale	CPU (sec)
Instance 10	20	97614	0.27	-	10800
Instance 11	20	13207	436.92	-	10800
Instance 12	20	21123	24.58	-	10800
Instance 13	25	45766	0.64	-	10800
Instance 14	25	47138	79.56	-	10800
Instance 15	25	48295	344.68	-	10800
Instance 16	30	55414	10086.40	-	10800
Instance 17	30	45880	2319.81	-	10800
Instance 18	30	684736	2470.76	-	10800

FIGURE 4.2 – Les instances de 10 à 18 résolues par les deux solveurs

4.3 Approche métaheuristique de résolution

4.3.1 Approche de résolution

« Les Méta-heuristiques sont des stratégies de recherche itérative de haut niveau, destinées à l'exploration de l'espace de solutions par l'utilisation de différentes techniques.

Ces méthodes n'essayent pas d'examiner toutes les solutions possibles, mais de trouver dans un temps raisonnable une solution satisfaisante par investigation intelligente de l'espace en s'appuyant sur deux principes : la diversification de la recherche dans tous "les endroits" de l'espace ; et l'intensification, en exploitant chaque point de l'espace d'état »

Dans cette section, nous proposons une approche de résolution basée sur l'algorithme du recherche taboue. Dans la littérature, il existe des travaux ayant adapté l'algorithme du recherche taboue, parmi les travaux récents, nous citons ceux de Keith Schmidt en 2001 [45] qui ont utilisé la recherche taboue pour résoudre un problème d'ordonnacment job shop, un problème des temps de réglage de séquence dépendante (setup), et de U.Al-Turki et al.en 2001 [45] qui ont utilisé aussi la recherche taboue pour une classe de problème d'ordonnancement d'une seule machine. et on peut pas passé sans cité le dernier travail de la recherche taboue fait par F.Xhafa & J.Carretero en 2009 [50]. c'est l'utilisation de l'algorithme recherche taboue pour la planification d'emploi indépendant dans la grille de calcul.

L'objectif principal de telles méthodes est de générer une variété de solution Pareto-optimales diversifiées dans l'espace de recherche.

G.Dahl & al. [15] dans une approche de la recherche taboue au problème minimisation des canaux en 1995, ils ont mis une méthode d'agrégation, la recherche taboue a été appliquée pour résoudre un problème d'affectation de fréquence. l'algorithme est exécuté avec un seul jeu de poids qui est fixé suivant les priorités des objectifs. A.Hertz & al. [33]l'utilisation de la recherche taboue pour la résolution du problème la formation des cellules dans la technologie de groupe avec des objectifs multiples dont méthode $\epsilon-$ contrainte, la recherche taboue est utilisée pour résoudre une suite de sous-problèmes à un seul objectif sous plusieurs contraintes, dans lesquels chaque objectif est considéré tour à tour, en suivant leur ordre d'importance relative (ordre lexicographique). En fait, cette méthode va chercher une bonne solution pour la fonction semblant la plus importante. En suite, la deuxième fonction dans l'ordre relatif d'importance va être optimisée en ayant comme contrainte supplémentaires ne pas trop détériorer la valeur obtenue pour la première fonction objectif(application d'un seuil). Ce procédé est itéré pour les autres fonctions. les auteurs testent leur approche pour des problèmes bi-critères. Le problème (PMO_q) résolu à l'étape q=1,...,n est :

$$PMO_q = \begin{cases} f^*{}_q = \min f_q(x) \\ \text{s.c } f_r(\text{x}) < f'_r, \text{r}=1,...,\text{q-1} \\ \text{x} \in \text{C} \end{cases}$$

où f'_r est le seuil sur la valeur de l'objectif f_r, calculée à partir de l'optimum f^*_r du problème PMO_r plus une détérioration maximale acceptée pour léobjectif f_r. l'importance relative de chaque objective peut être contrôlée en transformant la valeur de seuil.

PMO : arbre recouvrant minimum G.Zhou and M.Gen [51], sac à dos F.Ben Abdelaziz [6], conception de réacteurs G.T.Parks and I.Miller [41], distribution de l'eau D.Halhal et al. [30],ect. X.Gandibleux et al. La procédure recherche Taboue pour résoudre les problèmes multi-objectifs d'optimisation combinatoire, dans cet article la programmation par but la recherche taboue utilisée est classique, si ce n'est la détermination du choix du meilleur voisin. La solution sélectionnée est celle qui représente le meilleur compromis parmis toutes les solutions du voisinage qui ne sont pas tabous. Pour déterminer cette solution, une distance est introduite de telle sorte que le voisin accepté "contente" toutes les fonctions objectifs. La fonction coût utilisée pour selectionné le meilleur voisin est une norme pondérée (norme L_p de Tchebycheff) par rapport au vecteur idéal. Tout au long du processus, on garde en mémoire un nombre M de solutions. Toute solution acceptée lors de la recherche taboue est comparée au M solutions mémorisées. Si une ou plusieurs solutions sont dominées, elles sont éliminées de cet ensemble et remplacées par de nouvelles (pour le moment non dominées). Finalement, une solution est proposée, celle qui représente le meilleur compromis parmi les M solutions sauvegardées (en utilisant la notion du distance précédemment mentionnée).

4.3.2 Principe général de l'algorithme du recherhe tabou

Historique

La recherche Taboue est une métaheuristique développée par Glover [27]. Elle est basée sur des idées simples, mais elle est néanmoins très efficace. Elle combine une procédure de recherche locale avec un certain nombre de règles et de mécanismes pour surmonter l'obstacle des optima locaux, tout en évitant de cycler. Elle a été appliquée avec succès pour résoudre de nombreux problèmes difficiles d'optimisation combinatoire. Le principe de la méthode Tabou est simple : On part d'une solution initiale x, puis on cherche le meilleur voisin *s(x)* dans le voisinage de x *[s(x)]*. L'originalité de la méthode réside dans le fait que l'on retient le meilleur voisin de x même si celui-ci est plus mauvais que x. Ce critère autorisant les dégradations de la fonction objectif évite à l'algorithme d'être piégé dans un minimum local, mais il induit un risque de cyclage. Pour régler ce problème, l'algorithme mémorise la liste des transformations interdites (telle que de $s(x) \longrightarrow x$) dans la liste Taboue interdisant la transformation inverse d'une transformation faite récemment.

Principe

Etape1 : Construire une solution initiale $S(0)$
Etape2 : $S^* \longleftarrow S(0)$, Liste taboue $= \emptyset$
Etape3 : $S - courante \longleftarrow S(0)$
On génère les voisins de la solution courante ($v(S - courante)$)
Sélectionné meilleure solution de $v(S - courante)$ par exemple $s^{'}$ et qui n'appartient pas à la liste taboue.
$S - courante \longleftarrow s^{'}$
Etape4 : si ($S^* > S - courante$) alors $S^* \longleftarrow S - courante$
Critêre d'arrêt : si le nombre d'itération est atteint.

Le même mécanisme est répété pour tous les espaces d'une solution courante.

4.4 Adaptation de l'algorithme du recherche taboue pour le problème d'ordonnancement

L'organigramme présenté dans la figure ci-après, détaille le fonctionnement de l'approche proposée.

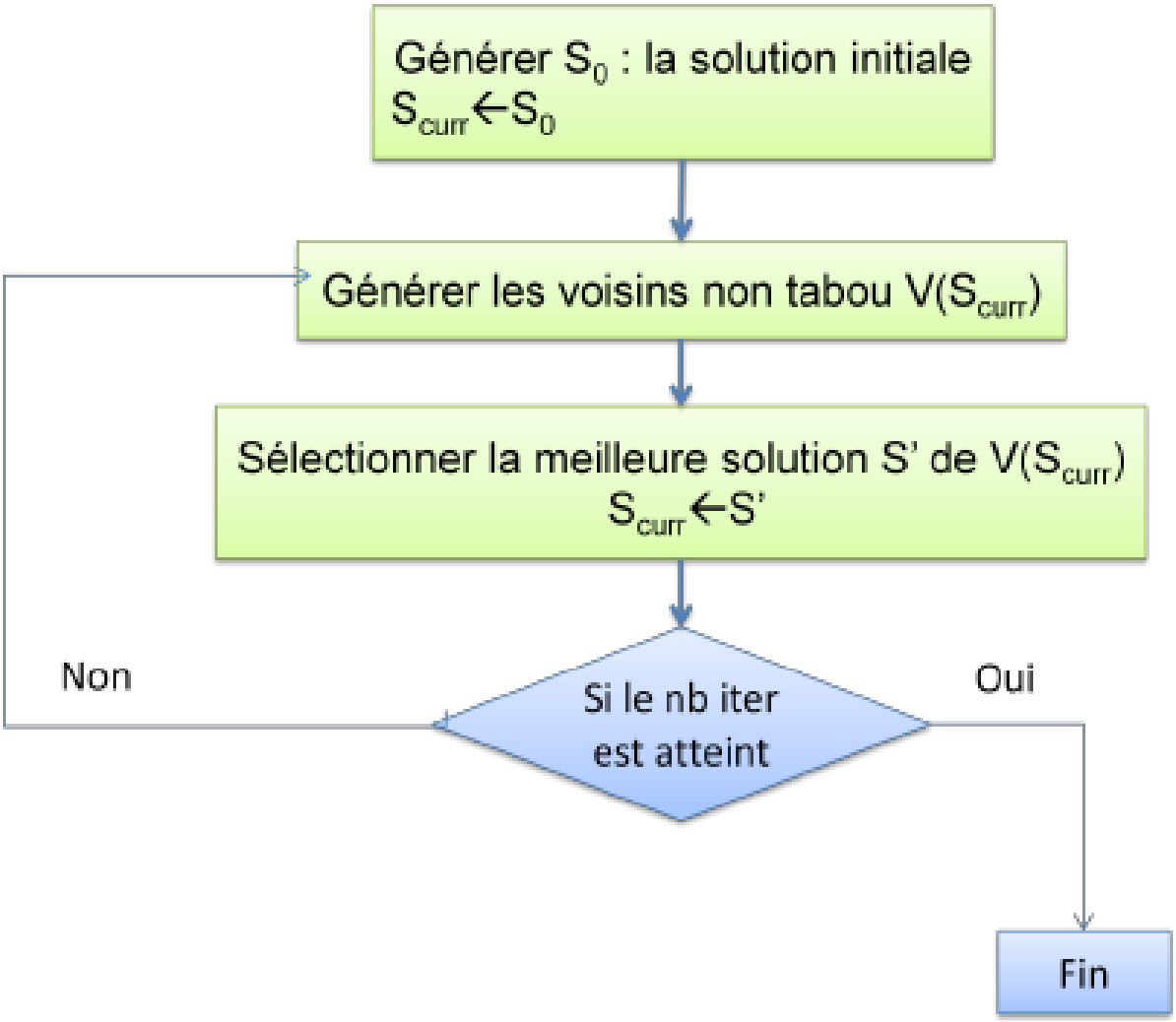


FIGURE 4.3 – Approche de résolution : Recherche taboue

Nous devons rappeler le théorème proposé par Pinedo pour le problème d'ordonnancement $1//\sum C_i$. [42] page 36.

Théorème 4.1 *La rège WSPT est optimale pour* $1//\sum_{i=1}^{n} C_i$ *En se basant sur ce théorème, nous ordonnançons les tâches selon l'ordre décroissant de* w_i/p_i, *i=1,..., n (WSPT : Small Processing Time avec pondération) .*

La preuve du théorème :

La contraposée. Supposons un ordonnancement S, qui n'est pas optimale dans la règle WSPT. Dans cet ordonnancement, il doit y avoir au moins deux tâches adjacentes, soit la tâche j suivie de la tâche k, tels que

$$w_j/p_j < w_k/p_k$$

Supposons la tâche j sa date de début d'éxécution à l'instant t. Effectuer un soi-disant échange des pairs adjacents sur les tâches j et k. Appelons le nouveau ordonnancement S' . Alors que dans l'ordonnancement initial S la tâche j commence son traitement à l'instant t et est suivie par la tâche k, dans le nouveau ordonnancement S', la tâche k commence son traitement à l'instant t et est suivie par la tâche j. Tous les autres tâches restent dans leur position d'origine. La somme pondérée des dates de fin d'éxécution des tâches traitées avant de faire les échages entre les tâches j et k, ni la somme pondérée des dates de fin d'éxécution des tâches traitées après les tâches j et k. Ainsi, la différence dans les valeurs des objectifs des ordonnancements S et S' est due uniquement aux tâches j et k .

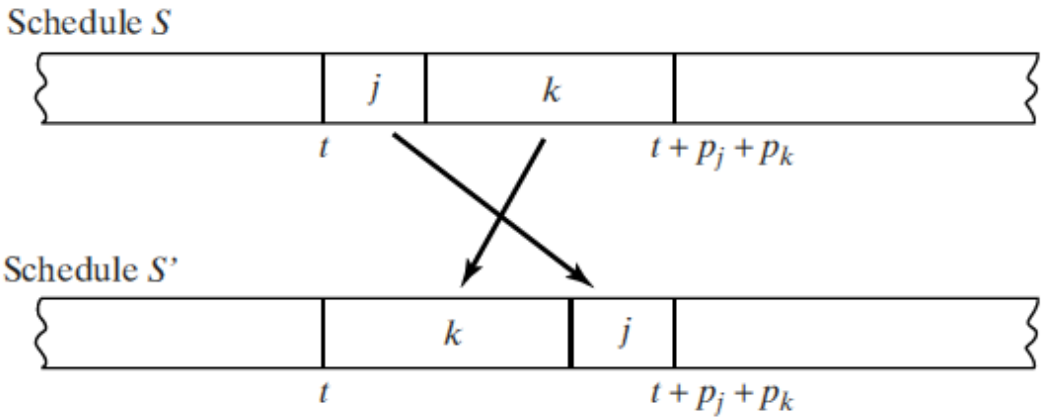


FIGURE 4.4 – Echanger le couple de tâches j et k

ci-dessous, la somme pondérées des dates de fin d'éxécution des tâches j et $k$de S :

$$(t + p_j)w_j + (t + p_j + p_k)w_k$$

, Tandis que S est :

$$(t + p_k)w_k + (t + p_k + p_j)w_j$$

Il est simple de vérifier que si $w_j/p_j < w_k/p_k$ la somme pondérée des dates de fin d'éxécution des tâches de l'ordonnancement S' est inférieure strictement à la somme pondérée des dates de

fin d'éxécution des tâches de l'ordonnancement S. Cela contredit l'optimalité de S et complète la démonstration du théorème.

4.4.1 Générer une solution voisine

On génère les voisins d'une solution c'est à dire, de trouver un autre ordre des tâches soit par permutation (swap) ou bien par déplacement (move) [14].

L'article des auteurs CRAUWELS & WASSENHOVE présente plusieurs heuristiques de recherche locale pour le problème d'ordonnancement à une seule machine pour minimiser La somme pondérée des retards absolus . Ils introduisent un nouveau schéma de codage binaire pour une représentation de solutions, avec une heuristique pour décoder les représentations binaires en séquences réelles. Ce schéma de codage binaire est comparée à la représentation habituelle «naturel» de permutation pour la descente, recuit simulé, avec l'acceptation de recherche taboue et des algorithmes génétiques sur une pile d'essai de problème. Les résultats des calculs indiquent que toutes les heuristiques qui emploient ce codage binaire sont très robustes et elles produisent de façon constante des solutions de bonne qualité, surtout quand les implémentations multi-start sont utilisés à la place de long terme. Le codage binaire est également utilisé dans un nouvel algorithme génétique qui fonctionne très bien et nécessite relativement peu de temps de calcul.

Une comparaison des méthodes de recherche de voisinage qui utilisent la permutation et la représentation binaire montre que les méthodes basées sur-permutation ont une plus forte probabilité de générer une solution optimale, mais elles sont moins robustes que certaines solutions. Les méthodes de recherche de voisinage, recherche taboue dominent clairement les autres.

La recherche taboue est la troisième stratégie de recherche à voisinage conçu pour permettre d'échapper un optimal local. Soit N le voisinage, et soit x_c la solution courante. La recherche taboue exécute le mouvement de x_c à x, où x est la meilleure solution dans N (x_c) (ou un sous-ensemble de N (x_c)), même si ce mouvement se détériore. Si, est le cas le voisinage est symétrique, càd, si $x_c \in$ N(x) lorsque x $\in$ N(x_c), alors il ya un danger de cyclage quand, à la prochaine étape, on génère N (x). La possibilité existe que x_c pourrait être la meilleure solution dans N (x), dans ce cas, la procédure serait immédiatement remise à zéro x_c soit la solution courante, et on alterne entre x et x_c.

Pour éviter le cyclage, une liste taboue est créé à interdire certains mouvements. Après un mouvement est exécuté, une (ou parfois plusieurs) de ses attributs ou des caractéristiques est stocké dans cette liste, et se déplace ce qui permettrait d'inverser cet attribut sont tabou. La liste est organisée de manière cyclique de sorte qu'un mouvement reste tabou que pendant un certain nombre prédéterminé d'itérations. Les mouvements tabou ne sont jamais autorisés sauf si un critère d'aspiration est utilisé, ce qui permet le statut de tabou de mouvement à être remplacé.

Les principales caractéristiques de la recherche taboue sont décrits par Glover [27]. En plus de choisir une représentation des solutions et un voisinage, une méthode de recherche taboue nécessite la sélection d'un ou plusieurs attributs pour stocker sur la liste taboue, spécification des mouvements qui sont interdits, et la définition de critères d'aspiration. Il y a, en outre, les fonctionnalités optionnelles telles que des dispositifs de diversifier la recherche. L'objectif de la diversification est de conduire la recherche dans des régions inexplorées de l'espace de solution.

4.4.2 Choix d'un voisinage adapté

Pour la représentation de permutation, le GPI voisinage nécessite un grand calcul d'investissement pour trouver le meilleur voisin. L'expérience des auteurs montre qu'il est préférable d'effectuer plusieurs itérations dans un voisinage restreint. Une méthode pour restreindre le voisinage est de ne considérer que les mouvements API. Toutefois, en vertu de cette approche, les mouvements sont trop restrictif pour permettre suffisamment de différentes régions de l'espace des solutions à rechercher. Comme alternative, nous créons un petit sous-ensemble contenant les voisins d'interchange n générales qui sont choisi en générant *(u, v)* paires d'instance. Un autre avantage de ce petit voisinage est qu'il existe une plus grande probabilité d'échapper à un optimum local, ce qui évite une longue séquence de mouvements sur une région . D'autre part, il est possible que certains bons mouvements améliorés ne sont pas testés parce que les voisins correspondantes ne figurent pas dans le sous-ensemble. Par conséquent, nous intégrons une étape d'intensification qui est décrite ci-dessus. Lorsque le codage binaire est utilisé, le meilleure mouvement non-tabou est choisi parmi l'inversion de bits de voisinage, qui est décrit précedemment. Pour les deux représentations, si un voisin non-tabou se trouve ce qui améliore la valeure de la fonction objective courante, alors le mouvement correspondant est exécuté sans autre de voisinage de la solution courante.

4.4.3 Liste taboue

Pour le codage de permutation, après chaque déplacement dans lequel les positions des tâches u et v sont échangés, la tâche en position v de la nouvelle séquence est stockée dans la liste taboue avec la valeur correspondante de la fonction objectif de la nouvelle séquence. La longueur de cette liste est égal à 7 dans la mise en œuvre des auteurs. Quelconque voisin qui est créé en échangeant une tâche qui figure sur la liste taboue est interdite sauf si un critère d'aspiration est satisfait. Ce critère est satisfait si la valeur de la fonction objectif de la séquence créée par cet échange est strictement inférieure à la valeure de la fonction objective dans la liste taboue qui correspond à la tâche qui est échangée lors de la création de cette nouvelle solution. Pour la représentation binaire, l'état de tabu est caractérisée par l'élément juste inversé :
cet élément ne peut pas être reinverse à sa valeur d'origine pendant un nombre d'itérations. La langueur de la liste taboue dépend de la taille du problème et est égal à *n 3*. Ici, un simple critère d'aspiration est utilisé : si la valeur de la fonction objectif d'un voisin tabou est meilleure que celle de toutes les solutions générées à ce jour, alors son état de tabou est remplacé.

4.4.4 Intensification et diversification

Selon Glover, une bonne heuristique de recherche stratégique est d'alterner entre les phases impliquant intensification et la diversification. La liste taboue est un dispositif de mémoire à court terme qui contribue à diversifier la recherche. Toutefois, afin de renforcer l'inhérente de l'intensification et la diversification des composants sont toujours présents dans la recherche taboue, nous intégrons deux appareils supplémentaires. Tout d'abord, comme dans l'heuristique de recuit simulé de Potts & Van Wassenhove [43], un algorithme de descente est appliqué à trouver un optimum local chaque fois une augmentation de la fonction objective est suivie immédiatement par une diminution. L'algorithme de descente utilise le voisinage de l'API. Parce que la liste

tabue est ignorée lors de la procédure de descente, le résultat est une recherche intensifiée à un optimum local. Toutefois, pour éviter les perturbations de l'algorithme de recherche taboue, la recherche continue de la séquence qui est entrée dans l'étape de descente, au lieu de partir de la séquence optimum local qui est sortie. Il n'y a aucune étape intensification supplémentaire sous la représentation binaire, parce que la dernière étape de notre décodage de l'heuristique effectue la descente avec le voisinage de l'API. Pour motiver l'utilisation de notre second dispositif, supposons que, pendant un certain nombre successive d'itérations le meilleur déplacement ne change pas la valeur de la fonction objectif parce que les deux tâches qui sont échangées sont tous les deux en avance. Un mouvement plus forte diversification est alors nécessaire de conduire la recherche dans une nouvelle région de l'espace de solution. Pour la représentation de permutation, ceci est obtenu en imposant des restrictions supplémentaires concernant la sélection des paires *(u, v)* . Nous choisissons le meilleur déplacement de trois paires de candidat *(u, v)* , où u est généré aléatoirement à partir de la première $n\ 4$ positions dans la séquence et v est est généré aléatoirement parmi les derniers $n\ 4$ positions. Nous utilisons cette dispositif de diversification après l'exécution de $n\ 4$ itérations consécutives avec des mouvements neutres. Nous adoptons également un dispositif pour la diversification de la représentation binaire. Tout d'abord, une nouvelle chaîne de bits est dérivée de la séquence décodée à partir de l'itération précédente. Les éléments correspondant aux tâches avancées de cette séquence décodée sont mis à 1 ; les autres sont mis à 0. Ensuite, seuls les voisins sont mis a1 à a0 sont considérés. Par conséquent, il existe une plus grande probabilité que le prochain mouvement se détériore, en raison d'une tâche supplémentaire sera en retard. Nous appliquons cette étape diversification après 10 mouvements neutres consécutifs.

4.5 Conclusion

Un modèle mathématique sous forme d'un programme linéaire bi-objectif en variables mixtes (entières et bivalentes) est proposé pour la résolution du problème d'ordonnancement d'atelier à une seule machine étudié. Ce type de modèle mathématique est connu d'être très difficile à résoudre et demande beaucoup de temps de calcul.

Dans la littérature, deux types de méthodes de résolution sont généralement utilisées pour résoudre les programmes linéaires en variables mixtes ; pour cela nous avons proposé une méthode de résolution dans la deuxième partie.

Nous avons décrit l'approche métaheuristique proposée, elle est basée sur l'algorithme du recherche taboue.

Dans le chapitre qui suit, nous allons mener une série d'expérimentations afin de juger sur la performance des modèles mathématiques, ainsi que l'efficacité des variantes de l'algorithme de résolution proposées.

Chapitre 5

Expérimentations numériques

5.1 Introduction

Dans ce chapitre, nous présentons les expérimentations réalisées pour l'évaluation des performances des méthodes développées dans le chapitre 3 pour la résolution du problème que nous traitons, puis nous analysons les différents résultats obtenus. La première méthode est basée sur le modèle mathématique pour la résolution exacte du problème, alors que la seconde est un algorithme du recherche taboue pour la résolution approchée du problème. Ainsi, dans la première partie, une série de tests sont effectués pour l'évaluation du modèle mathématique. Nous précisons que ce modèle est résolu par le solveur CPLEX 12.5.

Dans un second temps, nous testons les variantes de l'algorithme du recherche taboue proposées pour la résolution approchée du problème. Les développements informatiques sont effectués en C++ Builder XE4 sous Windows 7, et les tests expérimentaux sont réalisés sur un PC équipé d'un processeur Intel (R) Core (TM) i3-2328M CPU à 2.20 GHz et ayant 4 Go de mémoire vive (RAM).

Les instances utilisées pour la validation des méthodes sont de deux catégories : la première catégorie d'instances est générée aléatoirement (cf. Annexe) alors que la seconde catégorie (cf. chap4, § IV.2) utilise les Benchmarks d'OR-Library adaptés au problème étudié $1/r_i/\begin{cases} \sum_{i=1}^{n} \omega_i C_i \\ \sum_{i=1}^{n} \omega_i T_i \end{cases}$. Cette librairie est téléchargée à partir du site : http : //people.brunel.ac.uk/ mastjjb/jeb/orlib/wtinfo.html

5.2 Benchmarks d'OR-Library

Les Benchmarks d'OR-Library sont des problèmes de référence servants de mesure. Ils sont construits par plusieurs auteurs pour tester les performances des approches de résolution.

Chaque Benchmark d'OR-Library est une instance de taille n x m, (n étant le nombre de travaux et m le nombre de machines), avec la spécification que chaque travail comporte exactement m opérations et visite chaque machine une et une seule fois.

Le fichier d'OR-Library contient 3 fichiers avec des précisions données comme suit :

Il y a trois fichiers WT40 WT50, et contenant le WT100 d'instances de taille 40, 50, et 100 respectivement. Chaque fichier contient les données pour 125 instances. Les durées de traitement des tâches n placés en premier, suivi par les n poids, et, enfin, les dates d'échéance de la tâche n, pour chaque 125 instance.

Par exemple, dans WT40 les 40 premiers nombres entiers dans le fichier sont les durées de traitement pour les 40 tâches dans la première instance. Les prochaines 40 entiers sont les poids de la première instance. Les prochaines 40 entiers sont les dates d'échéance de la première instance. Les prochaines 40 entiers sont les durées de traitement de la deuxième instance, etc.

Les valeurs optimales et la meilleure solution connue

Les valeurs optimales de solutions sont disponibles pour 124 et 115 pour les instances de 40 et 50 tâches, respectivement. Les 40 tâches non résolus est le numéro 19, et pour 50 on a 11, 12, 14, 19, 36, 44, 66, 87, 88 et 111 sont toujours pas résolus.

Les valeurs pour les problèmes non résolus figurant dans les fichiers wtopt40 et wtopt50 sont les plus connus à Crauwels, Potts & Van Wassenhove en 1996 [14].

Les valeurs des solutions connues ne sont pas améliorées depuis des temps et semblent tout à fait susceptibles d'être optimale.

Les meilleures valeurs de solution connue à Crauwels, Potts & Van Wassenhove en 1998 [43] pour le problème de 100 tâches sont donnés dans le fichier wtbest100a. Ces valeurs de solution ont été utilisées comme la plus connue par les deux : Crauwels, Potts Van Wassenhove en 1998 et Congram, Potts & van de Velde en 1998 [13].

Par conséquent l'utilisation des meilleures valeurs de solution connue à Crauwels, Potts & Van Wassenhove en 1998 permet aux résultats de futures heuristiques à être comparées directement aux tableaux figurant dans ces documents.

L'heuristique de recherche local [13] a dans certains cas, trouvé de meilleures solutions aux problèmes de 100 tâches que ceux connu par Crauwels, Potts & Van Wassenhove en 1998. Les solutions les plus connus à ce jour sont donnés dans le fichier wtbest100b.

Le fichier le plus grand WT100 de la taille de 230 Ko (environ) L'ensemble des fichiers est de taille 440 Ko (environ).

5.3 Evaluation de la performance de l'approche

Le modèle mathématique permet de trouver la solution exacte du problème posé pour des instances de taille réduite. Le modèle proposé peut être mieux résolu par le solveur Cplex (chapitre 3).

Pour des instances de grande taille, une construction d'une approche est indispensable.

Dans cette section, nous nous intéressons à l'approche métaheuristique proposée et présentons les résutats d'experientation de l'algorithme du recherche taboue developpé et testé sur toutes les instances de la librairie "OR-Library".

Pour la validation de notre approche nous avons utilisé le logiciel LEKIN dont le fonctionnement est bien détaillé dans [42]. Page (537-544). Ce logiciel est téléchargé à partir de son site officiel http : //community.stern.nyu.edu/om/software/lekin/ et utilise principalement des règles de priorité et deux approches de résolution pour le problème d'ordonnancement.

5.3.1 Qu'est ce que le LEKIN

LEKIN est un système d'ordonnancement développé à la Stern School of Business de l'Université de New York.

Les principaux éléments du système ont été conçus et codés par des étudiants de l'Université Columbia.

LEKIN a été créé comme un outil éducatif avec pour principal objectif d'introduire les étudiants à la théorie de l'ordonnancement et de ses applications. En outre, l' extensibilité du système permet (et encourage) à utiliser dans le développement de l'algorithme. Le projet a été dirigé par le professeur Michael L. Pinedo, professeur Xiuli Chao et le professeur Joseph Leung. Cette évolution a été partiellement soutenu par la Foundation Nationale des Sciences.

Annonce : La version qui peut être téléchargée a été améliorée et permet de résoudre un problème d'ordonnancement de 50 jobs et 100 machines.

Les approches utilisées par LEKIN pour résoudre le problème d'ordonnancement $1/r_i/\sum_{i=1}^{n}\omega_i C_i$ et $1/r_i/\sum_{i=1}^{n}\omega_i T_i$ sont : General SB Routine et Shifting Bottelneck [2].

L'approche proposée nécessite un générateur développé pour générer des instances utilisables à la fois par l'application informatique développée pour l'approche et par le logiciel LEKIN.

Dans ce qui suit, nous nous intéressons à l'approche métaheuristique proposée et présentons les résutats d'experientation des trois exécutions de l'algorithmes du recherche tabou developpées et testées sur toutes les instances modifiées de la librairie OR-Library dont le nombre de tâche peut aller jusqu'à 100.

Les tableaux ci-dessous rapportent les résultats des expérimentations réalisées sur les instances testées pour le problème mon-objectif $1/r_i/\sum_{i=1}^{n}\omega_i C_i$ après pour le problème mono-objectif $1/r_i/\sum_{i=1}^{n}\omega_i T_i$, puis, sur le problème bi-objectif $1/r_i/\begin{cases}\sum_{i=1}^{n}\omega_i C_i \\ \sum_{i=1}^{n}\omega_i T_i\end{cases}$. On y trouve les valeurs exécution 1 , exécution 2 et exécution 3 représentant chacune le coût de la solution issus de trois lancements d'un même Benchmark, et à la fin de chaque ligne, on peut lire le temps maximum de résolution de ces trois exécutions.

Cette opération est refaite pour chaque voisinage adaptée swap, move et swap & move.

Notons que pour tous les Benchmarks testés l'écart maximum entre les solutions trouvées pour les trois lancements d'un même Benchmark est presque d'une unité. Il en découle que pour un benchmark donné, la meilleure solution représente la valeur minimale du coût pour les trois lancements.

Le problème monocritère : $1/r_i/\sum_{i=1}^{n}\omega_i C_i$

		Voisinage : swap/move				**Lekin**			
Instances	**Taille**	**Exécution (1)**	**Exécution (2)**	**Exécution (3)**	**C.P.U**	**GSB Routine**	**C.P.U**	**SB**	**C.P.U**
Wt40_1	40	217002	217999	216909	5.83	226343	6.00	222279	01
Wt40_2	40	169619	168593	171522	5.80	186626	09	177332	01
Wt40_3	40	203044	203737	204074	5.83	207958	04	217889	01

FIGURE 5.1 – Comparaison entre notre approche et les deux approches de LEKIN ($w.t-40$)

		Voisinage : swap/move				Lekin			
Instances	**Taille**	**Exécution (1)**	**Exécution (2)**	**Exécution (3)**	**C.P.U**	**GSB Routine**	**C.P.U**	**SB**	**C.P.U**
Wt50_1	50	402972	402352	402100	6.69	446019	18	432095	01
Wt50_2	50	319457	323988	317307	6.50	337347	17	382899	01
Wt50_3	50	343896	343564	343766	6.44	358850	12	372460	01

FIGURE 5.2 – Comparaison entre notre approche et les deux approches de LEKIN ($w.t-50$)

		Voisinage : swap/move			
Instances	**Taille**	**Exécution (1)**	**Exécution (2)**	**Exécution (3)**	**C.P.U**
Wt100_1	100	1430511	1433361	1427133	10.60
Wt100_2	100	1374515	1357801	1354442	10.51
Wt100_3	100	1188196	1182068	1177557	10.73

FIGURE 5.3 – Comparaison entre notre approche et les deux approches de LEKIN ($w.t-100$)

Le problème monocritère : $1/r_i/\sum_{i=1}^{n}\omega_i T_i$

		Voisinage : swap/move				Lekin			
Instances	**Taille**	**Exécution (1)**	**Exécution (2)**	**Exécution (3)**	**C.P.U**	**GSB Routine**	**C.P.U**	**SB**	**C.P.U**
Wt40_1	40	2429	3147	4035	5.80	4070	05	4863	01
Wt40_2	40	1633	1633	1633	5.74	1861	06	3969	01
Wt40_3	40	5193	7410	4345	5.94	4891	05	5979	01

FIGURE 5.4 – Comparaison entre notre approche et les deux approches de LEKIN ($w.t-40$)

		Voisinage : swap/move				Lekin			
Instances	**Taille**	**Exécution (1)**	**Exécution (2)**	**Exécution (3)**	**C.P.U**	**GSB Routine**	**C.P.U**	**SB**	**C.P.U**
Wt50_1	50	7896	8232	9200	6.59	13968	17	14279	01
Wt50_2	50	2605	4144	2610	6.83	2572	19	7082	01
Wt50_3	50	7816	6164	5656	6.77	8106	09	12187	01

FIGURE 5.5 – Comparaison entre notre approche et les deux approches de LEKIN ($w.t - 50$)

		Voisinage : swap/move			
Instances	**Taille**	**Exécution (1)**	**Exécution (2)**	**Exécution (3)**	**C.P.U**
Wt100_1	100	9947	12922	11757	15.16
Wt100_2	100	13991	44701	19148	10.90
Wt100_3	100	6712	10062	11557	10.85

FIGURE 5.6 – Comparaison entre notre approche et les deux approches de LEKIN ($w.t - 100$)

Bi-critère : Solutions non dominées

Obj1 : Somme pondérée des dates de fin d'exécution

Obj2 : Somme pondérée des retards absolus

n	Instances	Obj1	Obj2	
50	Wt50_4 **Exécution (1)**	60660,60662 60684,60697 60717,60744 60795	1931,1715 1651,1572 1500,1436 1407	7.50
50	Wt50_4 **Exécution (2)**	60639,60662 60675,60695 60717,60744 60802	1907,1715 1636,1564 1500,1436 1407	7.65
50	Wt50_4 **Exécution (3)**	60662,60675 60695,60702 60717,60768	1715,1636 1564,1564 1500,1471	7.50
50	Wt50_5 **Exécution (1)**	60639,60662 60671,60684 60697,60717 60744,60795	1907,1715 1715,1651 1572,1500 1436,1407	8.20
50	Wt50_5 **Exécution (2)**	60675,60697 60717,60744 60795,61317	1636,1572 1500,1436 1407,1400	7.34
50	Wt50_5 **Exécution (3)**	60675,60695 60717,60744	1636,1564 1500,1436	7.39

FIGURE 5.7 – Minimiser les deux objectifs pour n=50

Discussion

Une analyse plus approfondie sur les expérimentations de l'algorithme recherche taboue réalisé à partir des Benchmarks d'OR-Library, nous conduit à faire le constat que notre approche est plus performante que les autres de Lekin pour le problème mono-objectif. En effet, il donne de meilleurs résultats en terme de qualité des solutions retournées : 48 meilleures solutions parmi les 50 instances testées pour n=40 et 20 meilleures solutions parmi les 20 instances testées pour n=50, suivi par les 100 instances testées par notre approche, et enfin notre approche mène plus loin encore, à la résolution de problème bi-objectif.

Nous nous sommes aussi intéressé de calculer pour chaque instance, le minimum des coûts obtenus par les trois approches (GSB, SB et approche proposée). Ensuite, nous calculons pour chaque approche, l'erreur relative. Elle est par la formule.

$$\frac{Coût \quad de \quad la \quad solution - le \quad minimum \quad des \quad coût}{le \quad minimum \quad des \quad coût}$$

Dans le tableau ci-dessous nous précisons pour chaque instance : le coût de la solution (solution), son erreur relative (E.R) et le temps de résolution (C.P.U), tel que, E.R1 : représente le calcul de l'erreur relative à l'approche proposée, E.R2 : représente le calcul de l'erreur relative à GSB et E.R3 : représente le calcul de l'erreur relative à SB.

L'efficacité de l'approche est justifiée par trois repères.

i. Le nombre de meilleures solutions

ii. Le maximum de l'erreur relative

iii. La moyenne de l'erreur relative.

Minimiser $\sum_{i=1}^{n} \omega_i C_i$

	Voisinage : swap			Voisinage :move			Voisinage :swap/move		
Benchmark	E.R1	E.R2	E.R3	E.R1	E.R2	E.R3	E.R1	E.R2	E.R3
Wt40_1	0,019	0,018	0,000	0,000	0,049	0,030	0,000	0,043	0,025
Wt40_2	0,000	0,068	0,015	0,000	0,089	0,035	0,000	0,103	0,048
Wt40_3	44,951	41,519	0,000	40,629	41,519	0,000	40,649	41,519	0,000
Wt40_4	0,013	0,000	0,108	0,000	0,036	0,147	0,000	0,053	0,167
Wt40_5	0,000	0,014	0,057	0,000	0,032	0,076	0,000	0,041	0,085
Wt40_6	0,000	0,007	0,032	0,000	0,027	0,053	0,000	0,036	0,062
Wt40_7	0,000	0,062	0,079	0,000	0,094	0,111	0,000	0,104	0,122
Wt40_8	0,000	0,117	0,033	0,000	0,140	0,054	0,000	0,146	0,059
Wt40_9	0,000	0,042	0,092	0,000	0,041	0,091	0,000	0,046	0,096
Wt40_10	0,000	0,022	0,071	0,000	0,048	0,099	0,000	0,052	0,102
Wt40_11	0,000	0,117	0,182	0,000	0,114	0,179	0,000	0,128	0,194
Wt40_12	0,000	0,067	0,034	0,000	0,122	0,087	0,000	0,121	0,086
Wt40_13	0,000	0,240	0,185	0,000	0,248	0,193	0,000	0,257	0,202
Wt40_14	0,024	0,107	0,000	0,034	0,107	0,000	0,000	0,113	0,006
Wt40_15	0,000	0,128	0,091	0,000	0,113	0,077	0,000	0,120	0,083
Wt40_16	0,000	0,118	0,122	0,000	0,096	0,100	0,000	0,121	0,125
Wt40_17	0,000	0,098	0,018	0,006	0,079	0,000	0,000	0,106	0,025
Wt40_18	0,000	0,133	0,038	0,000	0,132	0,037	0,000	0,143	0,048
Wt40_19	0,000	0,131	0,075	0,000	0,138	0,082	0,000	0,142	0,086
Wt40_20	0,000	0,012	0,037	0,000	0,001	0,026	0,000	0,030	0,056
Wt40_21	0,000	0,006	0,005	0,000	0,006	0,005	0,000	0,006	0,005
Wt40_22	0,003	0,000	0,012	0,000	0,004	0,016	0,000	0,004	0,016
Wt40_23	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000
Wt40_24	0,000	0,001	0,006	0,000	0,001	0,006	0,000	0,001	0,006

FIGURE 5.8 – Calcul de l'erreur relative aux trois approches pour n=40 −Benchmark de 1 à 24 pour obj1

Wt40_25	0,000	0,002	0,000	0,000	0,002	0,000	0,000	0,002	0,000
Wt40_26	0,054	0,000	0,074	0,000	0,010	0,085	0,000	0,020	0,096
Wt40_27	0,000	0,030	0,183	0,000	0,030	0,182	0,000	0,037	0,190
Wt40_28	0,000	0,023	0,015	0,000	0,066	0,057	0,000	0,068	0,059
Wt40_29	0,031	0,000	0,161	0,000	0,025	0,189	0,000	0,027	0,192
Wt40_30	0,000	0,038	0,168	0,000	0,070	0,203	0,000	0,088	0,223
Wt40_31	0,000	0,080	0,165	0,000	0,093	0,180	0,000	0,095	0,181
Wt40_32	0,000	0,170	0,007	0,009	0,162	0,000	0,000	0,169	0,005
Wt40_33	0,000	0,062	0,111	0,000	0,089	0,139	0,000	0,099	0,149
Wt40_34	0,029	0,000	0,085	0,000	0,029	0,117	0,000	0,034	0,122
Wt40_35	0,000	0,064	0,062	0,000	0,067	0,065	0,000	0,071	0,069
Wt40_36	5,948	0,000	7,421	5,774	0,000	7,421	5,785	0,000	7,421
Wt40_37	0,000	0,057	0,146	0,000	0,045	0,134	0,000	0,055	0,145
Wt40_38	0,000	0,051	0,181	0,000	0,053	0,183	0,000	0,056	0,187
Wt40_39	0,000	0,022	0,171	0,000	0,041	0,194	0,000	0,045	0,199
Wt40_40	0,000	0,082	0,016	0,000	0,090	0,024	0,000	0,089	0,023
Wt40_41	0,000	0,073	0,023	0,000	0,062	0,013	0,000	0,074	0,024
Wt40_42	0,000	0,004	0,038	0,006	0,000	0,034	0,000	0,004	0,038
Wt40_43	0,000	0,068	0,024	0,000	0,079	0,034	0,000	0,076	0,032
Wt40_44	0,000	0,102	0,110	0,000	0,096	0,104	0,000	0,106	0,114
Wt40_45	0,000	0,082	0,017	0,000	0,073	0,009	0,000	0,093	0,027
Wt40_46	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000
Wt40_47	0,000	0,046	0,025	0,000	0,046	0,024	0,000	0,047	0,026
Wt40_48	0,000	0,069	0,032	0,000	0,108	0,070	0,000	0,110	0,071
Wt40_49	0,000	0,000	0,004	0,003	0,000	0,004	0,000	0,000	0,004
Wt40_50	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000

FIGURE 5.9 – Calcul de l'erreur relative aux trois approches pour n=40 −Benchmark de 25 à 50 pour obj1

Minimiser $\sum_{i=1}^{n} \omega_i T_i$

	Voisinage : swap			Voisinage :move			Voisinage :swap/move		
Benchmark	E.R1	E.R2	E.R3	E.R1	E.R2	E.R3	E.R1	E.R2	E.R3
Wt40_1	1,804	0,000	0,195	0,000	0,615	0,930	0,000	0,605	0,918
Wt40_2	0,000	0,166	1,487	0,000	0,209	1,579	0,000	0,209	1,579
Wt40_3	0,601	35,442	0,000	0,000	42,578	0,196	0,000	43,404	0,218
Wt40_4	0,652	0,000	0,515	0,000	0,363	1,066	0,000	0,472	1,231
Wt40_5	0,833	0,000	0,551	0,000	0,122	0,740	0,052	0,000	0,551
Wt40_6	0,000	0,171	0,303	0,000	0,464	0,630	0,000	0,252	0,394
Wt40_7	0,040	0,000	0,919	0,000	0,058	1,030	0,021	0,000	0,919
Wt40_8	0,000	0,114	0,584	0,000	0,232	0,751	0,000	0,133	0,611
Wt40_9	0,118	0,000	0,104	0,000	0,254	0,384	0,000	0,061	0,171
Wt40_10	0,000	0,135	0,359	0,000	0,439	0,723	0,000	0,441	0,724
Wt40_11	0,000	0,254	0,407	0,000	0,486	0,667	0,000	0,359	0,525
Wt40_12	0,000	0,181	0,178	0,000	0,383	0,379	0,000	0,371	0,367
Wt40_13	0,000	0,157	0,535	0,000	0,143	0,517	0,000	0,104	0,465
Wt40_14	0,000	0,000	0,020	0,000	0,059	0,080	0,024	0,000	0,020
Wt40_15	0,000	0,102	0,235	0,000	0,081	0,212	0,000	0,105	0,239
Wt40_16	0,000	0,009	0,190	0,000	0,023	0,206	0,000	0,025	0,209
Wt40_17	0,000	0,065	0,069	0,000	0,057	0,061	0,000	0,005	0,009
Wt40_18	0,000	0,045	0,154	0,000	0,043	0,152	0,000	0,013	0,119
Wt40_19	0,000	0,053	0,190	0,000	0,051	0,188	0,000	0,052	0,189
Wt40_20	0,008	0,000	0,166	0,000	0,006	0,173	0,000	0,006	0,174

FIGURE 5.10 – Calcul de l'erreur relative aux trois approches pour n=40 −Benchmark de 1 à 20 pour obj2

Wt40_21	0,000	0,006	0,006	0,000	0,006	0,006	0,000	0,005	0,005
Wt40_22	0,000	0,000	0,021	0,000	0,001	0,021	0,000	0,000	0,021
Wt40_23	0,000	0,000	0,001	0,000	0,000	0,001	0,000	0,000	0,001
Wt40_24	0,000	0,000	0,013	0,000	0,000	0,013	0,000	0,000	0,013
Wt40_25	0,000	0,000	0,008	0,000	0,000	0,008	0,000	0,000	0,008
Wt40_26	2,363	0,074	0,000	0,000	19,913	18,471	0,000	19,913	18,471
Wt40_27	15,535	0,000	20,257	0,000	0,000	20,257	0,000	0,000	20,257
Wt40_28	1,866	0,000	1,461	0,000	3,101	9,092	0,132	0,000	1,461
Wt40_29	7,454	0,000	0,774	0,000	7,565	14,194	0,000	2,136	4,563
Wt40_30	0,047	0,000	1,094	0,000	1,140	3,482	0,000	1,140	3,482
Wt40_31	0,000	0,279	1,185	0,000	0,285	1,196	0,000	0,285	1,196
Wt40_32	0,000	0,346	0,628	0,000	0,823	1,205	0,000	0,313	0,588
Wt40_33	0,386	0,000	0,385	0,000	0,059	0,467	0,000	0,065	0,474
Wt40_34	0,241	0,000	0,668	0,000	0,813	2,025	0,000	0,678	1,799
Wt40_35	0,425	0,000	1,496	0,000	0,085	1,708	0,000	0,093	1,728
Wt40_36	0,000	0,201	0,535	0,000	0,195	0,527	0,000	0,173	0,499
Wt40_37	0,009	0,000	0,471	0,000	0,034	0,521	0,000	0,028	0,511
Wt40_38	0,000	0,326	0,572	0,000	0,348	0,598	0,000	0,252	0,485
Wt40_39	0,119	0,000	0,315	0,000	0,111	0,461	0,000	0,104	0,452
Wt40_40	0,013	0,000	0,232	0,000	0,053	0,297	0,000	0,056	0,301
Wt40_41	0,000	0,079	0,252	0,000	0,080	0,253	0,000	0,070	0,241
Wt40_42	0,000	0,005	0,123	0,000	0,004	0,123	0,000	0,004	0,123
Wt40_43	0,000	0,052	0,181	0,000	0,056	0,185	0,000	0,048	0,176
Wt40_44	0,000	0,076	0,286	0,000	0,076	0,286	0,000	0,043	0,247
Wt40_45	0,000	0,116	0,154	0,000	0,116	0,154	0,000	0,111	0,149
Wt40_46	0,000	0,002	0,009	0,000	0,002	0,009	0,000	0,002	0,009
Wt40_47	0,000	0,036	0,038	0,000	0,036	0,038	0,000	0,036	0,037
Wt40_48	0,000	0,080	0,101	0,000	0,081	0,102	0,000	0,076	0,097
Wt40_49	0,000	0,008	0,009	0,000	0,008	0,009	0,000	0,004	0,006
Wt40_50	0,000	0,000	0,010	0,000	0,000	0,010	0,000	0,000	0,010

FIGURE 5.11 – Calcul de l'erreur relative aux trois approches pour n=40−Benchmark de 21 à 50 pour obj2

Minimiser $\sum_{i=1}^{n} \omega_i C_i$

	Voisinage : swap			Voisinage :move			Voisinage :swap/move		
Benchmark	E.R1	E.R2	E.R3	E.R1	E.R2	E.R3	E.R1	E.R2	E.R3
Wt50_1	0,000	0,099	0,064	0,000	0,090	0,056	0,000	0,108	0,073
Wt50_2	0,000	0,008	0,144	0,000	0,038	0,179	0,000	0,063	0,206
Wt50_3	0,000	0,004	0,042	0,000	0,040	0,080	0,000	0,045	0,084
Wt50_4	0,032	0,000	0,045	0,000	0,035	0,082	0,000	0,042	0,090
Wt50_5	0,000	0,015	0,103	0,000	0,025	0,114	0,000	0,026	0,115
Wt50_6	0,000	0,091	0,050	0,000	0,094	0,053	0,000	0,097	0,056
Wt50_7	0,000	0,035	0,042	0,000	0,065	0,072	0,000	0,092	0,099
Wt50_8	0,000	0,010	0,008	0,000	0,050	0,048	0,000	0,067	0,065
Wt50_9	0,000	0,017	0,122	0,000	0,004	0,108	0,000	0,053	0,162
Wt50_10	0,000	0,047	0,031	0,000	0,046	0,030	0,000	0,100	0,083
Wt50_11	0,000	0,093	0,051	0,000	0,064	0,023	0,000	0,097	0,055
Wt50_12	0,004	0,071	0,000	0,008	0,071	0,000	0,000	0,081	0,009
Wt50_13	0,000	0,144	0,094	0,000	0,139	0,089	0,000	0,163	0,112
Wt50_14	0,000	0,106	0,126	0,000	0,092	0,112	0,000	0,122	0,143
Wt50_15	0,000	0,046	0,062	0,000	0,048	0,064	0,000	0,053	0,069
Wt50_16	0,000	0,052	0,023	0,000	0,058	0,029	0,000	0,066	0,036
Wt50_17	0,000	0,115	0,021	0,000	0,147	0,051	0,000	0,147	0,051
Wt50_18	0,000	0,082	0,035	0,000	0,091	0,044	0,000	0,095	0,047
Wt50_19	0,000	0,102	0,092	0,000	0,099	0,089	0,000	0,111	0,101
Wt50_20	0,000	0,070	0,062	0,000	0,072	0,064	0,000	0,082	0,075

FIGURE 5.12 – Calcul de l'erreur relative aux trois approches pour n=50−Benchmark de 1 à 20 pour obj1

Minimiser $\sum_{i=1}^{n} \omega_i T_i$

	Voisinage : swap			Voisinage :move			Voisinage :swap/move		
Benchmark	E.R1	E.R2	E.R3	E.R1	E.R2	E.R3	E.R1	E.R2	E.R3
Wt50_1	0,148	0,000	0,022	0,000	0,641	0,677	0,000	0,943	0,986
Wt50_2	2,573	0,000	1,753	0,000	0,087	1,993	0,000	0,000	1,753
Wt50_3	0,210	0,000	0,503	0,000	0,380	1,075	0,000	0,359	1,043
Wt50_4	1,657	0,000	0,883	0,010	0,000	0,883	0,000	0,109	1,088
Wt50_5	0,733	0,000	1,903	0,000	0,052	2,053	0,000	0,080	2,135
Wt50_6	0,502	0,000	0,230	0,000	0,146	0,410	0,000	0,136	0,398
Wt50_7	0,033	0,000	0,014	0,000	0,016	0,030	0,000	0,211	0,227
Wt50_8	0,625	0,000	0,403	0,062	0,000	0,403	0,000	0,095	0,537
Wt50_9	0,000	0,178	0,845	0,000	0,351	1,116	0,000	0,434	1,246
Wt50_10	0,121	0,000	0,405	0,056	0,000	0,405	0,000	0,025	0,440
Wt50_11	0,066	0,000	0,185	0,000	0,019	0,208	0,000	0,059	0,255
Wt50_12	0,000	0,259	0,210	0,000	0,194	0,147	0,000	0,262	0,212
Wt50_13	0,000	0,171	0,253	0,000	0,159	0,240	0,000	0,251	0,339
Wt50_14	0,000	0,005	0,505	0,000	0,018	0,525	0,000	0,007	0,507
Wt50_15	0,000	0,162	0,273	0,000	0,187	0,300	0,000	0,186	0,299
Wt50_16	0,000	0,016	0,160	0,000	0,031	0,176	0,000	0,040	0,187
Wt50_17	0,000	0,108	0,129	0,000	0,098	0,119	0,000	0,098	0,119
Wt50_18	0,000	0,043	0,150	0,000	0,050	0,158	0,000	0,056	0,165
Wt50_19	0,000	0,082	0,191	0,000	0,068	0,176	0,000	0,084	0,193
Wt50_20	0,000	0,027	0,242	0,001	0,000	0,210	0,000	0,034	0,250

FIGURE 5.13 – Calcul de l'erreur relative aux trois approches pour n=50 −Benchmark de 1 à 20 pour obj2

Une approche donne une meilleure solution comparativement aux autres si son erreur relative est égale à zéro. Ainsi, nous remarquons que l'approche proposée a donnée pour minimiser la somme pondérée des dates de fin d'exécution et n=40 :
41/50 meilleures solutions (voisinage swap) et 43 meuilleures solutions (voisinage move)et 48/50 meuilleures solutions (voisinage swap/move), l'approche GSB a donnée 10/50 meilleures solutions (voisinage swap) et 6/50 meuilleures solutions (voisinage move) 5/50 meilleures solutions (voisinage swap/move) et l'approche SB Routine a donnée 7/50 meilleures solutions (voisinage swap)et 8/50 meilleures solutions (voisinage move) et 5/50 meilleures solutions (voisinage swap/move).
pour minimiser la somme pondérée des retards absolus et n=40 :
Notre approche donne 32/50 meilleures solutions (voisinage swap) et 50/50 meilleures solutions (voisinage move) et 47/50 meilleures solutions (voisinage swap/move).

l'approche GSB a donnée 22 meilleures solutions (voisinage swap) et 5/50 meuilleures solutions (voisinage move) 10/50 meilleures solutions (voisinage swap/move) et l'approche SB Routine a donnée 2 meilleures solutions (voisinage swap)et 0/50 meilleures solutions (voisinage move) et 0/50 meilleures solutions (voisinage swap/move).

Nous remarquons que l'approche proposée a donnée pour minimiser la somme pondérée des dates de fin d'exécution et n=50 :
18/20 meilleures solutions (voisinage swap) et 19/20 meuilleures solutions (voisinage move)et 20/20 meuilleures solutions (voisinage swap/move), l'approche GSB a donnée 1/20 meilleures solutions (voisinage swap) et 0/20 meuilleures solutions (voisinage move) 0/20 meilleures solutions (voisinage swap/move) et l'approche SB Routine a donnée 1/20 meilleures solutions

(voisinage swap)et 1/20 meilleures solutions (voisinage move) et 0/20 meilleures solutions (voisinage swap/move).

Nous remarquons que l'approche proposée a donnée pour minimiser la somme pondérée des retards absolus et n=50 :

10/20 meilleures solutions (voisinage swap) et 16/20 meuilleures solutions (voisinage move)et 20/20 meuilleures solutions (voisinage swap/move), l'approche GSB a donnée 10/20 meilleures solutions (voisinage swap) et 4/20 meuilleures solutions (voisinage move) 1/20 meilleures solutions (voisinage swap/move) et l'approche SB Routine a donnée 0/20 meilleures solutions (voisinage swap)et 0/20 meilleures solutions (voisinage move) et 0/20 meilleures solutions (voisinage swap/move).

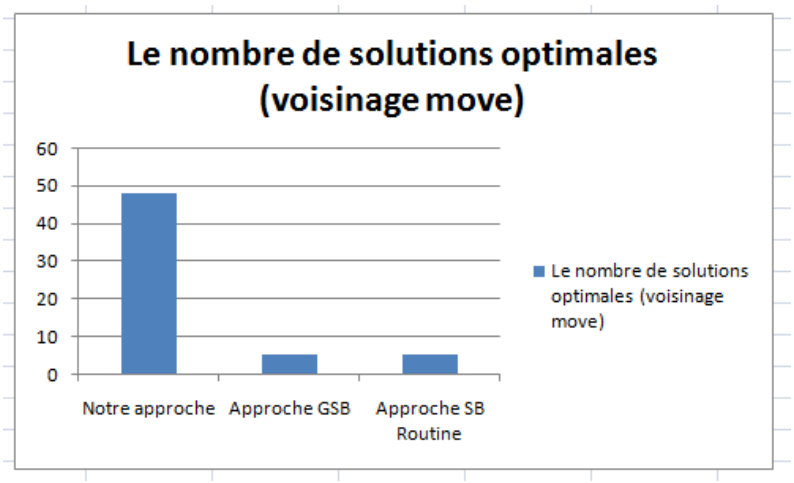


FIGURE 5.14 – Le nombre de solutions optimales pour minimiser obj1 (voisinage swap/move) pour n=40

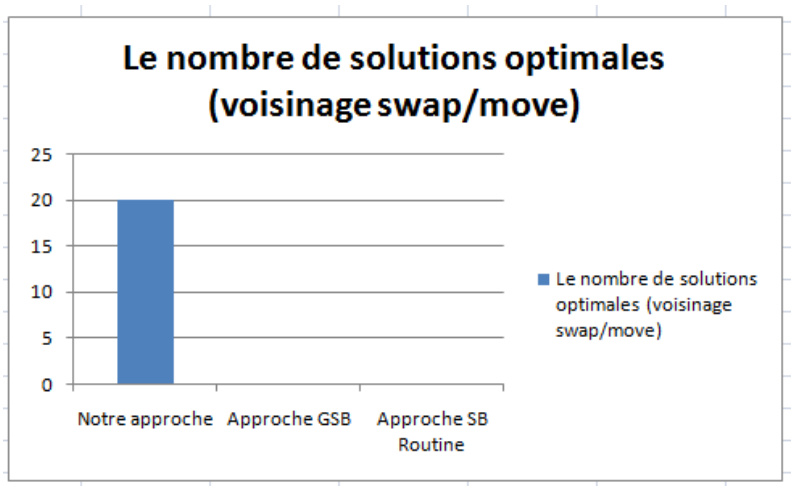


FIGURE 5.15 – Le nombre de solutions optimales pour minimiser obj1 (voisinage swap/move) pour n=50

Résultat

D'après les tests effectués, nous constatons que l'approche proposée est efficaces en nombre de meilleures solutions, maximum d'erreur relative et moyenne d'erreur relative, comparativement aux approches GB et SGB intégrées dans le logiciel LEKIN.

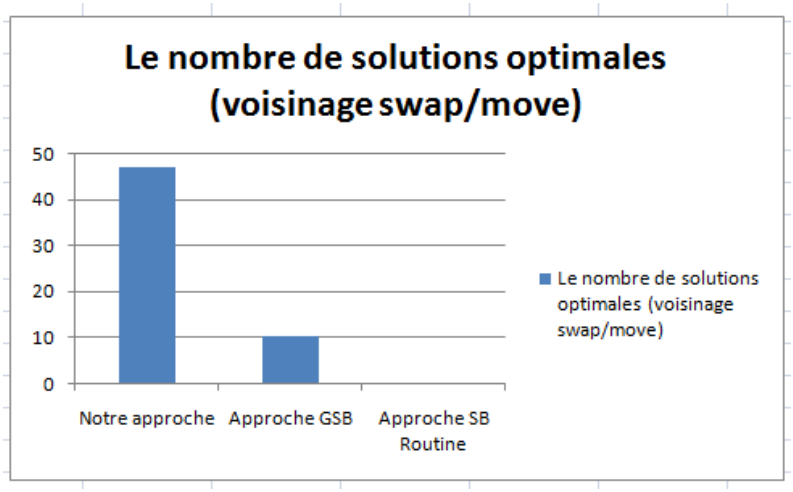


FIGURE 5.16 – Le nombre de solutions optimales pour minimiser obj2 (voisinage swap/move) pour n=40

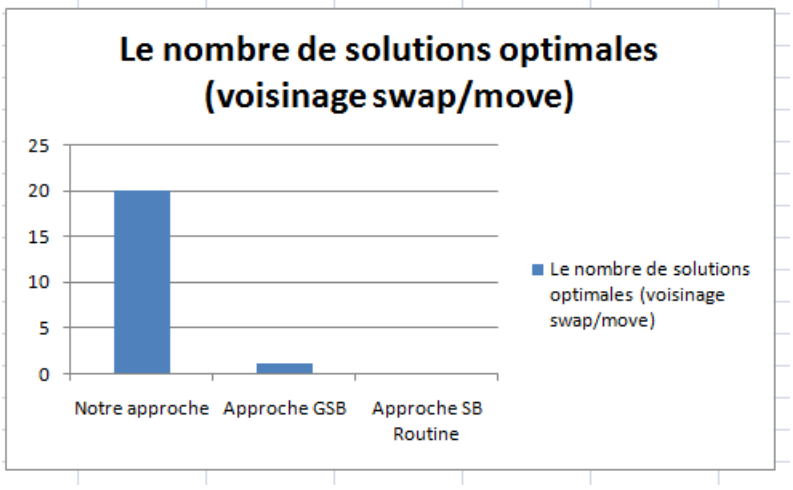


FIGURE 5.17 – Le nombre de solutions optimales pour minimiser obj2 (voisinage swap/move) pour n=50

5.4 Conclusion

Nous avons, dans ce chapitre, réalisé une série d'expérimentations afin d'évaluer la performance et de juger sur l'efficacité de l'algorithme du recherche taboue proposé pour la résolution du problème d'ordonnancement correspondant à la résolution du problème $1/r_i/ \begin{cases} \sum_{i=1}^{n} \omega_i C_i \\ \sum_{i=1}^{n} \omega_i T_i \end{cases}$

Pour la validation de l'approche métaheuristique de résolution, nous avons testé l'algorithme du recherche taboue sur les instances d'OR-Library adaptées au problème d'ordonnancement étudié. Les résultats des expérimentations prouvent l'efficacité de notre approche en termes de pourcentage de meilleures solutions retournées.

Signalons que le temps d'exécution de l'algorithme du recherche taboue étant étroitement lié à la durée d'exécution de la procédure d'évaluation d'une solution, cette durée dépend de la taille de l'instance.

A l'issu de ces résultats, nous concluons que la méthode de résolution proposée est efficace et la méthode de voisinage utilisée dans la littérature est performante.

Chapitre 6

Conclusion générale et perspectives

La naissance de l'optimisation multi-objectif remonte à un ouvrage de W. Pareto 1906 sur l'économie politique, dans lequel l'auteur définit pour la première fois ce qu'est un optimum multi-objectif. L'application de l'optimisation multi-objectif aux problèmes d'ingénierie remonte, aux, alentours de la seconde guerre mondiale, elle est longtemps restée une science « anecdotique » à cause de son aspect hermétique, dû à la nécessite de maîtriser un bagage mathématique théorique assez important. Un changement radical est apparu récemment avec les métaheuristique (recuit simule, algorithmes génétiques, etc.) qui sont des méthodes d'optimisation d'un abord plus facile. Deux approches de l'optimisation multiobjective s'affrontent :

– Résolution vectorielle du problème : on ne modifie pas l'expression du problème
– Transformation du problème en un problème d'optimisation monobjectif.

Aussi, le problème d'ordonnancement des ateliers de production figure parmi les problèmes d'optimisation les plus étudiés dans la littérature. Il est extrémement complexe, vu l'explosion combinatoire du nombre de solutions possibles, et est classé parmi les problèmes NP-difficile au sens fort. Ce problème reste NP-difficile au sens fort même dans le cas de machine unique.

Dans ce mémoire, nous nous sommes intéressés à l'étude d'un problème d'ordonnancement d'atelier à une machine. Ce problème est modélisé par un programme mathématique linéaire à variables mixtes (continues et binaires). La formulation regroupe principalement deux types de contraintes : les contraintes disjonctives , et les contraintes temporelles liées aux ressources. De plus, les contraintes étant disjonctives, nous les avons linéarisées.

Au vu de l'explosion combinatoire du nombre de solutions possibles qui croît exponentiellement avec la taille du problème d'optimisation, l'utilisation de la méthode exacte pour la résolution de modèle mathématique proposé, est de toute évidence limitée. Il en découle que le recours à une méthode approchée est nécessaire. Ainsi, nous avons proposé une approche basée sur l'algorithme du recherche taboue dans lequel nous avons distingué trois procédures pour la génération de la solution voisine.

Une série d'expérimentations a été réalisée sur des instances tirées de la littérature. Les résultats trouvés ont permis de juger sur l'efficacité de l'algorithme du recherche taboue développée.

Enfin, notre étude menée dans ce mémoire a touché à deux domaines de recherche, à savoir, l'ordonnancement d'atelier de production et le problème multi-critère. Ceci nous a permis de diversifier notre recherche ainsi que notre étude sur l'état de l'art. L'étude bibliographique nous a été bénéfique en termes d'idées et a permis de mener à terme le travail réalisé dans ce mémoire.

Elle nous a aussi inspiré, en plus du travail réalisé, pour mettre en perspective de nombreuses activités de recherche à mener dans le futur.

Nous soulevons ici quelques perspectives :

Dans ce même contexte, et en termes de travaux futurs, les pistes suivantes peuvent faire objet d'études :

a La minimisation de la somme des dates de fin d'exécution des travaux pour l'ordonnancement à machine unique avec des durées opératoires quelconques.

b La résolution des problèmes $1/r_i/\sum_{i=1}^{n}\omega_i C_i$ et $1/r_i/\sum_{i=1}^{n}\omega_i T_i$ et puis de $\begin{cases}\sum_{i=1}^{n}\omega_i C_i \\ \sum_{i=1}^{n}\omega_i T_i\end{cases}$ en utilisant une approche métaheuristique basée sur l'algorithme recherche taboue.

c Le développement d'un modèle mathématique pour la résolution des problèmes d'ordonnancement.

Cette piste permettrait ainsi de modéliser ces problèmes sans avoir recours aux contraintes disjonctives qui engendrent une explosion du nombre de cas à traiter séparément, lorsque le nombre de disjonction augmente.

Il serait aussi possible d'aborder le problème en considèrant le cas multi-objectif, et dans d'autre type de problèmes dordonnancement.

Bibliographie

[1] Al-Turki .U, Fedjki.C, Andijani. A, Tabu search for a class of single-machine scheduling problems, Department of Systems Engineering, King Fahd University of Petroleum and Minerals, Dhahran 31261, Saudi Arabia, Computers & Operations Research 28 (2001) 12231230.

[2] Asadathorn .N (1997) Scheduling of Assembly Type of Manufacturing Systems : Algorithms and Systems Development, Ph.D Thesis, Department of Industrial Engineering,New Jersey Institute of Technology, Newark, New Jersey (référencié par [2]).

[3] Baker. K.R. Introduction of sequencing and scheduling. Wiley NewYork, Etas Unis,1974.

[4] Baptiste P., Une étude théorique et expérimentale de la propagation des contraintes de ressources, Thèse de Doctorat, Université de Technologie de Compiègne, 1998. Barichard.V, " Approches Hybrides pour les problèmes multiobjectif ", Thèse de Doctorat,université d'Angres, Novembre 2003

[5] Ben Abdelaziz F. And Krichen S. (1997), A tabu search heuristic for multiple objective knapsack problems, Ructor Reasearch Report, RR 28-67.

[6] Ben Abdelaziz. F, Krichen. S,and Chouachi. J. Meta-heuristics : Advances and trends in local search paradigms for optimization,chapter A hybrid heuristic for multi-objective knapsack problems, pages 205-212, Kluwer Academic Publishers,1999.

[7] Carlier. J & P.Chrétienne. . Problèmes d'ordonnancement modélisation/complexité/algorithmes. Masson,1988

[8] Charon.I, Germa.A and Hurdy.O. Méthodes d'optimisation combinatoire, édition MASSON,1995.

[9] Chrétienne.P, Coffman.E.G and Lenstra ZLiu JK. Scheduling theory and its applications Wiley New York, Etats unis, 1995.

[10] Chu.C & Proth .JM. L'ordonnancement et ses applications. Masson, Paris, France, 1996

[11] Coello Coello.C.A, Christiansen.A.D, MOSES : A Multiobjective Optimization Tool for Engineering Design, Engineering Optimization, volume 31, numéro 3, pages 337-368, 1999. 1.5

[12] Collette.Y & Siarry.P,Optimisation multiobjectif : Algorithmes, ÉDITIONS EYROLLES 61, Bld Saint-Germain 75240 Paris Cedex 05, 2002, ISBN 2-212-11168-1.

[13] Congram. R.K, Potts .C.N and van de Velde .S.L (1998). An iterated dynasearch algorithm for the single-machine total weighted tardiness scheduling problem. In preparation.

[14] Crauwels. H.A.J, Potts. C.N and Van Wassenhove .L.N (1998). Local search heuristics for the single machine total weighted tardiness scheduling problem, Informs Journal on Computing 10, 341-350.

[15] Dahl .G, Dahl.G, Jornston. K and Lokketangen .A. A tabu search approach to the channel minimization problem. In G.Liu,K-H. Phua, J.Ma, J.Xu, F.Gu,and C.He,editors, Optimization-Techniques and Applications, ICOTA'95,volume1,page 369-377, Chengdu ,China 1995 .World Scientific.

[16] Deb.K, An overview of multi-objective evolutionary algorithms, Journée J.E.T., Copie de transparents, mai 1999. 1.6.2, 5.7.1.1.1, 5.7.2.1.1

[17] Deb.K, Multi-Objective Optimization using Evolutionary Algorithms, John Wiley Sons, 2001. 1.8.5, 5.7.2.3.3, 5.8

[18] Dewolf.D . Optimisation des flux, université du littoral cote d'Opale.2003

[19] Ehrgott.M, A characterization of Lexicographic Max-ordering Solutions, Methods of Multicriteria Decision Theory : Proceedings of the 6th Workshop of the DGOR Working Group Multicriteria and Decision Theory, Egelsbach, Häsel-Hohenhausen, pages 193-202, 1997. 1.8.2, 1.8.3, 1.8.4,1.13

[20] Ehrgott.M, Multicriteria Optimization, Lecture Notes in Economics and Mathematical Systems, numéro 491, Springer Verlag, 2000. 1.8.7, 2.13, 9.2.1, 9.2.2

[21] Eiben, Garzon. M. H, Honavar V, Jakiela .M, Smith .R. E (eds), Proceeding of the genetic and evolutionary Computation

[22] Esquirol.P & Lopez.L .L'ordonnancement. Economica, 1999.

[23] French.S . Sequencing and scheduling An introduction to the mathematics of the job-shopEllis Horwood, Chichester, 1982.

[24] Galinier.P ,Habib.M and J.K.Hao, Métha-heuristiques pour l'optimisation combinatoire et l'affectation sous contraintes, Revue d'intelligence artificielle. 1999.

[25] Gandibleux.X , Mezdaoui .N, and Freville .A. A Tabu search procedure to solve multi-objective combinatorial optimization problems. In R.Caballero, F.Ruiz, and R.Steuer, editors, Second Int. Conf. on Multi-Objective Programming and Goal Programming MOPGP'96,pages 291-300, Torremolinos, Spain, May 1996. Springer-Verlag.

[26] Garey.M & Johnson. D, Computer and Intractability : a Guide to the Theory of NP- Completeness. Freeman W-H, San Francisco, 1979.

[27] Glover. F. "Tabu search-Part I", ORSA Journal on Computing 1, 190-206, 1989

[28] Goldberg.D, Algorithmes génétiques, editions Addison Wesley, juin 1994.

[29] Gotha, Les problèmes d'ordonnancement. RAIRO.Operations Research, 27(1), 77-150.1993

[30] Halhal. D, Walters. G.A, Ouazar. D, and Savic. D.A. Water network rehabilitation with a structured messy genetic algorithm. Journal of Water Resources Planning and Management,123(3) : 137-146,1997.

[31] Hansen M.P, Metaheuristics for multiple objective combinatorial optimization, Ph.D. Thesis, IMM-PHS-1998-45, Technical University of Danmark, Lyngby.

[32] Hentoustous H., contribution au pilotage des systèmes de production de type Job Shop, Thèse de Doctorat, INSA Lyon, 1999.

[33] Hertz.A , Jaumard.B ,Ribeiro.C.C, and Formosinho Filho. W.P. A multi-criteria tabu search approach to cell formation problems in group technology with multiple objectives. RAIRO Recherche Opérationnelle \ Operations Research, 28(3) : 303-328,1994

[34] Iredi .S, Merkle. D and Middendorf. M, Bi-criterion optimization with multi colony algorithms, in : Proceeding of the First International Conference on Evolutionary Multi-Criterion Optimization. Lecture Notes in Computer Science , Springer, Berlin, 2001.

[35] Letouzey A., Ordonnancement interactif basé sur des indicateurs : Applications à la gestion de commandes incertaines et à l'affectation des opérateurs, Thèse de Doctorat, Institut National Polytechnique de Toulouse, 2001.

[36] Lopez.P & Roubelat.F Ordonnancement de la production. Hermes science publication, Paris, France, 2001

[37] Mariano .E & Morales.E , MOAQ and ant-Q algorithm for multiple objective optimization problems, in W. Banzhaf, J. Daida, A. E,1999.

[38] Miettinen. K. M, Nonlinear multiobjective optimization, éditions Kluwer academic publisher, 1999. 1.8.7, 1.13, 2.3.1, 2.4.1, 2.5.3, 2.7.3, 3.8, 9.1

[39] Mortan. TE & Pintico.D. Heuristics scheduling systems. Wiley, New York, Etats-Unis, 1993.

[40] Nessah. R & Kacem.I . Branch-and-bound method for minimizing the weighted completion time scheduling problem on a single machine with release dates. Computers and Operations Research.Volume 39, Issue 3, March 2012, Pages 471-478.

[41] Parks. G.T. & Miller. I. Selective breeding in multi-objective genetic algorithm. In Parallel Problem Solving from Nature PPSN'5, pages 250-259, Amsterdam, Sept 1998. Springer-Verkag.

[42] Pinedo. P, Scheduling Theory, Algorithms and Systems, 2008, Third Edition. Springer.

[43] Potts. C.N & Van Wassenhove. L.N, 1991. "Single machine tardiness sequencing heuristics", HE Transactions 23, 346-354.

[44] Sakarovitch.M, optimisation combinatoire, Herman, Paris,1984.

[45] Schmidt.K ,Using Tabu Search to Solve the Job Shop Scheduling Problem with Sequence Dependent Setup Times, May 18, 2001

[46] Serafini , Simulated annealing for multiple objective optimization problems, in : Proceeding of the Tenth International Conference on Multiple Criteria Decision Making, Taipei 19-24.07, vol. 1, 87-96,1992.

[47] Talbi.E, Métaheuristiques pour l'optmisation combinatoire multi-objective. Laboratoire d'Informatique Fondamentale de Lille, Université des Sciences et Technologies de Lille, Bat.M3 59655 Villeneuve d'Ascq,2001.

[48] T'kindt V. & Billaut J.-C., Multicriteria Scheduling Theory, Models and Algorithms, Translated from French by H. Scott, Second Edition, Springer-Verlag Berlin 2006.

[49] Van Veldhuizen.D.A, Multiobjective evolutionary algorithms : classifications, analyses and new innovations, Ph. D., Graduate School of Engineering. Air Force Institute of Technology, Wright Patterson AFB, Ohio, USA, janvier 1999. 1.12, 7.1, 7.10.1, 7.16

[50] $Xhafa.F\&Carretero.J^1$, $Dorronsoro.B^2$, $Alba.E^3$, A Tabu Search Algorithm For Scheduling Independent Jobs In Computational Grids,.(1)Department of Languages and Informatics Systems Polytechnic University of Catalonia, Spain,.(2)Faculty of Science, Technology and Communication University of Luxembourg, Luxembourg,.[3]Deptartment of Languages and Computer Science University of M´alaga, Spain Computing and Informatics, Vol. 28, 2009, 1001-1014, V 2009-Mar-2.

[51] Zhou. G & Gen. M. Genetic algorithm approach on multi-criteria minimum spanning tree problem. European Journal of Operational Research, 114 : 141-152,1999.