Résolution de problèmes d'ordonnancement multi-projet sous contraintes de ressources multi-compétences partagées

Meya Haroune

UNIVERSITÉ DE TOURS
UNIVERSITÉ DE NOUAKCHOTT AL-AASRIYA

ÉCOLE DOCTORALE MIPTIS

Laboratoire d'Informatique Fondamentale et Appliquée de Tours (EA 6300)

UNITÉ DE RECHERCHE CALCUL SCIENTIFIQUE, INFORMATIQUE ET SCIENCE DE DONNÉES

THÈSE présentée par :

Meya Haroune

soutenue le : 29-06-2022

pour obtenir le grade de Docteur de l'Université de Tours
et de l'Université de Nouakchott Al-Aasriya

Discipline/Spécialité : Informatique

Résolution de problèmes d'ordonnancement multi-projet sous contraintes de ressources multi-compétences partagées

Thèse dirigée par :

Néron Emmanuel	Professeur, Université de Tours
Nanne Mohamedade Farouk	Professeur, Université de Nouakchott Al-Aasriya

Rapporteurs :

Jouglet Antoine	Professeur, Université de Technologie de Compiègne
Oulamara Ammar	Professeur, Université de Lorraine

Jury :

Bellenguez Odile	Maître assistant, HdR, IMT Atlantique
Della Croce Di Dojola Federico	Professeur, École Polytechnique de Turin
Haouba Ahmedou	Professeur, Université de Nouakchott Al-Aasriya
Jouglet Antoine	Professeur, Université de Technologie de Compiègne
Néron Emmanuel	Professeur, Université de Tours
Nanne Farouk Mohamedade	Professeur, Université de Nouakchott Al-Aasriya
Oulamara Ammar	Professeur, Université de Lorraine
Soukhal Ameur	Professeur, Université de Tours, Co-Directeur

Remerciements

Après avoir terminé ce travail de recherche, il ne reste que d'exprimer et avec plaisir mes sincères remerciements et ma reconnaissance à toutes les personnes qui ont contribué à l'élaboration de ma thèse.

Tout d'abord, je tiens à remercier très chaleureusement mes directeurs de thèse Emmanuel Néron et Mohamedade Farouk Nanne. Emmanuel d'abord, pour le temps conséquent qu'il m'a accordé, ses orientations, ses idées intéressantes, sa franchise et sa sympathie. J'ai beaucoup appris à ses côtés et je lui adresse ma gratitude pour tout cela. Farouk ensuite, qui m'a posé la question deux jours après la soutenance de mon projet de master "ça te dit de faire une thèse qui sera en cotutelle entre deux universités". Je lui remercie donc de m'avoir attiré dans cette longue et riche expérience.

J'adresse mes sincères remerciements à mes encadrants de thèse Ameur Soukhal, Cheikh Dhib et Hafed Mohamed Babou. Ameur Soukhal pour ses idées et ses conseils qui m'ont été très utiles. Je le remercie également pour les dizaines d'heures qu'il a passées à lire et corriger mes documents de recherche. Cheikh avec qui j'ai développé une relation de confiance. En plus de la rare profondeur de son esprit scientifique, j'ai découvert quelqu'un de sincère et d'intègre, des qualités extrêmement rares et précieuses. J'aurai difficilement pu terminer ma thèse dans de bonnes conditions sans l'aide de Cheik Dhib. Hafed pour son attention et sa gentillesse tout au long de la période de ma thèse. Son énergie et sa confiance ont été des éléments moteurs pour moi. J'ai pris un grand plaisir à travailler avec lui.

Je tiens à remercier très sincèrement les membres de mon jury de thèse, pour l'honneur qu'ils m'ont fait. En premier lien, son président Ahmedou Haouba qui a bien dirigé les discussions. Mes rapporteurs, Antoine Jouglet pour avoir accepté de faire partie de mon comité de suivie de thèse et pour la grande attention qu'il a porté à la lecture de ce document. Ammar Oulamara pour ses questions pertinentes et ses conseils qui ouvriront de nouvelles perspectives pour ma thèse. En fin mes examinateurs, Federico Della Croce Di Dojola et Odile Bellenguez pour avoir accepté de faire parti de mon jury de thèse et pour leurs remarques constructives. Je commence tout juste à réaliser la chance que j'ai eu d'avoir ce jury.

Mes remerciements vont également à l'ensemble du personnel du SCAC (Service de coopération et d'action culturelle) de l'Ambassade de France en Mauritanie pour leur investissement dans ce travail et le travail qu'ils effectuent dans l'ombre avec efficacité et une attention et une écoute toute particulière.

Comment ne pas remercier l'ensemble du personnel de la société Alfa Conseil dirigé par son directeur Oumar Bellal pour leur accueil chaleureux et l'environnement professionnel qui s'est ouvert à moi lors de mes séjours en Mauritanie.

Je remercie les différents permanents du LIFAT : Hubert Cardot, directeur du laboratoire, Yannick Kergosien directeur de l'équipe ROOT, et tous les autres. Je remercie le personnel de l'école doctorale : Nils Erglund, Emmanuel Humbert, Bénédite Richard et Isabelle Foulon.

Je tiens à remercie profondément les différents permanents de EDST et des unités de recherche

URPIMA et CSIDS : Mohamed Vall Mohamed Abdallahi, directeur de l'école doctorale EDST, Mohamed Jebril, directeur de l'unité de recherche URPIMA, Mohamed Beddi, responsable de la formation doctorale. Je remercie tout particulièrement Mohamedou ndongo pour les efforts qu'il a déployé afin de résoudre mes problèmes d'inscription à l'UNA.

Mes sincères remerciements vont également aux personnels de Supnum, ESP et ISCAE avec qui j'ai eu la chance et le plaisir de travailler comme vacataire cette année. Moussa Demba, Mamadou Tourad Diallo, Sidi Biha, Hafed Mohamed Babou. Il m'est presque impossible de décrire les compétences et les qualités pédagogiques que j'ai acquises en travaillant avec vous.

Je remercier maintenant mes collègues doctorants et ceux qui sont déjà docteurs. Mes anciens colocataires d'abord, Diem, Hugo et Laura merci pour les bons moments passés avec vous. Un grand merci à David, Alexis, Limeme, Lemine, Boukhalfa, Mustapha, Minh, Laura, Diem et Hugo (les mêmes), Romain, Guillaume, Valentine pour toutes les matchs de baby-foot et tout le plaisir que j'ai eu à vivre à Tours.

Je souhaite remercier mes collègues doctorants et docteurs de l'UNA : Mariem, Salema, Labouda, Soukeina, Fatimetou Abdou et Fatimetou Khiddi.

Bien sûr je n'oublie pas non plus de remercier ma famille qui ont toujours été à mes côtés pour me supporter et m'encourager pendant mes études.

Enfin, je tenais à exprimer sincèrement toute ma reconnaissance, à toutes les autres personnes qui, même sans être citées distinctement dans ce présent mémoire, ont contribué de près ou de loin, à en garantir son aboutissement et sa réussite.

Je dédie ce modeste travail à la mémoire de mon meilleur professeur, mon meilleur ami, mon héros, celui qui été toujours là pour moi, à mon PÈRE. Papa, je l'ai fait, je suis devenue docteur comme tu l'as toujours voulu. Je le fais pour toi, pour t'honorer, comme tu as toujours tout fait pour moi...

Je dédie ce travail également à la mémoire de ma MÈRE. MAMAN, bien que le temps que tu es restée avec nous ait été très court, l'amour que j'ai pour toi est éternel. Je sais qu'un jour on se réunira tous ensemble dans une vie meilleure. À ce moment là, j'aurai tous les moments "mère-fille" que j'ai imaginé avoir avec toi.

Chapitre 1

Résumé

Dans cette thèse, nous considérons un ensemble de problèmes d'ordonnancement multi-projet à contraintes de ressources multi-compétences limitées. Les problèmes étudiés sont issus d'un cas industriel rencontré par une entreprise de développement de services informatiques. Dans chacun de ces problèmes, de multiples projets doivent être exécutés simultanément et achevés dans un horizon de planification fixe. Chaque projet est décomposé en un ensemble de tâches préemptives avec des dates de début au plus tôt et de fin souhaitées associées aux tâches. Chaque tâche doit être exécutée par une seule ressources possédant plusieurs compétences et un niveau d'efficacité par compétence. En outre, la durée de la tâche peut être réduite en fonction du niveau d'efficacité de la ressource affectée à cette tâche. Les ressources sont un ensemble d'employés multi-compétences de disponibilité limitée. Pour chaque employé, nous avons une disponibilité hebdomadaire connue au début de l'ordonnancement et qui n'est pas sujette au changement. De plus, un employé peut intervenir dans plusieurs projets avec un taux de participation maximum (quotité) par projet.

Deux variantes de ce modèle sont abordées : une variante à un seul agent avec un seul critère à optimiser ; une variante avec plusieurs agents où chacun cherche à optimiser son critère. Dans la première variante, les quotités des employés sur les projets sont fixées par le décideur. Ainsi, les disponibilités hebdomadaires des employés, sont réparties entre les différents projets au prorata de leurs quotités sur les projets. L'objectif est d'affecter les employés aux tâches des projets de manière à minimiser la somme des retards pondérés des tâches plus la somme pondérée des violations de certaines contraintes. Pour une résolution à l'optimum, nous présentons un modèle mathématique basé sur la programmation par objectif mixte en nombres entiers. Pour résoudre des instances de grandes tailles, nous proposons une heuristique gloutonne, une recherche locale et une recherche tabou. Nous comparons les performances de ces méthodes par rapport au modèle mathématique en utilisant des instances dérivées de données réelles données par l'entreprise partenaire.

Dans la seconde variante, nous étendons le modèle précédent pour inclure le cas où les quotités des employés sur les projets ne sont pas fixées. Nous considérons que ces quotités doivent être déterminées par le calcul d'ordonnancement, en faisant intervenir le critère de chaque agent. Deux problèmes d'ordonnancement à deux ou trois agents ont été abordés. Le premier problème est de type optimisation sous contrainte avec deux agents en compétition. Le deuxième problème s'agit d'un problème d'ordonnancement à deux agents avec une fonction objectif globale (agent global). Il ajoute au premier problème le faite que certaines contraintes peuvent être violées lorsqu'il n'est pas possible de trouver une solution réalisable pour le premier problème ou pour diminuer le coût du retard. Ainsi, un objectif global consistant à minimiser la violation de ces contraintes est ajouté. Afin de résoudre les problèmes étudies, nous présentons deux modèles mathématiques basés sur la programmation linéaire. Ces modèles sont utilisés chacun par l'approche ε-contrainte pour déterminer les fronts de Pareto exacts. Cependant, cette méthode n'est efficace que pour résoudre les

instances de petites tailles. Pour résoudre les instances de grandes tailles, des approches heuristiques utilisant une méthode génétique de type NSGA-II sont développées. Ces différentes approches ont été comparées en termes de qualité de solution sur des instances de taille réelle.

Abstract

In this thesis, we consider a set of multi-project scheduling and multi-skilled employees assignment problem with hard and soft constraints. The studied problems stem from an industrial case in an IT company. In each of those problems, multiple projects must be executed simultaneously and completed within a fixed planning horizon. Each project is broken down into a set of preemptive tasks with release and due dates associated with the tasks. We consider that every task needs exactly one skill and must be performed by one employee who has the corresponding skill with an efficiency level per skill. Thus, the processing time of the task may be reduced according to the efficiency level of the employee assigned to this task. Every employee has an availability per week (a working time) known in advance. Also, an employee may be involved in more than one project at the same time, with a maximum quota, i.e. percentage of his/her time, allotted to each project.

Two variants of this model are tackled : a variant with one agent and one criterion to optimize ; a variant with several agents where each agent desires to optimize his own criterion. In the first variant, the decision-maker fixes the quotas of the employees on the projects. Thus, the weekly availabilities of the employees are shared between the different projects in proportion to their quota on the projects. The objective is to assign employees to project tasks such that the total weighted tardiness of tasks and the undesirable goal deviations are minimized. For this problem, we present a mixed-integer goal programming (MIGP) formulation to produce an optimal schedule. Furthermore, a local search algorithm and a tabu search algorithm are proposed to tackle large-scale instances. We compare the performance of the heuristic algorithms against the corresponding MIGP with simulated instances derived from real-world instances got from the partner company.

In the second variant, we extend the previous model to include the case where the quotas of employees on projects are not known. We consider that these quotas must be determined by the scheduling computation, taking into account the agents' critaria. Each agent manages one or several projects and competes with another agent for the use of common multi-skilled employees. We address two scheduling problems with two or three agents and multi-skill constrained-resources. In the first problem, the objective is to minimize the criterion of each compiting agent. The second problem is a two compiting agent scheduling problem with a global objective function (global agent). It also considers that some constraints can be violated when there is no feasible schedule for the first problem or to reduce the cost of delay of each agent. Thus, we further consider a global objective function that minimizes the soft constraint violations. The objectives of each agent is to minimize the total weighted tardiness of its tasks. The overall objective is to find a schedule that minimizes both agents objective functions and the global objective function. In each of these problems. For these problems, we provide two mathematical models based on mixed integer linear programming (MILP). These models are each used by the ε-constraint method to determine the pareto-optimal frontier. However, this method is only effective for solving small instances. To solve large instances, we provide two hybrid heuristics based on a genetic method of type NSGA-II algorithm. We compare the performance of the hybrid heuristics with simulated instances derived from real-world instances.

Table des matières

Table des figures

Liste des tableaux

Introduction générale

Au sein des entreprises, le développement de méthodes de planification de projets acquiert une dimension stratégique de plus en plus importante. En effet, afin de pouvoir répondre au mieux aux demandes des clients, les entreprises cherchent à réaliser leurs projets avec une qualité de service garantie, en respectant les délais contractuels et en minimisant le coût de réalisation. Afin d'atteindre ces objectifs, l'optimisation de leurs ressources humaines, matérielles et financières est primordiale. C'est dans ce cadre que se situent les problèmes d'ordonnancement de projets.

L'ordonnancement de projet est un processus qui vise à organiser un enchaînement des opérations (ou tâches) de projet dans le temps, tout en optimisant un ou plusieurs critères. Il consiste également à désigner une date de début, une date de fin prévue et les ressources allouées à chaque tâche.

L'ordonnancement des projets informatiques est généralement un problème combinatoire complexe par nature. Cette complexité est justifiée par des chercheurs comme Chen et al. Chen et al. (2017) par deux principales raisons. Premièrement, la grande diversité des employés techniques qui peuvent être impliqués dans le processus de développement des projets : développeurs, concepteurs, administrateurs de bases de données, analystes, etc. Deuxièmement, ces employés possèdent souvent des compétences multiples et variées, avec des niveaux d'efficacité différents. Par exemple, un développeur peut maîtriser différents langages de programmation, tels que C++, python, C# ; et peut également avoir en parallèle plusieurs compétences non techniques, telles que la maintenance de systèmes logiciels ou la correction des bugs. En outre, plusieurs employés peuvent avoir la compétence requise par une tâche. Par conséquent, le problème n'est pas seulement de choisir les employés spécifiques qui seront en charge de chaque tâche, mais aussi les compétences avec lesquelles ils contribueront. Il s'agit d'un type de décision supplémentaire par rapport au célèbre problème d'ordonnancement de projet sous contraintes des ressources (RCPSP) Schwindt and Paetz (2015) ; Schwindt and Zimmermann (2015).

Étant donné les facteurs précédents, le problème devient de plus en plus compliqué lorsqu'il s'agit de plusieurs projets simultanés. En effet, les entreprises informatiques travaillent souvent sur plusieurs projets en parallèle qui partagent le même pool d'employés multi-compétences. Par conséquent, la principale préoccupation du gestionnaire dans une telle situation est de répartir la disponibilité des employés entre les projets, tout en respectant diverses contraintes et en optimisant un ou plusieurs critères spécifiques. Ce type de problèmes d'ordonnancement est appelé par Néron et al Néron and Baptista (2002) "ordonnancement multi-projet sous contraintes des ressources multi-compétences".

Cette thèse fait l'objet d'une convention de cotutelle entre l'université de Tours et l'université de Nouakchott Al-Aasriya. Il résulte d'une collaboration entre trois parties. Tout d'abord, le Laboratoire d'Informatique Fondamentale et Appliquée de Tours (LIFAT -EA6300), à travers l'équipe de Recherche Opérationnelle : Ordonnancement, Transport (ROOT, CNRS ERL 7002). Ensuite, l'unité de recherche Calcul Scientifique, Informatique et Science de Données (CSISD) de l'Université de Nouakchott. Enfin, la société Alfa Conseil fondée en 2013 à Nouakchott.

La société Alfa conseil opère dans le domaine de technologies de l'information et sa principale préoccupation est le développement de projets informatiques. En effet, elle met à disposition sa structure et les compétences de ses équipes humaines pour mener à bien ses projets. Son objectif est de gérer ses projets pour répondre au mieux aux besoins de ses clients tout en respectant leurs contraintes budgétaires.

Alfa conseil est confrontée à un cas particulier de problèmes d'ordonnancement de projet. Au sein de cette entreprise, plusieurs chefs de projet ont accès aux même compétences humaines nécessaires à l'achèvement de leurs projets et chaque chef de projet est responsable d'un ou plusieurs projets. D'une manière très générale, le problème consiste à ordonnancer simultanément de multiples projets, à affecter les ressources humaines nécessaires aux tâches de chaque projet et à garantir l'achèvement de ces projets dans un délai déterminé. Les ressources sont constituées par des équipes d'employés dont la disponibilité limitée et la capacité limitée, ainsi que les compétences variées sont nécessaires à la réalisation des projets. De plus, les engagements contractualisés imposent le respect des dates de fin des phases des projets. Le cas échéant, nous cherchons à minimiser la somme des retards pondérés de l'ensemble des tâches de tous les projets permettant par conséquent de réduire les paiements des pénalités.

Notre travail consiste au développement de méthodes d'aide à la décision capables de proposer aux décideurs une affectation d'employés à chaque projet, un enchaînement des tâches dans le temps, tout en respectant un ensemble de contraintes spécifiques telles que la charge réalisée de chaque tâche par semaine et les compétences des ressources et leur disponibilité. Toutes les contraintes et les objectifs à atteindre ont été discutés avec les responsables des projets au sein de l'entreprise. Les modèles étudiés prennent en considération des contraintes dures et des contraintes souples liées aux tâches et aux ressources. Ce problème et plus généralement les problèmes d'ordonnancement de projets sont difficiles du point de vue théorique et nécessitent des méthodes informatiques dédiées.

Lorsqu'il s'agit d'un problème d'ordonnancement de projet, on s'attend à ce qu'il existe des relations de précédence entre les tâches. Ces contraintes donnent un ordre (au moins partiel) d'exécution des tâches. Cependant, les relations de précédence ne sont pas toujours pertinentes dans un contexte de développement logiciel pour deux raisons principales. La première raison est qu'il est assez difficile de définir clairement les relations de précédence dans le cadre d'un projet de développement logiciel. Comme le produit final du projet de développement logiciel - une application en cours d'exécution - et sa progression ne sont pas facilement visibles, il peut être très difficile pour le gestionnaire de visualiser et d'évaluer l'état du projet. La deuxième raison est que la structure d'un projet de développement logiciel est souvent basée sur l'un des modèles de conception (par exemple, MVC pour Model View Controller) qui aident à concevoir des systèmes faiblement couplés. Par conséquent, le développeur de logiciels a le choix total de décider ce qu'il doit développer en premier. Pour ces deux raisons, les gestionnaires de projets logiciels sont rarement intéressés par les détails de qui fait quoi et dans quel ordre. Leur préoccupation principale est essentiellement centrée sur les dates de sortie, les jalons et les objectifs des phases du projet, ainsi que sur les effectifs globaux. L'une des stratégies souvent employées pour représenter les projets de développement logiciel consiste à utiliser une modélisation à gros grain Mei et al. (2005).

Les modèles à gros grains se caractérisent principalement par leur flexibilité, permettant de mettre à jour le planning du projet autant de fois que nécessaire. Dans notre cas, l'entreprise partenaire simule le comportement de son système à l'aide de sa représentation à gros grain. Cependant, cette représentation ne considère pas les relations de précédence entre les tâches à un niveau très détaillé. Seules certaines relations de précédence flexibles entre les phases du projet sont connues et sont contournées par des dates de débuts au plus tôt et dates de fin souhaitées associées aux tâches. Par exemple, un chef de projet peut espérer que la tâche n_j précède la tâche n_{j+1} en attribuant à n_{j+1} une date de début au plut tôt supérieure à la date de fin souhaitée de la tâche n_j. Cependant, il accepte que n_{j+1} commence avant la fin de n_j si cette dernière tâche est

en retard

Les travaux de cette thèse s'organisent selon les deux parties suivantes. La première partie s'intéresse à un problème d'ordonnancement multi-projet et d'affectation de ressources multi-compétences aux tâches des projets. Chaque projet est décomposé en un ensemble de tâches préemptives avec des dates de début et de fin souhaitées, sans contraintes de précédence explicites. Chaque tâche doit être exécutée par un seul employé possédant plusieurs compétences et un niveau d'efficacité par compétence. De plus, ce niveau d'efficacité est pris en compte dans les durées de réalisation des tâches. En d'autre termes, la durée de la tâche peut être réduite selon le niveau d'efficacité de l'employé affecté à cette tâche. Un employé peut intervenir dans divers projets qui s'exécutent simultanément. De plus, pour chaque employé nous avons une disponibilité hebdomadaire (temps de travail) connue au début de l'ordonnancement et qui n'est pas sujette au changement. Dans cette partie, nous supposons que les allocations des employés aux projets sont connues en avance. Nous supposons également que les taux de participation (ou quotités) des employés aux projets sont fixés par le décideur. Ce qui veut dire que pour chaque projet, nous savons quels employés doivent le mener à bien, ainsi que le temps maximum qu'ils peuvent y consacrer durant chaque semaine de planification. Les employés disponibles chaque semaine sont répartis entre les différents projets au prorata de leurs quotités sur chaque projet. L'objectif est d'affecter les employés aux tâches des projets, tout en minimisant la somme des retards pondérés ainsi que les violations de certaines contraintes molles.

Dans cette première partie, nous sommes confrontés à deux problèmes. Le premier est de trouver une affectation compatible des employés aux tâches des projets. Le second problème est de déterminer un ordonnancement des tâches que chaque employés doit accomplir dans l'horizon de planification.

La seconde partie de nos travaux s'intéresse à deux problèmes d'ordonnancement multi-agent et d'allocation des ressources humaines multi-compétences aux projets.

Dans le premier problème d'ordonnancement de projet multi-agent étudié, l'ensemble des tâches à réaliser constituent deux sous-ensembles disjoints. Les agents (chaque sous-ensemble de tâches) sont en compétition sur l'utilisation des ressources humaines multi-compétences. Chaque agent cherche à optimiser une fonction objectif appliquée uniquement sur ses tâches. De plus, dans cette deuxième partie du manuscrit, les quotités des employés sur les projets ne sont pas fixées par avance. Pour se faire, on considère que les chefs des projets (agents) doivent collaborer pour négocier l'utilisation des compétences requises par leurs projets. Ce modèle est donc un cas plus général du modèle abordé dans la première partie du manuscrit.

Pour le deuxième problème étudié, nous introduisons un troisième agent, dit agent global. Il ajoute au problème précédent le faite que certaines contraintes (dites contraintes souples) peuvent être violées lorsqu'il n'est pas possible de trouver une solution réalisable ou pour diminuer les coûts des retards de chaque agent. Ainsi, un objectif global consistant à minimiser la violation de ces contraintes pour l'exécution de l'ensemble total des tâches est ajouté.

Pour l'ensemble des problèmes étudiés, nous développons des méthodes exactes et approchées afin de déterminer les "meilleurs" front de Pareto.

Pour exposer nous travaux, la suite de ce manuscrit est composée de deux parties et d'une conclusion générale avec les perspectives de nos travaux :

– La première partie (Chapitres 2, 3 et 4) s'intéresse à un problème d'ordonnancement multi-projet à contraintes de ressources multi-compétences. Le Chapitre 2 introduit le contexte des problèmes d'ordonnancement de projet abordés dans cette partie de thèse. Puis, le Chapitre 3 établit un état de l'art approfondi des travaux en lien avec notre sujet. À savoir, problèmes d'ordonnancement de projets à contraintes de ressources limitées, problèmes d'ordonnancement à contraintes de ressources multi-compétences et problèmes d'ordonnancement multi-projet. Elle s'attarde ainsi sur les différentes approches de résolutions exactes et heuristiques

proposées dans la littérature pour résoudre ses problèmes. Enfin, le Chapitre 4 présente plus en détail le problème traité dans cette partie. Il présente également les méthodes de résolution exactes et approchées que nous proposons pour résoudre ce problème.

– La deuxième partie (chapitres 5, 6 et 7) de ce travail s'intéresse à des problèmes d'ordonnancement multi-agent et d'allocation des ressources multi-compétences. Le Chapitre 5 porte sur les notions de base associées aux problèmes d'ordonnancement multi-agent. Le Chapitre 6 dresse un état de l'art de problèmes d'ordonnancement multi-agent. Le dernier chapitre de cette partie (chapitre 7) présente les deux problèmes étudiés ainsi que les différentes méthodes de résolution proposées.

– Enfin, nous clôturons ce manuscrit par des conclusions de nos travaux suivies par des perspectives envisageables à court et moyen termes.

Première partie

Ordonnancement de projet sous contraintes de ressources limitées

Chapitre 2

Présentation du cadre de travail

Contents

Ce chapitre a pour but de présenter succinctement le contexte du problème d'ordonnancement de projet abordé dans cette partie de thèse. La section 2.1 présentera quelques notions introductives nécessaires à la compréhension du manuscrit. Ensuite, la section 2.2 discutera des différentes méthodes de résolution appliquées dans la littérature. Elle sera suivie par la section 2.3 qui clôtura ce chapitre.

2.1 Les problèmes d'ordonnancement : notions préliminaires

Nous donnons dans cette section quelques définitions qui vont nous permettre de mieux appréhender les problèmes d'ordonnancement de projet.

2.1.1 Introduction

Un projet est défini comme une séquence d'opérations qui doivent être accomplies avec des moyens donnés pour atteindre un résultat fixé. Il est défini et mis en œuvre pour élaborer une réponse au besoin d'un utilisateur, d'un client ou d'une clientèle Galvagnon (2000).

À première vue, l'ordonnancement de projet est un processus complexe visant à organiser de bout en bout le bon déroulement d'un projet. D'une importance capitale, il consiste en effet à organiser, à assurer et à optimiser l'ensemble des moyens nécessaires à la réalisation du projet. L'ordonnancement de projet aboutit généralement à des outils qui seront utilisés par le décideur

pour la prise de décisions. L'objectif est de désigner une date de début, une date de fin et les ressources humaines et matérielles nécessaires à l'exécution de chaque opération du projet, tout en optimisant un critère donné. Ce critère peut être de type minimisation ou de type maximisation. Sachant qu'on peut toujours transformer un problème de maximisation en un problème de minimisation.

2.1.2 Modélisation d'un projet

La structure de chaque projet peut se modéliser à l'aide des éléments suivants.

2.1.2.1 Les tâches

Une tâche ou activité ou opération est une entité élémentaire d'un projet qui doit être accomplie dans une période de temps définie ou avant une date limite pour atteindre des objectifs liés au projet. Pour la réalisation de chacun des tâches d'un projet on identifie une durée intrinsèque (temps de référence), des besoins en ressources et un intervalle d'exécution (dates de début et de fin). La première et la dernière tâche marquent, respectivement, le début et la fin du projet. On distingue deux types de tâches :

- Tâches préemptives : une tâche préemptive est autorisée à être interrompue et reprise ultérieurement avec ou sans pénalité. Il est parfois important d'exécuter une tâche avec une priorité plus élevée avant une autre tâche moins prioritaire, même si la tâche moins prioritaire est toujours en cours d'exécution.

- Tâches non-préemptives : une tâche non préemptive, quant à elle, doit être exécutée jusqu'à son achèvement sans aucune interruption.

2.1.2.2 Les ressources

Une ressource est un moyen (humain ou matériel) requis pour l'exécution d'une tâche. Elle se caractérise généralement par une disponibilité limitée et une capacité positive. On distingue deux types de ressources pouvant être requises pas les tâches : les ressources renouvelables et les ressources consommables.

- Ressources renouvelables : une ressource est renouvelable si elle est disponible avec la même capacité à l'instant $t + 1$, après avoir accompli toutes les tâches à l'instant t. Les exemples typiques de ressources renouvelables sont les ressources humaines, machines, matériels, etc.

- Ressources consommables : une ressource est dite consommable si sa quantité diminue avec l'utilisation des tâches de cette ressource. Cela conduit au fait que cette ressource peut être épuisée à un instant t. L'argent est un exemple typique de ressources consommables.

On utilise aussi le terme "capacité" qui caractérise les ressource de chaque type (renouvelables ou non-renouvelables). Pour les ressources renouvelables, la capacité d'une ressource indique la quantité d'unité de ressource qui est possible d'utiliser sur une période. Dans le cas des ressources consommables, elle indique la quantité de ressource qui peut être consommée sur l'horizon. Lorsqu'une ressource a une capacité égale à 1 par unité de temps de traitement, on parle d'une ressource disjonctive. Autrement, lorsque la capacité est supérieure à 1, on parle d'une ressource cumulative.

Dans cette thèse, nous n'utilisons que des ressources renouvelables disjonctives.

2.1.2.3 Les compétences

Dans notre contexte, une compétence modélise la capacité d'utiliser des connaissances d'une ressource appropriée dans l'exécution ou la réalisation d'une tâche. Si une ressource est capable d'exercer plusieurs compétences de types différents, elle est appelée ressource multi-compétence. Tel est le cas d'une machine multi-fonctionnelle ou une ressource humaine.

2.1.2.4 Les contraintes

Dans chaque projet, les tâches et les ressources en général sont soumises à des limitations qui affectent l'exécution du projet. Ces limitations sont exprimées avec des termes formels de gestion de projet comme des contraintes. Par exemple, une contrainte peut être utilisée pour limiter le début d'une tâche à une date donnée, plutôt que de laisser l'ordonnancement détermine cette date. En général, la plupart des contraintes peuvent être classées sous trois classes : les contraintes temporelles, les contraintes de partage de ressources et les contraintes de coût.

Examinons en détail chacune de ces trois classes de contraintes.

Contraintes temporelles

On considère habituellement deux type de contraintes temporelles, les contraintes de précédence et les contraintes de localisation temporelles.

- Contraintes de précédence : Elles imposent un ordre d'exécution de certaines tâches par rapport à d'autres. Pour mieux comprendre ce type de contraintes, considérons une tâche n_j qui est associée avec une date de début S_i, une date de fin de réalisation C_i et un temps de traitement p_i. Les équations ci-dessous montrent des contraintes de précédence qui peuvent le plus souvent relier la tâche n_i avec une autre tâche n_j.

  $$S_j - S_i \geq p_i$$

  La tâche n_j doit commencer après la fin de la tâche n_i.

  $$S_j - S_i > 0$$

  La tâche n_j doit commencer après le début de la tâche n_i.

  $$0 < S_j - S_i < p_i \quad \text{et} \quad C_j - C_i > 0$$

  La tâche n_j doit commencer avant la fin de la tâche n_i.

  $$C_j - C_i > 0$$

  La tâche n_i doit finir avant la tâche n_j.

- Contraintes de localisation temporelles : ce type de contraintes, quant à lui, est utilisé pour positionner de manière absolue les tâches dans le temps. On peut trouver ce type de contraintes sur la date de début et/ou sur la fin de la tâche. Dans le première cas, dénommé date de disponibilité (ou *release date*), il s'agit d'imposer une date avant laquelle une tâche ne peut être commencée. Cela peut se produire, par exemple, si l'exécution d'une tâche doit attendre la livraison d'un approvisionnement. Dans certains cas, comme celui de ce manuscrit, on parle de contraintes sur la date de début au plus tôt. Cette contrainte est utilisée lorsque les date de disponibilité sont déduites des contraintes de précédence. Dans le second cas, il s'agit plutôt d'imposer une date maximale (jalon) avant laquelle la tâche doit se terminer.

Contraintes de partage de ressources

Les ressources utilisées pour la réalisation du projet sont souvent soumises à des contraintes de disponibilités et de capacité. Deux types de contraintes peuvent être induites selon la nature des ressources (renouvelables ou non-renouvelables) : les contraintes disjonctives et les contraintes cumulatives.

- Contraintes disjonctives : comme mentionné avant, les ressources disjonctives ne peuvent effectuer qu'une seule tâches à la fois. Si une contrainte disjonctive relie deux tâche n_i et n_j, alors elle modélise la non-similarité entre ces deux tâche : (n_i est avant n_j) ou (n_j est avant n_i) .
- Contraintes cumulatives : elles imposent la prise en compte de la disponibilité des ressources cumulatives. Plus précisément, une contrainte cumulative peut imposer que la somme de l'utilisation des ressources durant chaque période de temps ne dépasse pas la capacité de ces ressources. Elle peut aussi imposer que la consommation des ressources sur l'ensemble de l'horizon ne dépasse pas la capacité des ressources.

Contraintes de coût

Le coût ou le budget d'un projet comprend l'ensemble de ressources financières nécessaires à sa réalisation. Par nature, le budget d'un projet ne peut être respecté puisqu'il s'agit d'une estimation. D'un autre coté, le succès d'un projet dépend souvent de son achèvement dans la limite de son budget imparti. Pour limiter autant que possible l'excès du coût du projet durant le calcul d'ordonnancement, une contrainte de coût peut être imposée sous forme de montant qu'il ne faut pas dépasser sous peines d'avoir des pénalités.

Dans cette thèse, nous considérons uniquement les contraintes de localisation temporelles et les contraintes de partage de ressources.

2.1.2.5 Contraintes dures et souples

Dans de nombreux problèmes d'ordonnancement de projet, il est parfois impossible de trouver une solution qui satisfait toutes les contraintes du modèle. Dans ce cas, il est préférable d'établir une classification des contraintes selon les préférences du décideur. Cette classification peut être basée sur une priorité permettant de distinguer les contraintes dures des contraintes molles (ou souples). Une contrainte dure est une contrainte qui doit absolument être respecter par toute solution réalisable du modèle. Par contre, une contrainte molle ou souple peut être violée dans certains cas fixés le plus souvent par le décideur. Par exemple si aucune solution réalisable ne peut être trouvée, ou bien la solution est réalisable mais de mauvaise qualité (coût élevé, ressources surchargées, etc.). Pour un ordonnancement efficace, il est préférable de respecter autant que possible les contraintes molles. Par conséquent, la violation de ces contraintes peut entraîner des pénalités à minimiser durant le processus de l'ordonnancement.

2.1.2.6 Objectif

Dans chaque problème d'ordonnancement de projet, le qualité d'une solution est évaluée en fonction d'un ou plusieurs objectifs à optimiser. Lorsque plusieurs objectifs sont à considérer par le processus de l'ordonnancement, on parle d'un problème d'ordonnancement multi-objectif ou multi-critère (nous reviendrons sur ce sujet dans le Chapitre 5). Plusieurs types de fonctions objectifs sont considérés dès lors nous nous intéressons aux problèmes d'ordonnancement. Nous ne présentons ci-dessous que ceux liées à la durée du projet.

- Minimiser la durée du projet ou makespan, consiste à la minimisation de la date de fin de la dernière tâche du projet. Minimiser le makespan d'un projet est noté dans la littérature par

$\sum C_{max}$. Le makespan d'un projet est souvent mesuré par rapport au makespan de son chemin critique non-contraint. Une estimation de ce dernier correspond à la durée du projet obtenu en exécutant chaque tâche le plus rapidement possible tout en négligeant les contraintes sur les ressources.

- Minimiser les retards lorsque les tâches ou le projet sont soumis à des dates de fin souhaitées. Cet objectif modélise les exigences des clients en terme de délais de mise en œuvre. Si une tâche n_j non-préemptive commence à la date s_j, elle se termine alors à la date $C_j = s_j + p_j$, p_j est la durée de cette tâche. On peut calculer le retard absolu de cette tâche par la fonction : $T_j = max\{0, C_j - d_j\}$, où T_j et d_j désignent respectivement le retard absolu et la date de fin souhaitée de la tâche n_j. Le retard algébrique est défini dans la littérature par L_j, et se calcule également de la même manière. On utilise la fonction T_{max} (resp. L_{max}) pour dénoter le retard absolu (resp. algébrique) maximum.

  Dans certaines situations, comme dans le cas de cette thèse, les retards des tâches sont associés à des pénalités particulières. Ce qui permet de prioriser la finalisation de certaines tâches par rapport à d'autres ; on parle alors de retards pondérés. On utilise la fonction $\sum w_j T_j$ pour modéliser la somme des retards pondérés.

- Minimiser la somme des dates de fin des tâches, notée dans la littérature par $\sum C_j$. Cet objectif correspond à la minimisation du temps d'attente total des tâches avant de commencer leur exécution. Autrement, il correspond à minmiser les encours.

2.1.3 Notation de Graham

À la fin des années 1970, Graham et al. Graham et al. (1979) ont introduit une notation standard, largement utilisée aujourd'hui, permettant de distinguer les différents problèmes d'ordonnancement. Cette notation, nommée notation de Graham, consiste à trois champs : α, β et γ. Le champ α spécifie l'environnement de travail, β décrit les caractéristiques des tâches et les contraintes, et γ indique la fonction objectif. Selon le problème étudié, les trois champs prennent des formes différentes. Par exemple, les problèmes ci-dessous sont des problèmes d'ordonnancement d'atelier.

- $1 \mid pmtn, r_j \mid \sum w_j T_j$: Une seule machine, tâches préemptives, dates de début au plus tôt, minimiser la somme des retards pondérés.
- $Pm \mid prec, d_j \mid C_{max}$: Plusieurs machines parallèles identiques, contraintes de précédence, dates de fin souhaitées, minimiser la durée totale du projet.
- $Qm \mid\mid L_{max}$: Plusieurs machines parallèles uniformes, minimiser le retard algébrique maximum.

2.1.4 Complexité

Dans cette section, on aborde les différentes notions de la théorie de la complexité. Cette dernière comprend deux aspects : la complexité des algorithmes et la complexité des problèmes.

2.1.5 Complexité des algorithmes

La complexité d'un algorithme est l'analyse visant à mesurer la difficulté intrinsèque à résoudre un problème donné en exécutant cet algorithme. Cette complexité est un indicateur très important, principalement utilisé pour évaluer et comparer les performances d'un algorithme par rapport à d'autres réalisant le même traitement. Le calcul de la complexité d'un algorithme peut se baser sur

le temps d'exécution de l'algorithme (complexité temporaire) et l'espace mémoire requis pour son exécution (complexité spatiale).

La complexité temporaire est un mesure efficace, permettant d'évaluer la performance d'un algorithme. Elle implique les limites du temps qu'il faut à un algorithme pour produire ses résultats. Ces limites sont données en termes de nombre d'opérations élémentaires à exécuter par l'algorithme dans le pire des cas. Par exemple, si on veux renvoyer l'élément minimum dans un tableau de taille n, dans le pire des cas, cela prend n opérations (l'élément est trouvé à l'indice n du tableau). La complexité temporaire dépend donc fortement de la taille de l'instance résolue. On note que dans la plupart des cas, c'est la complexité temporaire qui est utilisée pour comparer la performance des algorithmes.

La complexité spatiale, quant à elle, est définie par la quantité totale de l'espace mémoire utilisé par un algorithme pour son exécution. Ainsi, pour trouver la complexité spatiale d'un algorithme, il suffit de calculer dans le pire des cas l'espace mémoire occupé par les variables utilisées. Pour calculer cette complexité, nous devons alors connaître l'espace mémoire utilisé pour stocker chaque variable de différents type de données. Cet espace mémoire varie généralement selon les systèmes d'exploitation ainsi que les langages de programmation.

La complexité d'un algorithme est couramment notée à l'aide d'un grand $\mathcal{O}$, par exemple $\mathcal{O}(n)$, où n est le nombre des données d'entrée. On distingue par ordre croissant quelque grandes classiques de complexité : $\mathcal{O}(1)$ (constante), $\mathcal{O}(\log(n))$ (logarithmique), $\mathcal{O}(n)$ (linéaire), $\mathcal{O}(n^k)$ (polynomiale), $\mathcal{O}(c^n)$ (exponentielle), $\mathcal{O}(n!)$ (factorielle). Il est à noter que la classe de complexité d'un algorithme est un indicateur de son efficacité. En effet, plus sa classe sera proche de $\mathcal{O}(1)$ ou $\mathcal{O}(\log(n)$ plus l'algorithme sera meilleur.

2.1.6 Complexité des problèmes

L'étude de la complexité des problèmes vise à déterminer la complexité intrinsèque d'un problème et à classifier les problèmes selon celle-ci. La complexité d'un problème correspond à la complexité dans le pire des cas de l'algorithme le plus efficace qui le résout. Un des rôles de l'étude de la complexité d'un problème est d'obtenir une borne inférieure (on ne peux pas faire mieux) sur ses méthodes de résolution. Par exemple, le problème du tri d'un tableau de n éléments peut être résolu au pire des cas par un algorithme dit "tri rapide" en n^2 opérations, tandis que celui de "tri par fusion" ne nécessite que $\mathcal{O}(n \cdot \log(n))$ opérations. Ce qui rend la complexité de ce problème est donc de $\mathcal{O}(n \cdot \log(n))$.

Au cœur de la théorie de la complexité se trouve les problèmes de décision ainsi que les problèmes d'optimisation. Un problème de décision peut être posé sous la forme d'une question dont la réponse est "oui" ou "non". Dans les problèmes de décision, le but est de trouver une solution parmi d'autres, tandis que dans les problèmes d'optimisation, le but est de trouver la meilleure solution. Chaque problème d'optimisation peut être transformé en un problème de décision. Par exemple, dans le problème du voyageur de commerce Rosenkrantz et al. (1974), le but est de trouver le plus court chemin qui passe une seule fois par chaque ville d'une liste de villes données et revient à la ville de départ. Ce problème d'optimisation peut être formulé dans la question suivante : existe-t-il un chemin de longueur inférieure à N. Ainsi, la complexité d'un problème d'optimisation est relative à la complexité du problème de décision correspondant.

Les problèmes de décision sont regroupés en plusieurs classes de complexité, dont les principales sont les suivantes.

Definition 1. ***Classe*** $\mathcal{NP}$ *Un problème de décision appartient à la classe* $\mathcal{NP}$*, si pour toute instance de ce problème, on peut vérifier en temps polynomial la validité de la réponse pour cette instance. Autrement dit, toutes les solutions données pour un problème* $\mathcal{NP}$ *peuvent être vérifiées rapidement, mais on ne connaît pas un algorithme efficace pour résoudre ce problème. On distingue*

deux grandes familles de problèmes dans cette classe : les problèmes faciles à résoudre (classe $\mathcal{P}$) et les problèmes difficiles à résoudre (classes $\mathcal{NP}$-complet et $\mathcal{NP}$-difficile).

Definition 2. ***Classe*** *$\mathcal{P}$ Un problème de décision appartient à la classe $\mathcal{P}$, si chaque instance de ce problème peut être résolue par un algorithme en temps polynomial ($\mathcal{O}(n^k)$), où n est le nombre de données d'entrée et k une constante positive.*

Definition 3. ***Classe $\mathcal{NP}-$complet*** *On dit qu'un problème P est $\mathcal{NP}$-complet, s'il appartient à la classe $\mathcal{NP}$ et que tout problème dans $\mathcal{NP}$ se réduit polynomialment à P. La réduction est définie ci-dessous.*

Definition 4. ***Réduction polynomiale*** *Un problème de décision P_1 est réductible au problème P_2 (noté $P_1 \propto P_2$), s'il existe une fonction polynomiale f qui peut construire pour toute instance I_1 de P_1, une instance $I_2 = f(I_1)$ de P_2 tel que la réponse au problème P_1 pour l'instance I_1 est "oui" si et seulement si la réponse au problème P_2 pour l'instance I_2 est "oui".*

Definition 5. ***Classe $\mathcal{NP}$-difficile*** *Un problème de décision est dit $\mathcal{NP}$-difficile si un algorithme pour le résoudre peut être traduit en un algorithme pour résoudre tout autre problème $\mathcal{NP}$. Les problèmes de cette classe sont au moins aussi difficiles que les problèmes de la classe $\mathcal{NP}$.*

Selon le niveau de difficulté des problèmes $\mathcal{NP}$-difficiles, on distingue deux sous-classes de problèmes, à savoir la classe des problèmes $\mathcal{NP}$-difficiles au sens fort et la classe des problèmes $\mathcal{NP}$-difficiles au sens faible. Un problème $\mathcal{NP}$-difficile au sens faible peut être résolu par un algorithme pseudo-polynomial. Sinon, si aucun algorithme pseudo-polynomial ne peut le résoudre, le problème est dit $\mathcal{NP}$-difficile au sens fort.

2.2 Méthodes de résolution

La majeure partie des problèmes d'ordonnancement de projet sont connus dans la littérature spécialisée comme des problèmes de forte complexité. Ainsi, cette complexité a mené à une variété de méthodes de résolutions proposées pour des modèles d'ordonnancement inspirés souvent des applications réelles. En effet, la résolution d'un problème d'ordonnancement se fait à l'aide de deux grandes familles de méthodes de résolution. D'une part, les méthodes exactes qui garantissent l'optimalité des solutions, d'autre part, les méthodes approchées qui retournent des solutions admissibles dans des délais raisonnables.

2.2.1 Méthodes exactes

Une méthode exacte est toute procédure capable de toujours donner la solution optimale à un problème donné. Ce type de méthodes ne convient pas pour résoudre des instances des problèmes $\mathcal{NP}$-difficiles, sauf s'ils sont de petites tailles.

Sans être exhaustif, nous nous intéressons aux méthodes exactes qui seront particulièrement utilisées dans cette thèse.

2.2.1.1 Programmation linéaire

La programmation linéaire est un cas particulier de la programmation mathématique. Cette dernière est définie comme un outil très puissant utilisé pour formaliser des problèmes d'optimisation au moyen de paramètres, de variables de décision, fonctions objectifs et contraintes Leo (2009). Pour faire simple, la forme typique d'un programme mathématique peut être exprimée comme suit :

$$\text{Minimiser} \quad f = \sum_{i=1}^{n} c_i x_i$$

$$\text{Sous contraintes} \sum_{i=1}^{n} a_i x_i < b$$

$$\forall x_i \geq 0$$

La programmation linéaire (PL), quant à elle, cherche à minimiser ou à maximiser une fonction objectif linéaire sous contraintes d'égalités et d'inégalités linéaires.

La programmation linéaire en nombres entiers (PLNE), comme son nom l'indique, est une extension de la programmation linéaire, dans laquelle toutes ou une partie des variables de décision sont des nombres entiers. On distingue trois variantes des programmes linéaires en nombres entiers : les programmes linéaires en nombres entiers simples, les programmes linéaires mixtes, et les programmes en variables binaires.

Dans les programmes linéaires en nombres entiers simples toutes les variables doivent être des nombres entiers. Alors que, dans les programmes linéaires mixtes, l'ensemble des variables contient à la fois des variables entiers ou binaires et des variables continues. Dans les programmes linéaires en variables binaires, les variables sont toutes des nombres binaires (c'est-à-dire qu'une variable doit être égale à 0 ou 1).

2.2.1.2 Programmation par objectif

La programmation par objectif, également connue sous le terme "goal programming", est une généralisation de la programmation linéaire utilisée pour gérer la présence de multiples objectifs simultanément dans un modèle. Plus précisément, à partir des buts (fixés par le décideur), elle cherche à trouver une solution qui minimise les déviations des objectifs par rapport à ces buts. Les déviations des objectifs peuvent être positives, en cas de dépassement des buts, ou bien négatives, dans le cas contraire. Contrairement à la programmation linéaire qui ne considère que les contraintes dures, la programmation par objectifs considère des contraintes dures et molles. Ces contraintes molles sont modélisées comme des objectifs pour lesquels les déviations (violations) doivent être minimisées dans la fonction objectif globale.

2.2.2 Méthodes approchées

Une méthode approchée, également appelée une heuristique, est une démarche visant à déterminer dans un délai raisonnable une solution admissible pour un problème donné. Cette solution peut être ou ne pas être la solution optimale. Ces méthodes sont évaluées et comparées en fonction de leurs déviations (relatives ou absolues) par rapport à l'optimalité. Plus cette déviation est petite, plus la solution proposée est bonne.

Sans être exhaustif, nous présentons ci-dessous les méthodes approchées les plus répandues pour résoudre les problèmes d'ordonnancement de projet. Ces méthodes sont d'ailleurs utilisées dans le cadre de cette thèse.

2.2.2.1 Algorithme glouton

Un algorithme glouton (*greedy algorithm*, en anglais) est une heuristique dédiée au problème étudié, qui est souvent basée sur une recherche miopique. Son idée consiste à choisir une solution qui est meilleure que toutes les autres solutions qui sont légèrement différentes. Cette solution est dite un minimum local. Par ailleurs, un algorithme glouton ne garantit pas que la solution obtenue soit une solution optimale (minimum global). À cette fin, ce type d'algorithmes est souvent utilisés par des méthodes plus complexes pour générer des solutions initiales qui seront ensuite améliorées de manière itérative.

Tab. 2.1 : algorithm

Pseudo-code de la recherche locale

```
1 : x* ← x0 ;Initialiser la meilleure solution trouvée à la solution initiale.
2 : tant que Areet faire
3 :     V(x*) ← GenererVosinage() ; Solutions voisines générées par un opérateur
    de voisinage.
4 :     pour chaque x' ∈ V(x*) faire
5 :         si (x' est meilleure que x*) alors
6 :             x* ← x'
7 :             Aller à l'itération suivante.
8 :         fin si
9 :     fin pour
10 :     Aller à l'itération suivante.
11 : fin tant que
```

Par exemple, le problème de la sélection des tâches, dont l'énoncé ci-dessous, pourrait être résolu de manière optimale avec un algorithme glouton.

Considérons que vous avez N tâches avec leurs dates de début et de fin connues et fixées. Toutes les tâches ont la même priorité et même poids. L'objectif est de sélectionner le nombre maximum des tâches qui peuvent être exécutées par une seule ressource, en supposant qu'une ressource ne peut travailler que sur une seule tâche à la fois. Il se peut qu'il ne soit pas possible de réalisées toutes les tâches, car leurs durées se chevauchent.

Une heuristique gloutonne peut être utilisée pour trouver la solution. Son idée consiste à toujours choisir la tâche dont la date de fin est la plus proche et dont la date de début est supérieure ou égale à la date de fin de la tâche précédemment sélectionnée. Les tâches peuvent être triées en fonction de leurs dates de fin afin de toujours considérer la tâche suivante comme la tâche ayant la date de fin la plus proche.

2.2.2.2 Recherche locale

Les méthodes de recherche locale ont été appliquées avec succès pour résoudre divers problèmes d'optimisation (voir par exemple, Ishibuchi and Murata (1998) ; Tseng and Lin (2009) ; Choi and Choi (2002)). Cette méthode consiste à déterminer itérativement une solution dans l'espace des solutions candidates (l'espace de recherche) en appliquant des modifications locales simples. La procédure commence à partir d'une solution initiale donnée et se déplace itérativement vers une solution voisine (proche) dans l'espace de recherche. La solution initiale est considérée comme la solution actuelle jusqu'à ce qu'une meilleure solution soit trouvée. Les solutions sont comparées en se basant sur la valeur de la fonction objectif du problème (par exemple, le coût de la solution, le nombre de contraintes violées). Dans sa variante la plus simple, à chaque itération, de nouvelles solutions sont générées à partir de la solution courante. Ces nouvelles solutions voisines sont générées en appliquant un opérateur de voisinage. Si une nouvelle solution améliore la solution courante, la solution voisine devient alors la nouvelle solution courante et le point de départ de l'itération suivante. Dans le cas contraire, c'est-à-dire après avoir évalué toutes les solutions voisines et qu'aucune amélioration n'est observée, alors on dit qu'un minimum local est atteint. L'algorithme s'arrête lorsqu'un critère d'arrêt est satisfait, ou il n'y a aucun moyen d'améliorer la solution actuelle. Dans le dernier cas, on dit qu'un minimum local est atteint.

Un pseudo-code de cette méthode est présenté dans l'algorithme 2.1.

2.2.2.3 Recherche tabou

Le principal défaut des recherches locales est qu'elles s'arrêtent lorsqu'un minimum local est atteint. C'est pourquoi elles exploitent peu toutes les parties de l'espace de solutions. Il existe plusieurs mécanismes permettant d'échapper aux minimums locaux, soit par exemple en redémarrant

Algorithm 1 : Pseudo-code de la recherche tabou

$x^* \leftarrow x_0$;Initialiser la meilleure solution trouvée à la solution initiale.
$x^c \leftarrow x_0$; Initialiser la solution courante à la solution initiale.
$T = \phi$; Initialiser la liste tabou.
tant que *Areet* **faire**
    $V(x^c) \leftarrow GenererVosinage()$; Solutions voisines générées par un opérateur de voisinage.
    $x^c \leftarrow x'_0$; Initialiser la solution courante à la première solution voisine évaluée.
    pour chaque $x' \in V(x^c)$ **faire**
        si (x' est meilleure que x^c) et (x' n'est pas dans T) **alors**
            $x^c \leftarrow x'$
            Mettre à jour T.
        fin si
    fin pour
    si (x^c est meilleure que x^*) **alors**
        $x^* \leftarrow x^c$
    fin si
    Aller à l'itération suivante.
fin tant que

la recherche à partir d'une nouvelle solution ou bien en augmentant la taille du voisinage, etc. Les méthodes les plus courantes qui utilisent ces mécanismes sont les métaheuristiques. Les métaheuristiques sont des heuristiques de haut niveau conçues pour trouver des solutions suffisamment bonnes à un problème d'optimisation. Contrairement aux heuristiques simples, l'implémentation d'une métaheuristique est indépendante de la nature du problème. Ce qui signifie qu'elle est capable de traiter une grande variété de problèmes. Parmi les exemples connus de métaheuristiques, citons les algorithmes génétiques, la recherche tabou, le recuit simulé, méthodes à large voisinages, bien qu'il en existe d'autres.

La recherche tabou (ou tabu search en anglais) est une métaheuristique, initialement proposée par Glover (1989, 1990), qui a été largement utilisée pour résoudre des problèmes d'optimisation combinatoire : voir, par exemple Alonso-Pecina et al. (2013) ; Pan et al. (2008) ; Mika et al. (2008). Cette méthode emploie des stratégies de diversification et de mémorisation permettant d'éviter respectivement les minimums locaux et les solutions déjà visitées dans l'espace de recherche.

La technique de base de la recherche tabou est la suivante. Dans sa forme la plus simple, la recherche tabou peut être considérée comme une recherche locale, ou les voisinages des solutions courantes sont itérativement évaluées jusqu'à ce qu'un critère d'arrêt soit satisfait. Contrairement à la recherche locale qui s'arrête si aucune solution améliorante n'est obtenue, la recherche tabou autorise les mouvements qui n'améliorent pas la valeur de la fonction. Autrement dit, à chaque itération, la meilleure solution voisine obtenue est retenue, même si elle n'améliore pas la fonction objectif courante. Elle sera considérée comme la solution courante de l'itération suivante. Pour éviter de revenir à une solution déjà visitée, la méthode utilise une mémoire temporaire (appelée liste tabou) qui est mises à jour et exploitée au cours de la recherche. Il s'agit de stocker les solutions qui sont récemment visitées pour ne pas retourner trop rapidement vers ces solutions. Ces solutions sont alors considérées tabou, d'où le nom de la méthode.

D'autres techniques sophistiquées peuvent être appliquées au cours de la recherche pour améliorer l'efficacité de cette méthode. Ces techniques sont l'intensification et la diversification de la recherche. En utilisant une technique de diversification, la procédure de la recherche tabou passe à une nouvelle partie de l'espace de des solutions lorsque celle-ci se bloque autour d'un minimum local. Contrairement à la technique de diversification, l'intensification est appliquée lorsque l'espace de voisinage de la solution courante est prometteuse. Ainsi, la recherche doit être intensifiée autour de cet espace de solutions.

Un pseudo-code de cette méthode est présenté dans l'algorithme 1.

2.3 Conclusion

Ce chapitre a été consacré à la présentation du contexte global des problèmes d'ordonnancement abordés dans cette thèse. Dans un premier temps, nous avons introduit les concepts qui nous permettront de mieux comprendre les problèmes d'ordonnancement de projets. Dans un second temps, nous avons évoqué les méthodes de résolution, méthodes exactes et méthodes approchées, qui seront particulièrement utilisées dans cette thèse.

Dans le chapitre suivant, nous présentons un état de l'art qui se centre essentiellement sur les travaux liés aux problèmes abordés dans la première partie de cette thèse. Plus précisément, nous présentons une revue de la littérature sur les problèmes d'ordonnancement de projets sous contrainte de ressources et ses deux extensions, multi-projet et multi-compétence.

Chapitre 3

État de l'art

Contents

Les problèmes d'ordonnancement abordés dans le cadre de cette thèse sont liés principalement à deux sujets d'ordonnancement. La majeur partie concerne le problèmes d'ordonnancement de projet à contraintes de ressources limitées (ou RCPSP). Nous nous intéressons particulièrement à ses extensions : multi-compétence et multi-projet.

Afin de mieux situer nos travaux, nous présentons dans ce chapitre un état de l'art sur les travaux les plus pertinents liés au problème RCPSP. Une attention particulière est apportée aux différentes approches de résolutions proposées. La seconde partie de cette thèse porte sur un problème d'ordonnancement multi-agent. Dans le cadre de ce sujet, une revue de la littérature est présentée dans le Chapitre 6.

3.1 Introduction

Le RCPSP (Resource Constrained Project Scheduling Problem) est l'un des problèmes d'ordonnancement les plus couramment rencontrés. Les études sur ce problème et ses différentes extensions représentent la majeure partie de la littérature sur l'optimisation et la gestion de projet.

Sans perte de généralité, le RCPSP consiste en particulier à planifier l'exécution d'un ensemble fini des tâches à l'aide des ressources renouvelables (humaines ou machines) limitées. Kolisch and Hartmann (1999) ont défini la version classique de ce problème comme suit. Soit un projet découpé en J tâches non péemptives labellisées $n_j = 1, \ldots, n_J$. Les tâches sont liées entre elles par des

relations de précédence classiques de type fin-début. Ce qui se traduit par le fait qu'une tâche ne peut être commencée tant que l'ensemble des tâches qui la précèdent ne sont pas achevées. Chaque tâche n_j a une durée opératoire notée p_j, à l'exception des tâches fictives n_0 et $nJ+1$, qui indiquent respectivement le début et la fin du projet. Afin d'accomplir le projet, K ressources renouvelables sont disponibles en quantité constante durant l'horizon de planification. Chaque tâche nécessite une quantité constante r_{jk} de la ressource k à chaque période de son exécution. L'objectif est de trouver un ordonnancement des tâches qui respect les contraintes de précédence entre les tâches, ainsi que la disponibilité des ressources. La solution doit minimiser la durée totale du projet (C_{max}). Le RCPSP appartient à la classe de problèmes $\mathcal{NP}$-difficiles au sens fort Blazewicz et al. (1983b).

Bien que le RCPSP, tel que présenté ci-dessus, soit déjà un modèle puissant, il ne peut pas couvrir toutes les situations dans la vie réelle qui nécessitent de l'ordonnancement et de l'optimisation des ressources. De ce fait, de nombreux chercheurs ont développé des modèles d'ordonnancement de projets plus généraux, en se basant sur le RCPSP standard comme modèle de départ. Nous renvoyons le lecteur, pour en savoir plus sur les étapes de l'évolution des modèles étudiés, à trois revues de la littérature espacées dans le temps : Herroelen et al. (1998) ; Hartmann and Briskorn (2010, 2021).

Le problème RCPSP et ses différentes variantes sont encore largement étudiés. Compte tenu de l'ampleur des recherches sur ce sujet, seuls les travaux examinant les variantes du RCPSP en lien avec les problèmes identifiés dans cette thèse sont abordés dans les sections ci-dessous. Dans les tableaux 3.1 et 3.2, nous résumons les études référencées tout au long de ce chapitre.

3.2 RCPSP préemptif

Une hypothèse fondamentale caractérisant le RCPSP est que les tâches en cours ne peuvent pas être interrompues. Cette hypothèse peut empêcher la modélisation de nombreuses situations pratiques. En effet, il arrive souvent, qu'une tâche soit interrompue en raison de l'indisponibilité ou de l'absence de ressources ou simplement pour exécuter une autre tâche plus critique. Dans le RCPSP préemptif, également connu par PRCPSP (pour Preemptive Resource-Constrained Project Scheduling Problem), les tâches peuvent être interrompues et reprises plus tard sans un coût supplémentaire ajouté. Selon Moukrim et al. Moukrim et al. (2015) une telle situation peut exister dans l'industrie textile, le développement des projets informatiques, l'ordonnancement multi-processeurs, etc.

Il convient de noter que dans cette section, nous ne discutons que des travaux d'ordonnancement préemptif basés sur le RCPSP standard. D'autres modèles d'ordonnancement préemptif dans un contexte multi-compétence sont présentés dans la Section 3.3.

Sur la base de notre revue de la littérature, on peut distinguer trois cas de préemption dans le PRCPSP : préemption entière, préemption continue et préemption discrète. La préemption entière, suppose que les interruptions des tâches ne peuvent se produire qu'à des instants entiers durant le processus de l'exécution. Une revue de la littérature sur les problèmes considérant la préemption entière est disponible dans Quintanilla et al. (2015). La préemption continue autorise l'interruption d'une tâche encours d'exécution à tout moment. Ce type de problème d'ordonnancement est considéré dans le premier travail sur le PRCPSP publié dans SłowínskiR (1980). Cependant, Ce n'est qu'en 2007, que le PRCPSP avec la préemption continue a été repris par Damay et al. Damay et al. (2007). Une revue approfondie de la littérature sur le PRCPSP contenue est développée dans Schwindt and Paetz (2015), qui étudient également un problème PRCPSP avec la préemption continue. Dans la préemption discrète, l'interruption d'une tâche est limitée à des instants définis durant le processus de l'exécution où la tâche a déjà atteint des portions de progression prédéfinies. À travers nos travaux de recherche, seule la préemption discrète est considérée.

À notre connaissance, la contrainte sur la non préemption des tâches à été relaxée pour la

première fois par Słowínski SłowínskiR (1980), qui utilise la programmation linéaire pour résoudre plusieurs problèmes de planification de projets avec des contraintes de ressources. Kaplan Kaplan (1988) dans sa thèse de doctorat présente des formulations mathématiques basées sur la programmation dynamique pour résoudre des problèmes PRCPSP considérant des préemptions entières. L'objectif dans son étude est de minimiser la durée du projet (C_{max}). Dans Demeulemeester and Herroelen (1996), un branch-and-bound (BB) est proposé pour résoudre un problème PRCPSP considérant également le cas des préemptions entières, où l'objectif est de minimiser le makespan. Dans Bianco et al. (1999), les auteurs considèrent une variante PRCPSP avec des préemptions entières et des ressources de capacité unitaire. Ils montrent que le problème est équivalent à un problème de coloration des sommets pondérés. Pour résoudre le problème étudié, les auteurs développent une méthode exacte basée sur une branch-and-bound proposé pour un problème de coloration des sommets pondérés. Damay et al. Damay et al. (2007) considèrent que certaines tâches peuvent être interrompues à des instants arbitraires (préemption continue). Les auteurs présentent une méthode de recherche de voisinages basée sur la programmation linéaire (ou LP-based neighborhood search). Les auteurs de Vanhoucke and Debels (2008) adaptent un branch-and-bound, présenté dans Demeulemeester and Herroelen (1992) pour résoudre un problème RCPSP standard, afin d'inclure des préemptions entières et l'exécution de plusieurs parties de la tâche en parallèle (ou fast-tracking). Dans l'article de Ballestín et al. Ballestín et al. (2008), ils envisagent un modèle PRCPSP avec le cas préemption discrète, où une seule interruption maximale par tâche est autorisée. Les auteurs montrent comment trois éléments basiques (c'est-à-dire le codage de la solution, le schéma de génération de programmes en série SGS, et la double justification) dans plusieurs heuristiques résolvant le RCPSP peuvent être adoptées pour résoudre le cas général intègre la préemption. Ils se focalisent également sur une généralisation du problème où un nombre maximum d'interruptions par tâche est autorisé. Dans Zhu et al. (2011), un nombre maximum d'interruptions par tâche est également autorisé. Un algorithme génétique est proposé pour le problème considéré avec l'objectif de minimiser la valeur du makespan. Moukrim et al. Moukrim et al. (2015) présentent un algorithme de type branch-and-price pour résoudre toutes les instances $J30$ de PSPLIB Kolisch and Sprecher (1997), et pour obtenir des bornes inférieures pour les instances $J60$, $J90$, et $J120$. Dans Schwindt and Paetz (2015), les auteurs proposent un modèle PLNE pour un problème PRCPSP avec des préemptions continues. L'objectif du modèle est de minimiser le makespan. Shou et al. Shou et al. (2015) présentent un algorithme hybride d'optimisation par essaims de particules (ou particle swarm optimization (PSO)) pour résoudre un modèle PRCPSP dans lequel une seule interruption par tâche est autorisée. Chen et al. Chen and Zhang (2016) s'intéressent à un problème PRCPSP avec des tâches de durées aléatoires et préemptives. Les auteurs proposent un modèle mathématique avec des variables aléatoires floues pour le problème étudié. L'objectif du modèle est de maximiser la somme de la marge libre qui contient deux parties : l'une est le délai dans lequel la date de début de certaines tâches peut être repoussé, l'autre est le manque de ressources autorisé. Dans Cheng et al. (2015), les auteurs examinent l'impact de la préemption sur le makespan du projet. Des expérimentations sont menées pour comparer les durées optimales du projet avec trois paramétrages différents : le RCPSP standard, le RCPSP avec fractionnement des tâches, et le RCPSP préemptif. Sur la base des résultats obtenus, les auteurs concluent que la préemption des tâches peut conduire à une réduction importante du makespan du projet. Récemment, l'auteur de Creemers (2019) s'est intéressé à un problème PRCPSP avec des préemptions continues et des tâches de durées stochastiques. Il utilise une nouvelle chaine de markov à temps Continu (ou continuous-time Markov chain (CTMC)) pour la résolution exacte du problème étudié. Les articles publiés dans Afshar-Nadjafi and Majlesi (2014) ; Vanhoucke and Coelho (2019) supposent que chaque préemption de la tâche ne peut se produire qu'à un instant entier, et nécessite un temps de réglage (setup time) additionnel avant la reprise de la tâche. L'objectif est de minimiser le makespan. Afshar-Nadjafi et al. présentent une formulation PLNE ainsi qu'un algorithme génétique. Vanhoucke et al. considèrent cinq types de setup times afin de pénaliser la fragmentation de la tâche. Ils proposent une nouvelle métaheuristique afin de résoudre le problème traité.

La majeur partie des études susmentionnées affirment que l'effet de la préemption des tâches est très significatif sur la valeur du makespan (réduit la durée du projet) et qu'il augmente avec la taille et la complexité du réseau de projet. Citons par exemple mais sans s'y limiter : Ballestín et al. (2008) ; Creemers (2019) ; Vanhoucke and Coelho (2019).

3.3 RCPSP multi-compétence

La version standard du RCPSP suppose que chaque personne n'a qu'une seule compétence spécifique ; et que chaque tâche nécessite une seule compétence pour son exécution. Toutefois, ces hypothèses ne sont pas toujours vérifiées d'un point de vue pratique. Par exemple, sans s'y limiter, il est très courant dans les entreprises de services informatiques que les employés soient multi-compétant, c'est-à-dire capables d'effectuer plusieurs type de tâches. On peut citer un développeur qui maîtrise différents langages de programmation (java, C#, C++). En outre, une tâche peut nécessiter une ou plusieurs compétences maîtrisées par différents employés. Par conséquent, le problème ne consiste pas seulement à choisir les employés qui seront chargés de chaque tâche, mais aussi les compétences avec lesquelles ils contribueront. Ce problème est connu dans la littérature par le MS-RCPSP (pour Multi-Skill Resources Constrained Project Scheduling Problem).

Depuis l'introduction de ce problème par Néron et al. Néron and Baptista (2002), le MS-RCPSP s'est avéré être une orientation fructueuse de recherches sur les modèles de l'ordonnancement de projets. Comme mentionné juste avant, le MS-RCPSP représente une variante du RCPSP, dans laquelle des ressources humaines multi-compétences sont impliquées. Cette variante RCPSP multi-compétence se concentre davantage sur les particularités des ressources humaines. Telles que les compétences qu'elles maîtrisent et parfois le niveau d'efficacité à exercer ces compétences. Dans le MS-RCPSP standard, chaque tâche nécessite une ou plusieurs personnes ayant des disponibilités limitées et des compétences variées. Les besoins en ressources pour chaque tâche sont exprimées par le nombre de compétences nécessaires à sa réalisation durant chaque période de temps. Ainsi, plusieurs personnes peuvent être impliquées à la fois pour remplir chacune des compétences requises par une même tâche. L'objectif est de déterminer un ordonnancement qui respecte les contraintes de précédences et des disponibilités de ressources, tout en minimisant la durée totale du projet (c'est-à-dire le makespan).

Du fait de la forte complexité du MS-RCPSP Bellenguez and Néron (2008), cette extension du RCPSP a retenu une attention non négligeable de la part des chercheurs. Ce problème permet d'adresser plusieurs cas réels d'ordonnancement dans différents domaines, tels que la construction des bâtiments, la production des logiciels informatiques, parmi d'autres Montoya et al. (2014). De ce fait, différentes approches de résolution, exactes ou approchées, ont été proposées pour résoudre ce problème d'optimisation combinatoire.

La majorité des modèles MS-RCPSP ont pour objectifs de minimiser soit la durée du projet, soit le coût de réalisation, ou les deux critères en même temps. Par exemple, afin de minimiser la durée du projet (ou le makespan), Bellenguez et al. Bellenguez and Néron (2007) proposent un algorithme exacte de type BB, alors que Almeida et al. Almeida et al. (2016) proposent une heuristique basée sur des règles de priorité. Montoya et al. Montoya et al. (2014) proposent un algorithme exacte de type Branch-and-Price (BP) pour résoudre le même problème. Leur modèle prend en considération la synchronisation entre les ressources. Cela signifie que toutes les ressources affectées à une tâche doivent commencer leur exécution simultanément. Afin de minimiser le coût du projet, Haitao et al. Haitao and Keith (2008) présentent deux formulations PLNE et PPC pour le problème traité. Les auteurs proposent également une heuristique hybride basée sur les deux modèles (PLNE/PPC) pour résoudre les instances de grandes tailles. Correia et al. Correia and da Gama (2014) développent une formulation PLNE pour traiter le même problème. Afin de minimiser les deux critères simultanément (c'est-à-dire la durée et le coût du projet), l'article publié dans Xiao et al. (2013) adressent une version du MS-RCPSP dans laquelle la durée de la tâche

est en fonction du nombre de ressources affectées à celle-ci. Un algorithme de colonies de fourmis (ou ant colony optimization (ACO)) est présenté pour résoudre le problème. L'article publié dans Laszczyk and Myszkowski (2019) propose un algorithme évolutionnaire de type NSGA-II (pour Non-dominated Sorting Genetic Algorithm-II).

3.3.1 Avec des niveaux hiérarchiques

D'autres études sur le MS-RCPSP ont inclus des niveaux hiérarchiques de compétences dans les modèles étudiés. Dans cette variante MS-RCPSP, chaque employé maîtrise plusieurs compétences avec des niveaux d'efficacité spécifiques. Dans certaines situations, une tâche peut exiger plusieurs compétences avec des niveaux de maîtrise minimum. Ce qui fait qu'une ressource pour accomplir une compétence requise par une tâche, doit avoir un niveau d'efficacité supérieur ou égal au niveau de compétence requis par cette tâche.

Bellenguez et al. Bellenguez and Néron (2005) développent des bornes inférieures ainsi qu'un modèle PLNE pour résoudre à l'optimum des instances de petites tailles. L'objectif est de minimiser la valeur du makespan (C_{max}). Dans Fırat and Hurkens (2012), les auteurs considèrent une contrainte supplémentaire qui exige la présence d'un ensemble de techniciens pendant toute la durée d'une journée de travail. Un modèle PLNE qui minimise la somme pondérée des dates de fin de tâches ($\sum w_j Cj$) est proposé dans leur étude. Les auteurs de Yannibelli and Amandi (2013) étudient un problème d'ordonnancement multi-objectif. L'un des objectifs est de minimiser le C_{max}. L'autre est d'affecter les ressources les plus efficaces à chaque tâche. Les auteurs supposent que l'efficacité d'un employé dépend de son contexte de travail. Cela inclut l'efficacité de tous les employés qui sont affectés avec lui à une tâche. De ce fait, pour chaque employé, il est possible de considérer différents niveaux d'efficacité en fonction de différents contextes de travail (par exemple, la tâche à laquelle l'employé est affecté, la compétence à laquelle l'employé est affecté, l'ensemble des employés affectés avec lui à la tâche). Afin de résoudre le problème traité, les auteurs développent un algorithme hybride composé d'un algorithme recuit simulé (simulated annealing (SA)) et d'un algorithme génétique. Maghsoudlou et al. Maghsoudlou et al. (2017) considèrent que le choix d'une ressource qui maîtrise une compétence à un niveau d'efficacité supérieur peut réduire le risque de retravailler une tâche, mais aussi peut augmenter le coût de la réalisation. Deux objectifs conflictuels sont considérés dans leur étude, à savoir la minimisation du risque de retravailler les tâches et la minimisation des coûts de l'affectation des ressources aux tâches (le coût du projet). Trois mécanismes multi-objectifs basés sur un algorithme de recherche coucou (ou cuckoo search (CS)) sont développés. Dans Myszkowski et al. (2017), une tâche nécessite plusieurs compétences mais elle doit être affectée à une seule ressource possédant toutes les compétences requises par cette tâche. L'objectif est de minimiser à la fois la durée et le coût du projet. Les auteurs utilisent quatre variantes d'algorithmes évolutionnaires qu'ils examinent sur des instances iMOPSE Myszkowski et al. (2015). Dans Hossein and Baradaran (2020), les auteurs considèrent un modèle MS-RCPSP bi-objectif qui intègre l'effet de détérioration linéaire des durées des tâches et des contraintes sur le budget financier. La détérioration linéaire signifie que la durée de la tâche augmente avec le retard de sa date de début (plus une tâche est traitée tardivement, plus le temps nécessaire à son exécution est long). L'objectif dans leur modèle est de minimiser simultanément la durée et le coût du projet. Les auteurs proposent un algorithme d'optimisation du loup gris basé sur Pareto (ou Pareto-based Grey Wolf Optimizer (P-GWO)) pour la résolution du problème étudié.

Dans les travaux publiés dans yu Zheng et al. (2017), Myszkowski et al. (2018), Wang and long Zheng (2018), Dai and Cheng (2019), Zhu et al. (2019), Lin et al. (2020), Zhu et al. (2021), chaque tâche exige une seule compétence d'un certain type à un niveau d'efficacité minimum. Dans yu Zheng et al. (2017), l'objectif est de minimiser le makespan. Un algorithme d'optimisation basé sur l'enseignement et l'apprentissage (ou Teaching–learning-based optimization (TLBO) algorithm) est proposé. Myszkowski et al. Myszkowski et al. (2018) appliquent une heuristique basée sur un

algorithme à évolution différentielle hybride combiné avec une heuristique gloutonne (ou Differential Evolution and Greedy Algorithm (DEGR)). L'objectif de leur travail est également de minimiser le makespan. Dans Wang and long Zheng (2018), les auteurs étudient un problème MS-RCPSP bi-objectif, minimisant simultanément le makespan et le coût du projet. Une métaheuristique de type mouche du vinaigre (ou fruit fly optimization algorithm (FOA)) est proposée. Dans Dai and Cheng (2019), les auteurs considèrent un modèle MS-RCPSP avec des détériorations linéaires des durées des tâches. L'objectif dans leur modèle est de minimiser également le makespan. Les auteurs proposent un algorithme mémétique (memetic algorithm (MA)) pour résoudre le problème traité. Dans Zhu et al. (2019), un algorithme évolutionnaire est présenté, basé sur l'optimisation multi-verse discret et oppositionnel (ou discrete oppositional multi-verse optimization (DOMVO)). L'objectif de leur travail est aussi la minimisation du makespan. Les auteurs du Lin et al. (2020) utilisent un algorithme de programmation génétique hyper-heuristique (ou genetic programming hyper-heuristic GP-HH), avec pour objectif de minimiser le makespan. Dans Zhu et al. (2021), les auteurs considèrent le même problème mais l'objectif cette fois-ci est de minimiser à la fois le coût et le makespan. Les auteurs appliquent un algorithme de type GP-HH.

Cependant, les études susmentionnées considèrent que les niveaux d'efficacité des ressources ont un impact uniquement sur l'affectation des ressources aux tâches, et pas sur les durées des tâches. Or, il est très courant dans un contexte réel que plus l'employé est efficace, moins il consacre d'efforts et de temps à sa tâche (c'est-à-dire la durée diminuera). L'inverse est vrai pour l'employé le moins efficace. Il existe peu d'extensions MS-RCPSP considérant que le niveau de maîtrise de la compétence a un effet sur la durée de la tâche. Le premier article à considérer cette hypothèse est celui de Valls et al. Valls et al. (2009). Dans leur étude, ils développent une relation linéaire entre l'efficacité et la durée de la tâche, comme dans les problèmes traités dans cette thèse. Les ressources (ou employés) sont classées selon trois niveaux d'efficacité : senior (125% efficace), standard (100% efficace) et junior (75% efficace). L'affectation d'un employée senior (resp. junior) à une tâche implique une réduction (resp. augmentation) de la durée standard de la tâche. Chaque tâche doit être affectée à une seule ressource et les critères à optimiser sont centrés sur les ressources, c'est-à-dire sur les niveaux d'urgence, sur l'efficacité de l'affectation et sur l'équilibre des charges de travail. Dans Snauwaert and Vanhoucke (2021), une relation réciproque est considérée entre l'efficacité et la durée de la tâche. Dans leur modèle, chaque tâche nécessite une ou plusieurs compétences et une ressource peut remplir exactement une de ces compétences. L'efficacité de chaque employé varie entre cinq niveaux : $\{0.5, 0.75, 1, 1.25, 1.5\}$. Une ressource avec un niveau d'efficacité standard a une valeur par défaut égale à 1. Une ressource plus (resp. moins) efficace est représentée par une valeur supérieure (resp. inférieure) à 1. Les auteurs considèrent alors que plus les niveaux d'efficacité des employés affectés à une tâche sont élevés, plus ils accompliront celle-ci rapidement. L'objectif dans leur étude est de minimiser la valeur du makespan. Pour résoudre ce problème, les auteurs développent un algorithme génétique.

L'effet de l'efficacité est également considéré dans des modèles MS-RCPSP liés à un contexte multi-projet, comme c'est le cas dans cette thèse. Dans les travaux publiés dans Heimerl and Kolisch (2010a) et Kolisch and Heimerl (2012), les auteurs développent une relation réciproque entre l'efficacité des ressources et la charge de travail. Ce qui signifie que l'efficacité est utilisé uniquement pour réduire la charge de travail des ressources, et non la durée de la tâche. Deux niveaux d'efficacité sont considérés dans leur étude : [0.5,1.5]. Walter et al. Walter and Zimmermann (2016) développent également une relation réciproque entre l'efficacité et la durée de la tâche. L'efficacité de chaque ressource est définie dans l'ensemble $\{0.5, 1, 1.25, 1.5\}$.

3.3.2 Problèmes MS-RCPSP préemptifs

Certaines variantes du MS-RCPSP autorisant la préemption des tâches ont également été considérées. Drezet et al. Drezet and Billaut (2008) considèrent qu'une tâche peut être interrompue uniquement si elle correspond à un changement des ressources affectées à cette tâche. Cependant,

il n'est pas possible d'interrompre une tâche si cela va entraîner une augmentation de sa durée. Toutes les tâches sont soumises à des dates de fin souhaitées et des dates de début au plus tôt. Ainsi, les besoins des tâches en ressources sont définis par la quantité minimale de ressources et la quantité maximale nécessaires pour exécuter la tâche. Une partie du problème consiste à déterminer la quantité de ressources affectées à chaque tâche pour chaque période de temps. Cependant, la durée de la tâche est une donnée qui ne dépend pas de la quantité de ressources affectées à celle-ci. L'objectif dans leur modèle est de minimiser le retard algébrique maximum L_{max}. Dhib et al. (Dhib et al. (2015) s'intéressent à un modèle MS-RCPSP rencontré dans une société de services informatiques. Les auteurs développent une formulation PLNE, ainsi que des heuristiques basées sur des règles de priorité comme des approches de résolution différentes. Dans Maghsoudlou et al. (2019), les tâches sont soumises à des dates de fin strictes qui doivent être strictement respectées, et à des dates de fin légères qui peuvent être violées avec des pénalités liées à l'avance ou au retard. Plusieurs ressources peuvent exercer la même compétence avec le même niveau d'exécution mais des coûts de réalisation différents. La préemption d'une tâche donnée est associée à une pénalité de reprise. L'objectif dans leur modèle est de minimiser le coût total du projet qui inclut les pénalités des avances et des retards des tâches, ainsi que les coûts de préemption. Les auteurs proposent des modèles PLNE pour résoudre les instances de petites tailles. Ils proposent également une métaheuristique basée sur un algorithme de colonies de fourmis (ACO) pour traiter les instances de grandes tailles. Les auteurs de Polo-Mejía et al. (2019) s'intéressent à un modèle MS-RCPSP pour gérer les activités de recherche dans un laboratoire nucléaire. L'ensemble des activités comprend à la fois des activités préemptives et d'autres non-préemptives (critiques). Le problème est modélisé sous forme de deux différentes variantes du MS-RCPSP. Dans la première variante, la préemption est autorisée avec une pénalité à chaque fois qu'une activité critique est interrompue. Les auteurs proposent un modèle PLNE ayant pour objectif la minimisation du makespan ainsi que les pénalités de préemption. La seconde variante intègre pour la première fois ce que les auteurs appel la préemption partielle. Ce concept interdit dans certains cas la libération des ressources si l'activité est interrompue. Un modèle PLNE et un modèle de programmation par contraintes (PPC) ont été adoptés pour résoudre les instances de petites tailles. Les auteurs développent également une heuristique gloutonne (ou greedy heuristic (GH)) pour traiter les instances de tailles réelles.

3.3.3 Synthèse

Le Tableau 3.1 présente une synthèse des études qui se concentrent soit sur un problème d'ordonnancement de projet à contraintes de ressources (RCPSP) ou soit sur une de ses extensions spécifiques. La première colonne représente l'article. La deuxième colonne indique le type du problème traité ("S" pour un problème RCPSP standard, "Pm" pour un modèle preemptif, "MS" pour un problème RCPSP multi-compétence). La troisième fournit les principales caractéristiques des tâches et des ressources ("TypeR" pour le type de ressources (s'il s'agit de ressources renouvelables "R", ou non renouvelables "RN"), "#R/C" pour le nombre de ressources ou de compétences nécessaires pour réaliser la tâche, "NH" pour un problème MS-RCPSP avec des niveaux de compétence hiérarchiques, "NM" pour un problème MS-RCPSP avec un niveau minimum de compétence pour exécuter la tâche). La quatrième colonne "MObj" pour indiquer un problème muli-objectif. La cinquième colonne "Obj(min)" indique l'objectif(s) considéré(s) dans l'article. La sixième montre les méthodes de résolutions proposées, suivis des instances utilisées dans la colonne 7. La dernière colonne indique si l'étude considère d'autres contraintes spécifiques non indiquées dans la colonne 2.

Tableau 3.1 utilise de nombreux acronymes et abréviations pour indiquer soit l'objectif ou soit la méthode de résolution. Concernant l'objectif : C_{max} : la durée du projet (ou makespan), f : le coût de réalisation du projet, L_{max} : le retard algébrique maximum, wjT_j : le retard absolu pondéré, A : pour tout autre objectif. Concernant les méthodes de résolution : GP-HH : *genetic programming hyper − heuristic*, GA : *genetic algorithm*,

P-GWO/FA : $pareto-based\ \ grey\ \ wolf\ \ optimizer/fibonacci$ algorithm, MILP : $mixed-integer\ \ linear\ \ program$, CP : $constraint\ \ programming$, GH : $greedy\ \ heuristic$, GP-HH : $genetic\ \ programming\ \ hyper-heuristic$, CTMC : $continuous-time\ \ markov\ \ chain$, MA : $memetic\ \ algorithm$, NSGA-II : $non-dominated\ \ sorting\ \ genetic\ \ algorithm-II$, ACO : $ant\ \ colony\ \ optimization$, DOMVO : $discrete\ \ oppositional\ \ multi-verse\ \ optimization$, PR : $path-relinking$, DEGR : $differential\ \ evolution\ \ and\ \ greedy\ \ algorithm$, FOA : $fruit\ \ fly\ \ optimization\ \ algorithm$, CS : $cuckoo\ \ search$, TLBO : $teaching–learning-based\ \ optimization$, BB : $Branch-and-Bound$, BC : $branch-and-cut$, PSO : $particle\ \ swarm\ \ optimization$, SA : $simulated\ \ annealing$ LPNS : $LP-based\ \ neighborhood\ \ search$.

Les problèmes d'ordonnancement présentés ci-dessous consistent tous à produire un ordonnancement de tâches qui utilisent un ou plusieurs types de ressources. La principale difficulté de ce type de problème est généralement liée au partage des ressources limitées par les tâches d'un même projet. Cependant, dans de nombreuses situations réelles, les ressources ne sont pas seulement partagées par des tâches du même projet, mais aussi par d'autres tâches appartenant à différents projets. En fait, cet aspect "partage des ressources" n'apparaît, comme nous le verrons dans la section ci-dessous, que dans un contexte multi-projet.

3.4 RCPSP multi-projet

Les problèmes d'ordonnancement traités dans le cadre de cette thèse sont tous rencontrés dans un environnement multi-projet. Cette section vise alors à situer ces problèmes par rapport à la littérature d'ordonnancement multi-projet. Il convient de noter que nous nous sommes basés sur les récentes revues de la littérature publiées par Hartmann et al Hartmann and Briskorn (2021) et Eynde et al Eynde and Vanhoucke (2020) pour élaborer cette partie de l'état de l'art.

En général, l'ordonnancement multi-projet consiste à planifier dans le temps l'exécution de multiples projets, qui partagent souvent un ou plusieurs types de ressources (machines, humaines, etc.). Résoudre un tel problème implique de trouver une solution qui satisfait non seulement les contraintes liées à chaque projet séparément (par exemple, contraintes de précédence, dates d'échéance, contraintes budgétaires), mais aussi celles liées aux partages des ressources entre ces projets. De ce fait, l'ordonnancement d'un projet particulier n'est pas indépendant de l'ordonnancement des autres projets. Ce qui rend le processus de résolution beaucoup plus complexe. Ce type de problèmes d'ordonnancement est référencé dans la littérature par problèmes d'ordonnancement multi-projet à contraintes de ressources ou RCMPSP (pour Resource-Constrained Multi-project Scheduling Problem).

En termes simples, le RCMPSP est défini comme une généralisation du RCPSP, dans laquelle plusieurs projets doivent être panifiés en parallèles sous contraintes de ressources limitées. Le RCMPSP, en tant que généralisation du RCPSP, est également $\mathcal{NP}$-difficile au sens fort Blazewicz et al. (1983a). Deux différentes approches ont été utilisées pour traiter cette extension multi-projet. Une approche à projet unique, utilisant des tâches fictives et des arcs de précédence pour combiner tous les projets en un seul méga-projet, réduisant par la suite le RCMPSP à un problème RCPSP. Une seconde approche multi-projet maintient les détails de chaque projet séparés des autres projets. On note que dans cette section, nous nous intéressons plus particulièrement aux problèmes considérant cette seconde approche, étant la plus considérée par les chercheurs intéressés par ce type de problème d'ordonnancement.

Auteurs	Problème			Tâches et Ressources			MObj	Obj(min)	Méthodes	Jeux de données	Autres
	S	Pm	MS	# R/C	NH	NM					
Zhu et al. (2021)			•	1	•	•	•	C_{max}&C	GP-HH	iMOPSE	
Snauwaert and Vanhoucke (2021)			•	≥ 1	•			C_{max}	GA-hybride	Propre(RanGen))	
Hossein and Baradaran (2020)			•	≥ 1	•	•	•	C_{max}&C	P-GWO/FA	iMOPSE	Effet de détérioration et des c
Lin et al. (2020)			•	1	•	•		C_{max}	GP-HH	iMOPSE	
Creemers (2019)	•	•		≥ 1				C_{max}	CTMC	PSPLIB	Des durées stoch
Dai and Cheng (2019)			•	1	•			C_{max}	MA	Propre/iMOPSE	Des détériorations
Laszczyk and Myszkowski (2019)			•	1			•	C_{max}&C	NSGA-II	iMOPSE	
Maghsoudlou et al. (2019)		•	•	≥ 1				C	MILP/ACO	Propre(ProGen)	
Polo-Mejía et al. (2019)		•	•	≥ 1				C_{max}	MILP/CP/GH	Propre	
Vanhoucke and Coelho (2019)	•	•		≥ 1				C_{max}	metaheuris	Propre	Avec setup t
Zhu et al. (2019)			•	1	•	•		C_{max}	DOMVO	iMOPSE	
Myszkowski et al. (2018)			•	≥ 1	•	•		C_{max}	DEGR	iMOPSE	
Wang and long Zheng (2018)			•	1	•	•	•	C_{max}&f	FOA	iMOPSE	
Maghsoudlou et al. (2017)			•	≥ 1	•	•	•	f&A	CS	Propre(RanGen)	
yu Zheng et al. (2017)			•	1	•	•		C_{max}	TLBO	iMOPSE	
Almeida et al. (2016)			•	≥ 1				C_{max}	Heuris	Correia et al., (2012)	
Chen and Zhang (2016)	•	•		≥ 1				A	Modèle math	Propre(Patterson)	Durées aléatoire
Cheng et al. (2015)		•		≥ 1				C_{max}	BB, Heuris	Propre(PSPLIB)	multi-mod
Dhib et al. (2015)		•	•	≥ 1				C_{max}	MILP, Heuris	Propre(PSPLIB)	
Moukrim et al. (2015)	•	•		≥ 1				C_{max}	B&P	PSPLIB	
Shou et al. (2015)	•	•		≥ 1				C_{max}	PSO	Propre(PROGEN)	
Afshar-Nadjafi and Majlesi (2014)	•	•		≥ 1				C_{max}	MILP/GA	PSPLIB	Avec setup t
Correia and da Gama (2014)			•	≥ 1				f	MILP	Correia et al., (2012)	
Montoya et al. (2014)			•	≥ 1				C_{max}	B&P	Propre(PSPLIB)	
Koné et al. (2011)	•			≥ 1				T	MILPs	PSPLIB/Baptiste et al. (2000)/Carlier et al. (2003)/Propre	
Xiao et al. (2013)			•	≥ 1			•	C_{max}&C	ACO	Propre	
Yannibelli and Amandi (2013)			•	≥ 1	•		•	M/A	SA-GA	Propre(PSPLIB)	
Correia et al. (2012)			•	≥ 1				C_{max}	MILP/Heuris	Propre(PSPLIB)	
Fırat and Hurkens (2012)			•	≥ 1	•	•		C	MILP	Propre	
Zhu et al. (2011)	•	•		≥ 1				C_{max}	GA	Propre(PROGEN)	
Valls et al. (2009)			•	≥ 1	•		•	A	GA-H	Propre	Calendrier des tra
Drezet and Billaut (2008)	•		•	≥ 1				L_{max}	MILP/TS	Propre	Modification de l'affectation
Haitao and Keith (2008)			•	≥ 1				C	MILP/CP	Propre(ProGen/max)	Décalages temporels minim
Vanhoucke and Debels (2008)	•	•		≥ 1				C_{max}	BB	Propre	Fast-tracking, durées de
Bellenguez and Néron (2007)			•	≥ 1				C_{max}	BB	Propre	
Damay et al. (2007)	•	•		1				C_{max}	LPNS	PSPLIB	
Bellenguez and Néron (2005)			•	≥ 1	•	•		C_{max}	Bornes inf	Propre(PSPLIB)	
Néron and Baptista (2002)			•	≥ 1				C_{max}	Bornes inf	/	

Tab. 3.1 : Aperçu de la littérature sur des problèmes d'ordonnancement de projet à contraintes de ressources

3.4.1 RCMPSP standard

Browning et al. Browning and A.Yassine (2010) adressent une version standard du RCMSP avec plusieurs fonctions objectifs, incluant le retard moyen par projet ainsi que le pourcentage moyen de retard par projet. Les auteurs comparent entre les règles de priorité les plus populaires (exactement 20 règles de priorité), qui sont toutes basées sur un schéma de génération parallèle (ou parallel schedule generation scheme (PSGS)). Dans Wang et al. (2017d), les auteurs étendent le travail de Browning and A.Yassine (2010) en intégrant des incertitudes sur les durées de tâches. L'auteur de Singh (2014) s'intéresse à une version RCMPSP qui intègre des pénalités de retard liées aux priorités des projets. La fonction objectif cherche à minimiser le makespan de chaque projet, mesuré par rapport au makespan de son chemin critique non-contraint, ainsi que les coûts des pénalités des projets. En se basant sur différents critères, la priorité de chaque projet est calculée en utilisant un processus hiérarchique analytique (AHP). Pour la résolution du problème, une heuristique hybride basée sur des règles de priorité et le AHP est présentée. Dans Pérez et al. (2016), les auteurs considèrent également une extension RCMPSP standard. Deux objectifs ont été abordés dans leur étude. Le premier minimise la durée globale de tous les projets (makespan global), qui revient à minimiser la date de fin du dernier projet. Le deuxième minimise le pourcentage du retard moyen, qui revient à minimiser la somme du pourcentage moyen de retard de tous les projets. Le retard moyen de chaque projet est mesuré par rapport à la durée de son chemin critique. Les auteurs présentent des algorithmes génétiques multi-modaux (ou multi-modal genetic algorithms (MMGAs)). Dans Gonçalves et al. (2015), les auteurs considèrent un modèle standard du RCMPSP avec plusieurs mesures de performances intégrées en une seule fonction objectif pondérée. Cette fonction objectif inclut la minimisation de la somme des retards des projets, des avances des projets et des déviations des durées des projets par rapport aux longueurs de leurs chemins critiques. Pour la résolution du problème, les auteurs développent un algorithme génétique à clé aléatoire biaisée (ou biased random-key genetic algorithm (BRKGA)). Un modèle similaire est considéré dans Gonçalves et al. (2008), intégrant des dates de début et de fin affectées aux projets. Les auteurs présentent un algorithme génétique pour la résolution du problème. Les auteurs de He et al. (2020) s'intéressent à un problème RCMPSP dans lequel les tâches sont associées à des coûts de réalisation qui s'ajoutent aux sorties de trésorerie, tandis que la réalisation de certaines tâches entraîne des coûts payés par le client qui s'ajoutent aux entrées de trésorerie. L'objectif est de minimiser l'écart maximum entre les sorties et les entrées de trésorerie. Trois métaheuristiques sont présentées pour résoudre le problème : une recherche tabou (TS), un recuit simulé (SA) et un algorithme hybride qui combine les deux métaheuristiques (TS-SA). Les auteurs de F. and Z. (2018) envisagent un modèle RCMPSP dans lequel les projets sont gérés par des différents chefs de projets et sont menés en parallèle sur des différents sites. Leur modèle considèrent deux types de ressources renouvelables. Des ressources locales (c'est-à-dire spécifique à chaque projet) et des ressources globales (partagées par tous les projets). Chaque projet a une date de début au plus tôt. L'objectif est de minimiser les coûts totaux de retard des projets. Le retard de chaque projet est mesuré par rapport à la durée de son chemin critique. La durée du chemin critique du projet correspond à son makespan obtenu en exécutant chaque tâche le plus rapidement possible tout en négligeant les contraintes sur les ressources. Dans Adhau et al. (2012), un modèle similaire est également considéré, ayant pour objectif de maximiser le revenu des gestionnaires. Dans Ben Issa et al. (2021), les auteurs adressent des modèles RCMPSP sous différentes hypothèses liées aux tâches. Le premier modèle considère le cas standard ou les tâches sont non preemptives et ont des durées et des besoins en ressources fixes. Le second modèle relâche la contraintes de non-préemption. Ce dernier modèle suppose que les tâches peuvent être exécutées en utilisant des ressources flexibles sur des durées flexibles et qui peuvent être interrompues ou non. L'objectif est de minimiser le makespan total. Une formulation PLNE est développée pour définir le problème qui combine les trois modèles. Des heuristiques sont également présentées pour sa résolution.

3.4.2 RCMPSP multi-mode

Le RCMPSP multi-mode, également appelé M-RCMPSP (pour Multi-mode Resource-Constrained Multi-Project Scheduling Problem), ajoute à RCMPSP standard l'hypothèse qu'une tâche peut être accomplie selon des multiples modes d'exécution, avec une durée et des besoins en ressources spécifique pour chaque mode. Toutefois, une tâche doit être réalisée tout au long de son exécution selon un seul mode. Ce qui rend le M-RCMPSP encore plus compliqué que le RCMPSP. L'objectif dans le M-RCMPSP est de trouver un mode d'exécution ainsi qu'une date de début pour chaque tâche, tout en satisfaisant les contraintes de précédence et des ressources sachant que la durée totale du projet doit être minimisée. Contrairement au RCMPSP standard, qui considère uniquement des ressources renouvelables, les deux types de ressources renouvelables et non renouvelables sont souvent considérés dans le M-RCMPSP.

Cette extension a fait l'objet du défi MISTA 2013 Wauters et al. (2016), et plusieurs algorithmes soumis ont été publiés. Les solutions obtenues sont évaluées par une fonction objectif pondérée à deux composants, à savoir la minimisation du retard total et du makespan total des projets. Le retard d'un projet est mesuré par rapport à la durée de son chemin critique.

Les auteurs de Asta et al. (2016), les lauréats du défi proposent une approche combinant un algorithme monte carlo et hyper-heuristique, ainsi que plusieurs opérateurs de voisinages. L'auteur de Geiger (2017) considère le même problème et il présente une recherche locale multi-thread pour sa résolution. Toffolo et al. Toffolo et al. (2016) proposent une heuristique hybride basée sur un modèle de programmation en nombres entiers (ou integer programming (IP)), ainsi qu'une recherche locale pour trouver des solutions approchées. Les auteurs de Araujo et al. (2020) considèrent également un problème M-RCMPSP. Une méthode des plans sécants (ou cutting plane algorithm (CPA)) est proposée pour minimiser le retard total et le makespan global des projets. Dans Beşikci et al. (2015), les auteurs abordent un modèle M-RCMPSP avec un budget global dédié à la réalisation de tous les projets. L'utilisation de chaque ressource est associée à un coût donné qui sera déduit du budget. Une partie de la solution consiste à déterminer la disponibilité globale des ressources ainsi qu'une répartition de cette disponibilité entre les projets. Les projets sont associés à des dates de fin souhaitées et l'objectif est de minimiser la somme des retards pondérés des projets.

3.4.3 La sélection et l'ordonnancement des projets

Le problème de la sélection et de l'ordonnancement consiste à sélectionner un portefeuille optimal de projets parmi plusieurs projets disponibles, en tenant compte plusieurs contraintes (budget, capacité de ressources, deadlines des projets, par exemple). Ainsi, après avoir sélectionné un portefeuille de projets, l'étape suivante est de les ordonnancer pour optimiser une ou plusieurs mesures de performances. Le problème combiné de la sélection et de l'ordonnancement des projets a récemment fait l'objet d'une attention particulière dans la littérature spécialisée.

Les auteurs de Shariatmadari et al. (2017) s'intéressent à un problème de sélection et de l'ordonnancement multi-projet, ayant pour objectif la maximisation du budget (ou cash flow) restant à la fin de l'horizon de planification. Pour la résolution du problème, les auteurs présentent un algorithme de recherche gravitationnelle (ou Gravitational Search Algorithm (GSA)). Dans Kumar et al. (2018), les auteurs intègrent deux types d'interdépendance entre les projets : l'exclusivité mutuelle (c'est-à-dire un seul projet au maximum parmi un ensemble de projets peut être sélectionné) et la complémentarité (c'est-à-dire un ensemble de projets connexes doivent être sélectionnés en même temps). Les ressources ont des disponibilités variables dans le temps. Les projets sont considérés à un niveau agrégé (c'est-à-dire qu'ils ne sont pas divisés en tâches). Trois métaheuristiques : TLBO, TS et une hybride TLBO-TS ont été développées, ayant pour objectif la maximisation du bénéfice total attendu des portefeuilles de projets sélectionnés. Le bénéfice attendu d'un projet est considéré comme dépendant du temps. Cela signifie qu'un retard dans la réalisation d'un projet en-

traînera une diminution du bénéfice attendu. Un modèle similaire est considéré dans Liu and Wang (2011) incluant des relations de précédence entre les projets et des limites budgétaires variables dans le temps. Tofighian et al. Tofighian and Naderi (2015) considèrent également un modèle similaire bi-objectif. L'un des objectifs maximise le bénéfice total attendu, tandis que l'autre minimise la variation de l'utilisation des ressources. Pour résoudre le problème, les auteurs présentent une formulation PLNE ainsi qu'un algorithme à base de colonies de fourmis artificilles multi-objectif.

3.4.4 RCMPSP multi-compétence

En raison de la complexité de ce type de problèmes d'ordonnancement, peu d'études ont été menées sur l'ordonnancement multi-projet et l'affectation des ressources humaines multi-compétences aux tâches des projets. Wu et al. Wu and Sun (2006) envisagent un problème RCMPSP multi-compétence avec des niveaux d'efficacité hétérogènes. L'efficacité de compétences des ressources est variable dû au phénomène d'apprentissage, c'est-à-dire que l'efficacité des ressources augmente du fait de la répétition des tâches identiques ou similaires. Le modèle étudié suppose qu'une stratégie d'externalisation sera appliquée chaque fois qu'une tâche ne peut pas être achevée avant sa date d'échéance. L'objectif est de minimiser les coûts d'externalisation. Un algorithme génétique (GA) est proposé pour résoudre le problème. Heimerl et al. Heimerl and Kolisch (2010a) et Kolisch et al Kolisch and Heimerl (2012) présentent respectivement une formulation PLNE et un algorithme génétique hybride combiné avec une recherche tabou pour un problème d'ordonnancement multi-projet et d'affectation des ressources humaines multi-compétences à la charge de travail. Les ressources ont des niveaux de compétences variables qui influencent la charge des projets. Heimerl et al. subdivisent les projets en lots plutôt qu'en tâches avec une date de début au plus tôt et une date de fin souhaitée associées à chaque projet. Tandis que Kolisch et al. considèrent les mêmes paramétrages (c'est-à-dire dates de début au plus tôt et de fin souhaitées) mais associés aux tâches. Aussi chaque tâche a exactement un prédécesseur lié à elle par une relation de précédence de type start-to-start. L'objectif dans les deux études est la minimisation des coûts d'utilisation des ressources internes et externes qui s'accumulent en travaillant sur les tâches des projets. Le même modèle est étendu récemment dans Felberbauer et al. (2019) en considérant une version stochastique du problème. Dans cette version, Chaque tâche peut être constituée de plusieurs lots de travaux, où chaque lot de travail requiert une quantité stochastique d'efforts exercés dans une compétence spécifique. Deux approches de résolution différentes sont présentées. La première approche est une matheuristique basée sur une recherche locale et l'algorithme Frank-Wolfe. La deuxième approche utilise un simple modèle d'approximation qui a abouti à une formulation PLNE résolue par le solveur CPLEX. Walter et al. Walter and Zimmermann (2016) envisagent un modèle d'ordonnancement multi-projet et d'affectation des ressources multi-compétences à la charge des projets. Les projets sont décomposés en charges nominales. La structure organisationnelle de l'entreprise est représentée par un ensemble de département, avec chaque ressource étant membre d'un seul département. Dans chaque département, la charge de travail départemental doit être accomplie à chaque période par les ressources de ce département. Comme dans le cas des problèmes traités dans cette thèse, les ressources ont des compétences hétérogènes et leur efficacité peut augmenter ou diminuer la charge nominale de travail. L'objectif est d'affecter chaque équipe (sous ensemble de ressources) aux charges de chaque projet, de telle sorte que la taille moyenne de l'équipe soit minimisée. Des algorithmes heuristiques sont proposées pour la résolution du problème. Les auteurs de Gutjahr et al. (2010) s'intéressent à une extension multi-objectif du modèle RCMPSP multi-compétence. Leur modèle prend en considération des ressources renouvelables de disponibilités variables. En outre, l'efficacité des compétences est dynamique (c'est-à-dire qu'elle augmente par l'apprentissage et diminue par l'oubli). De plus, les tâches sont soumises à des dates de début au plus tôt et à des dates de fin impératives. Les objectifs considérés sont la maximisation des gains économiques des projets sélectionnés ainsi que l'augmentation de l'efficacité des ressources grâce à l'effet d'apprentissage. Afin de déterminer les solutions Pareto-optimales, le problème est décomposé en deux sous

problèmes. Un problème maître traitant la sélection du portefeuille des projets. Pour sa résolution, les auteurs présentent et comparent des algorithmes multi-objectifs : NSGA-II et P-ACO (pour Pareto ant colony optimization). Le deuxième problème "esclave", linéarisé par une approximation asymptotique linéaire, résout l'affectation des ressources aux charges des projets sélectionnés. Chen et al. Chen et al. (2017) traitent également une extension multi-objectif axée sur l'ordonnancement des projets informatiques. Les auteurs considèrent que l'efficacité des compétences des ressources est influencée par les phénomènes d'apprentissage et d'oubli (learning and forgetting effects). Ils développent une relation réciproque entre la durée des tâches et l'efficacité des compétences. Trois fonctions objectifs sont considérées dans leur modèle : la maximisation du niveau d'efficacité des compétences des ressources à la fin du développement des projets, la minimisation du temps d'exécution de tous les projets, et la minimisation des coûts des projets (correspondant à la somme des salaires de tous les ressources sélectionnées). Un algorithme génétique de type NSGA-II est développé pour résoudre le problème traité. Les auteurs de Hematian et al. (2020) considèrent un modèle plus ou moins similaire intégrant des incertitudes liées aux durées des tâches. Une tâche nécessite plusieurs compétences avec un niveau minimum par compétence. L'efficacité des compétences des ressources augmente en fonction du phonème d'apprentissage. L'objectif est de minimiser le makespan des projets ainsi que le coût total de réalisation des projets. Un modèle mathématique flou est présenté. Les instances de petites tailles sont résolues par le logiciel GAMS. Le livre de Walter (2015) couvre trois versions du RCMPSP multi-compétence : la sélection et l'ordonnancement des projets, la composition de petites équipes de projet et l'équilibration de la charge de travail entre les ressources. Ils considèrent des ressources humaines multi-compétences avec des niveaux de compétences hétérogènes. Dans l'article récent publié par Cui et al. (2021), les auteurs considèrent un modèle RCMPSP intégré à la fois à un problème multi-mode et un autre multi-compétence. Chaque tâche a plusieurs modes d'exécution, et doit être affectée à une seule ressource multi-compétence. Chaque projet à une date de début au plus tôt et l'objectif est de minimiser le makespan total (c'est-à-dire le temps de réalisation de la dernière tâche de l'ensemble des projets). Un algorithme de recherche de voisinage variable (ou Variable Neighborhood Search Algorithm (VNS) est développé pour résoudre le problème.

3.4.5 Synthèse

Le problème d'ordonnancement abordé dans la première partie du manuscrit, diffère de ceux déjà traités dans la littérature par les points suivants. Nous considérons que les compétences hétérogènes des ressources ont une influence sur la durée de la tâche. Malgré le fait que cette spécificité est beaucoup plus réaliste à plusieurs égards, peu d'études sur le RCMPSP multi-compétence l'ont prise en compte. Comme mentionné précédemment dans la Section 3.3.1, les articles de Heimerl and Kolisch (2010a) et Kolisch and Heimerl (2012) considèrent une relation réciproque entre l'efficacité des ressources et la vitesse de l'exécution de la charge des projet. Cependant, les auteurs utilisent l'aspect de l'efficacité pour réduire la charge du travail régulière et supplémentaire des ressources, tout en négligeant la durée de tâche. Gutjahr et al. Gutjahr et al. (2010) et Chen et al. Chen et al. (2017) considèrent des modèles qui incluent des compétences dynamiques et les effets d'apprentissage. Le degré auquel une ressource possède une certaine compétence augmente lorsque cette compétence est utilisée pendant l'exécution d'une tâche. En revanche, il diminue lorsqu'elle n'est pas utilisée. Le degré de compétence est lié à l'efficacité avec laquelle une ressource peut réaliser une tâche. Cependant, le modèle traité par Gutjahr et al. est intégré dans un problème de sélection et d'ordonnancement des portefeuilles de projets. Tandis que, le modèle abordé par Chen et al. est intégré dans un environnement d'ordonnancement des projets informatiques, comme c'est le cas dans notre problème. Cependant, Chen et al. considèrent qu'une ressource (employé) multi-compétence ne doit exercer qu'une seule compétence dans le même projet. De plus, une compétence requise par un projet doit être exécutée par un seul employé afin de maintenir la continuité du travail. Dans Hematian et al. (2020), les auteurs prennent en compte l'efficacité des ressources dans

les durées des tâches qui leur sont assignées. De plus, l'efficacité des ressources augmente durant le processus de l'exécution due à l'effet de l'apprentissage. Contrairement à notre modèle, les auteurs supposent des incertitudes sur les durées des tâches.

Le deuxième aspect caractérisant notre modèle est la préemption. Nous pouvons constater très clairement à partir du tableau que cet aspect n'a pas encore été pris en compte dans un contexte d'ordonnancement multi-projet et d'affectation des ressources multi-compétences aux tâches des projets. Le troisième aspect est lié aux autres hypothèses et contraintes caractérisant les tâches et les ressources, qui rendent notre modèle encore beaucoup plus réaliste. Nous pouvons citer les contraintes sur la quantité qui pourra être réalisée de certaines tâches dans une semaine. Plus précisément, les charges minimales et maximales qui simulent la situation dans laquelle le chef de projet souhaite qu'une tâche avance régulièrement plus tôt que par bloc. Nous pouvons également citer, la contrainte sur le nombre de tâches différentes sur lesquelles un employé pourra travailler dans la semaine. Il s'agit surtout de limiter les changement de contexte pour un employé. Au lieu de prendre en compte des setup entre les tâches, le chef de projet préfère limiter le nombre de tâches par semaine pour chaque employé.

Auteurs	Problème				Tâches et Ressources				MObj	Obj(min)	Méthodes	Jeux de données	
	S	Pm	MM	MS	TypeR	# R/C	NH	NM					
Problème dans le chapitre 4.2		•		•	R	1	•			$\min(w_j T_j)$	MILP,TS,LS,MCF	Propre	
Ben Issa et al. (2021)	•	•			R	≥ 1				min(TMS)	Propre(PSPLIB)	Heuris	
Cui et al. (2021)			•	•	R	1				min(TMS)	Données réelles	VNS	
Felberbauer et al. (2019)													
Browning and A.Yassine (2010)	•				R	1				A	20 PRs	Propre	
Pérez et al. (2016)	•				R	≥ 1				T/D	MMGAs	RCMPSPlib	
Gonçalves et al. (2015)	•				R	≥ 1				A	BRKGA	Gonçalves et al. (2008)	
Toffolo et al. (2016)	•		•		R-NR	≥ 1				T	Heuris	MISTA	
Araujo et al. (2020)			•		R-NR	1				A	CPA	MISTA/PSPLIB	
Shariatmadari et al. (2017)	•				R-RN	1				A	GSA	Propre(PSPLIB)	
Gutjahr et al. (2010)				•	R	≥ 1	•	•	•	A	NSGA-II/P-ACO/LP	Propre	Compétences dynamiques,
Kumar et al. (2018)	•				R	-				max(B)	TLBO/TS/TLBO-TS	Propre(Tofighian et al. (2015))	Contraintes
Liu and Wang (2011)	•				R	-				max(B)	CP	Étude de cas	Contraintes
Tofighian and Naderi (2015)	•				R	-			•	max(B),min(A)	MILP/MPACO	Propre	Contraintes
F. and Z. (2018)	•				R	≥ 1				min(TC)	AG/SGBNM	MPSPLIB	Glob
Adhau et al. (2012)	•				R	≥ 1				min(APD,TMS)	Heuris	Homberger (2009)	Glob
Geiger (2017)			•		R-NR	≥ 1			•	min(TPD,TMS)	M-TS	MISTA/MMLIB	
Asta et al. (2016)			•		R-NR	≥ 1			•	min(TPD,TMS)	M-TS	MISTA/MMLIB	
Beşikci et al. (2015)			•		R-NR	≥ 1				$\min(w_j T_j)$	GA/MGA	Propore(PSPLIB)	
Kolisch and Heimerl (2012)				•	R-NR	≥ 1	•			min(C)	GA-TS	Propre	
Heimerl and Kolisch (2010b)				•	R-NR	≥ 1	•			min(C)	MILP	Propre	
Wu and Sun (2006)				•	R	≥ 1	•			min(C)	GA	Propre	
Browning and A.Yassine (2010)	•				R	≥ 1				min(A)	PRs	Propre	
Gonçalves et al. (2008)	•				R	≥ 1				min(A)	GA	Propre	
Singh (2014)	•				R	≥ 1				min(T,C)	Heuris	Étude de cas	
Chen et al. (2017)				•	R	1	•		•	A	Données réelles	NSGA-II	
Hematian et al. (2020)				•	R	≥ 1	•	•	•	min(T,C)	Propre	Modèle math	
He et al. (2020)	•				R	≥ 1				min(A)	TS/SA/TS-SA	Browning et al. (2010)	
Browning and A.Yassine (2010)	•				R	1				A	20 PRs	Propre	
Walter and Zimmermann (2016)				•	R	-	•			min(A)	heuris	Propre	

Tab. 3.2 : Aperçu de la littérature sur des problèmes d'ordonnancement multi-projet à contraintes de ressources

Chapitre 4

Problème d'ordonnancement multi-projet sous contraintes de ressources multi-compétences

Contents

4.1 Présentation du problème

Le problème étudié a été soulevé par notre partenaire informatique (Alfa-conseil), qui partage des employés multi-compétences entre plusieurs projets pour assurer leur réalisation simultanée.

Le modèle proposé vise à se rapprocher le plus possible de l'organisation de cette entreprise, que ce soit au niveau de la modélisation globale du projet ou de la façon dont les compétences sont utilisées et les tâches sont modélisées. Il convient de noter que la confidentialité des données doit être préservée. Le tableau 4.1 montre la notation des paramètres du problème utilisés tout au long de ce chapitre.

4.1.1 Structure globale du projet

Soit $K = \{k_1, \ldots, k_L\}$ un ensemble des L projets devant être réalisés dans un horizon de planification fixé. Chaque projet $k_l \in K$ est décomposé en un ensemble de tâches indépendantes et préemptives, avec des dates de début au plus tôt et de fin souhaitées associées aux différentes tâches. Chaque projet nécessite plusieurs compétences ; et chaque compétence est exercée par plus d'un employé. L'horizon de planification consiste en H semaines consécutives, et chaque semaine est divisée en 10 demi-journées de temps de travail. On note qu'une solution est considérée comme infaisable si les tâches ne peuvent pas être réalisées dans l'horizon de planification. L'unité de temps est donc la demi-journée. Dans notre modèle, les relations de précédence ne sont pas considérées, ni entre les projets, ni entre les tâches. Ceci est dû au fait que les projets réalisés sont à gros grains.

4.1.2 Les caractéristiques des employés

Soit $E = \{e_1, \ldots, e_I\}$ un ensemble de I employés multi-compétences travaillant dans l'entreprise. Chaque employé a une disponibilité par semaine (temps de travail) allant de 0 à 10 demi-journées. Cette disponibilité est connue au début de l'ordonnancement et n'est pas sujette au changement. On désigne par $D_{i,s}$ la disponibilité de l'employé e_i durant la semaine h_s, où $h_s \in H$. De plus, les employés sont affectés aux différents projets avec des pourcentages de temps maximum (quotités) prédéfinis par le décideur. Ainsi, au cours de chaque semaine h_s, tout employé e_i affecté au projet k_l ne peut consacrer à ce projet plus de $D_{i,s} \times Q_{i,l}$, où $Q_{i,l}$ est la quotité maximale de l'employé e_i sur le projet k_l. Les quotités des employés sur les projets varient entre 0 et 100, où une valeur de 0 signifie qu'un employé ne travaille pas sur un projet et 100 signifie que seul ce projet lui est confié. À un instant donné, chaque employé ne peut travailler que sur une seule tâche à la fois.

4.1.3 Modélisation des tâches

Chaque tâche nécessite une seule compétence et doit être attribuée à un seul employé ayant la compétence requise. En outre, il n'est pas possible de changer l'affection de la tâche une fois son exécution commencée. Comme susmentionnée, il n'y a pas de relations de précédence entre les projets ou entre les tâches. Chaque tâche appartient à un seul projet.

Pour chaque tâche n_j, on dispose d'une charge nominale c_j (exprimée en hommes-jours, d'une date de début au plus tôt r_j (donnée en nombre de semaines) et d'une charge minimale notée $c_j^{\min}$ (exprimée en demi-journées) pour quantifier le degré minimal de réalisation de cette tâche par semaine. La charge minimale de la tâche par semaine permet de modéliser certaines tâches qui ne peuvent être interrompues pendant plus d'une semaine. Dans le cas où l'employé affecté à une tâche n_j travaille sur cette tâche pendant une semaine donnée, il doit réaliser au moins sa charge minimale $c_j^{\min}$. Ceci permet de limiter la fragmentation de la tâche sur l'horizon temporel. Durant chaque semaine, l'employé affecté à la tâche n_j ne doit pas dépasser sa charge maximale $c_j^{\max}$. Nous voulons éviter la perte de temps due au changement de contexte des employés qui est nécessaire lors du passage d'une tâche à une autre. Ainsi, au cours de chaque semaine, le nombre de tâches différentes sur lesquelles un employé travaille est inférieur à une valeur donnée b (fixée pour tous

les projets et tous les employés). Ces deux dernières contraintes sont dites souples, i.e peuvent être violées dans certaines conditions (4.1.5).

4.1.4 Modélisation des compétences et de l'efficacité

Dans notre modèle, une fois qu'une tâche est attribuée à un employé possédant la compétence requise, elle le reste pendant tout le temps d'exécution de cette tâche. Les capacités d'exécution des tâches par les employés sont présentées par une matrice de compétences binaire désignée par m, où $m_{j,i} = 1$ si l'employé e_i maîtrise la tâche n_j, sinon $m_{j,i} = 0$. Cela pour dire que tous les employés ne peuvent pas être affectés à une tâche. De plus, nous supposons que plusieurs employés peuvent avoir des niveaux d'efficacité différents pour une même compétence. Comme chaque tâche ne requiert qu'une seule compétence, nous associons directement le niveau de compétence de l'employé à la tâche. Le chef de projet estime le niveau d'efficacité des employés selon la classification standard du niveau d'expertise : junior, moyen et senior. Sur la base de ces estimations, nous attribuons un coefficient d'efficacité égal à 0, 0.5 et 1 à un junior, un moyen et un senior, respectivement. Ce coefficient est un rapport entre le temps de traitement réel consacré par un employé pour réaliser une tâche et le temps de traitement théorique (charge nominale) nécessaire pour réaliser la tâche correspondante. Ainsi, pour prendre en compte le niveau d'efficacité de l'employé dans le calcul du temps de traitement de la tâche, on suppose une simple formule linéaire entre la charge nominale de la tâche et l'efficacité de l'employé qui y est affecté. Nous appliquons cette formule pour convertir la charge nominale (c_j) de la tâche n_j en durée (temps de traitement, $p_{j,i}$) en fonction du niveau d'efficacité ($v_{j,i}$) de l'employé e_i (e_i maîtrise n_j) : $p_{j,i} = (2 - v_{j,i})c_j$. Comme la charge nominale de la tâche est donnée en nombre de jours, nous la multiplions par 2 pour la convertir en demi-journées. Par exemple, une tâche qui nécessite un développeur java et 2 jours pour être exécutée, pourra être réalisée par un développeur senior en 2 demi-journées (c'est-à-dire la moitié du temps), et par un développeur junior en 4 demi-journées (le temps d'exécution réel, autrement dit sa durée intrinsèque).

4.1.5 Fonction d'objectif

La fonction objectif initiale est définie par $f = \sum_{j=1}^{J} w_j T_j$, où T_j est le nombre de semaines de retard de la tâche n_j et w_j le coût de pénalité de retard pour cette tâche. Nous indiquons par d_j et F_j (toutes deux données en nombre de semaines) la date de fin souhaitée et la date de fin de la tâche n_j, respectivement. Si la tâche n_j prend au moins une demi-journée de la semaine $(d_j + 1)$ avant sa date de fin, elle est en retard d'une semaine ($T_j = 1$).

Les contraintes sur la charge minimale de la tâche par semaine et sur le nombre de tâches différentes sur lesquelles un employé pourra travailler dans la semaine sont des contraintes souples imposées par le gestionnaire pour augmenter la productivité de ses employés. Pour un ordonnancement efficace, ces contraintes souples doivent être prises en considération. Cependant, le gestionnaire permet la violation de ces contraintes lorsqu'il n'est pas possible de trouver une solution réalisable pour le problème. L'approche de la résolution doit minimiser la violation de ces contraintes en réduisant leurs déviations indésirables par rapport à leurs objectifs respectifs. Nous introduisons de nouvelles fonctions pour pénaliser les violations de contraintes souples, puis nous ajoutons ces fonctions à la fonction objectif initiale, formant ainsi une nouvelle fonction objectif finale. Nous décrivons l'objectif final dans la section 4.2. Toutes les autres contraintes (également appelées contraintes dures) doivent être respectées par la solution à proposer.

Nous sommes confrontés à deux problèmes : le premier est de trouver une affectation appropriée des employés aux tâches, et le second est de déterminer un ordonnancement des tâches à accomplir par chaque employé dans l'horizon de planification du projet H. Notons qu'un ordonnancement est défini par une affectation des employés aux tâches ainsi que la charge (ou le nombre de demi-

journées) de chaque employé dédiée à l'exécution de chacune de ses tâches au cours de chaque semaine.

Le problème étudié est un cas général du problème d'ordonnancement de machines parallèles $Qm \mid r_j, pmtn \mid \sum w_j T_j$, qui est $\mathcal{NP}$-difficile (Brucker et al. (1997)). Rappelons que le problème d'ordonnancement sur une seule machine $1 \mid r_j, pmtn \mid \sum w_j T_j$ est également $\mathcal{NP}$-difficile (Labetoulle et al. (1984)), ce qui veut dire que même si les affectations des employés aux tâches sont données, le problème demeure $\mathcal{NP}$-difficile.

	Données générales
H	Horizon de planification des projets
K	Ensemble des projets, $K = \{k_1, \ldots, k_L\}$, $\lvert K\rvert = L$
E	Ensemble des employés, $E = \{e_1, \ldots, e_I\}$, $\lvert E\rvert = I$
N	Ensemble des tâches, $N = \{n_1, \ldots, n_J\}$, $\lvert N\rvert = J$
N_l	Ensemble des tâches appartenant au projet k_l
	Données sur les tâches
c_j	Charge nominale de la tâche n_j (mesurée en jours-homme)
r_j	Date de début au plus tôt de la tâche n_j (exprimée en nombre de semaines)
d_j	Date de fin souhaitée de la tâche n_j (exprimée en nombre de semaines)
$c_j^{\max}$	Charge maximale de la tâche n_j par semaine (mesurée en demi-journées)
$c_j^{\min}$	Charge minimale de la tâche n_j par semaine (mesurée en demi-journées)
w_j	Coût de pénalité de retard de la tâche n_j
F_j	Date de fin de la tâche n_j (exprimée en nombre de semaines)
T_j	Retard de la tâche n_j (exprimé en nombre de semaines)
	Données sur les employés
$D_{i,s}$	Disponibilité de l'employé e_i durant la semaine h_s (en demi-journées)
$Q_{i,l}$	Quotité maximale de l'employé e_i sur le projet k_l (pourcentage du temps)
b	Nombre de tâches sur lesquelles chaque employé peut travailler durant chaque semaine
	Données sur les tâches et les employés
$v_{j,i}$	Niveau d'efficacité de l'employé e_i pour la tâche n_j
$p_{j,i}$	Temps de traitement de la tâche n_j en fonction du niveau d'efficacité de l'employé e_i (en demi-journées)
$m_{j,i}$	1 si l'employé e_i maîtrise la tâche n_j ; 0 sinon
$\overline{v_j}$	Efficacité moyenne des employées pour réaliser la tâche n_j.

Tab. 4.1 : Notations et paramètres du problème

4.1.6 Exemple

Dans cette section, nous introduisons un exemple simplifié du problème traité afin de fixer les idées. Il s'agit de deux projets k_1 et k_2, chacun d'eux étant composé de quatre tâches à planifier sur un horizon de trois semaines consécutives (h_1, h_2, h_3). Deux employés (e_1, e_2) sont affectés aux deux projets. Le tableau 4.2 indique pour chaque tâche n_j : les dates de début au plus tôt et de fin souhaitée (r_j, d_j), la charge nominale c_j, les charges minimales et maximales ($c_j^{\min}$, $c_j^{\max}$), la pénalité de retard (w_j) et le projet k_l auquel cette tâche appartient. Le tableau 4.3 indique pour chaque tâche n_j : l'employé possédant la compétence requise pour la réaliser ($m_{j,i}$), le niveau de l'efficacité de l'employé ($v_{j,i}$), et le temps de traitement en fonction du niveau d'efficacité de l'employé ($p_{j,i}$). Enfin, le tableau 4.4 précise pour chaque employé e_i, sa disponibilité durant chaque semaine ($D_{i,s}$) et sa quotité maximale sur chaque projet ($Q_{i,l}$). Considérons par exemple l'employé e_1. Cet employé est disponible 9 demi-journées pendant la semaine h_1. Pendant cette période, il ne doit pas dépasser 50% (4 demi-journées) (resp. 60%, 5 demi-journées) de son temps sur le projet k_1 (resp. k_2). Nous supposons que chaque employé ne doit pas travailler sur plus de 3 tâches différentes par semaine, c'est-à-dire $b = 3$ pour chacun des deux employées.

Une solution à ce problème consiste à définir l'affectation des employés aux différentes tâches en respectant toutes les contraintes dures et autant que possible les contraintes souples. L'objectif est de minimiser la somme des retards pondérés des tâches. A travers cette affectation, nous recherchons pour chaque employé un ordonnancement adéquat à réaliser durant les trois semaines (c'est-à-dire la quantité de temps réalisée de chaque tâche par semaine).

	n_1	n_2	n_3	n_4	n_5	n_6	n_7	n_8
$(r_j,\ d_j)$	(1,2)	(1,2)	(1,2)	(1,2)	(1,2)	(1,2)	(2,2)	(3,3)
c_j	4	2	4	6	6	2	3	3
$(c_j^{\mathbf{min}}, c_j^{\mathbf{max}})$	(2,6)	(2,5)	(2,6)	(4, 6)	(3,10)	(2,7)	(2,10)	(2,6)
w_j	10	10	20	20	30	30	20	10
k_l	k_1	k_1	k_1	k_1	k_2	k_2	k_2	k_2

Tab. 4.2 : Caractéristiques des tâches.

	n_1	n_2	n_3	n_4	n_5	n_6	n_7	n_8
$m_{j,1}$	1	1	1	0	1	1	0	1
$m_{j,2}$	0	1	1	1	1	1	1	1
$v_{j,1}$	0.5	1	0.5	-	0.5	1	-	0
$v_{j,2}$	-	0.5	1	0.5	1	0.5	0	1
$p_{j,1}$	6	2	6	-	9	2	-	6
$p_{j,2}$	-	3	4	9	-	3	6	3

Tab. 4.3 : Paramètres caractérisant les employés et les tâches.

	e_1	e_2
h_1	9	8
h_2	9	8
h_3	9	10

Tab. 4.4 : La disponibilité des employés sur l'horizon de planification du projet.

	$Q_{1,l}$	$Q_{2,l}$
k_1	50%	60%
k_2	60%	60%

Tab. 4.5 : Les quotités maximales de temps des employés allouées aux projets

Les tableaux 4.7 et 4.6 présentent deux ordonnancements partiels (π_0^1 et π_0^2) pour les employés e_1 et e_2, respectivement, sur une période de trois semaines. Ces ordonnancements respectent les contraintes dures et souples du problème. Pour chaque tâche, nous reportons le temps passé par l'employé qui y est affecté durant la semaine. Par exemple, l'employé e_1 a réalisé 2 demi-journées de la tâche n_1 pendant la semaine h_1. Les tâches n_1, n_2, n_5 et n_7 ont chacune un retard d'une semaine, donc $T_1 = T_2 = T_5 = T_7 = 1$ semaine. Ainsi, le coût total dû à ce retard est de 70.

	h_1	h_2	h_3
n_1	2	0	4 ×
n_3	2	4	0
n_5	3	3	3 ×
n_6	0	2	0

× indique une tâche en retard

Tab. 4.6 : Ordonnancement pour l'employé e_1.

	h_1	h_2	h_3
n_2	0	0	3 ×
n_4	4	5	0
n_7	0	3	3 ×
n_8	0	0	3

Tab. 4.7 : Ordonnancement pour l'employé e_2.

4.2 Formulation mathématique

Cette section présente un modèle de programmation par objectif (goal programming) mixte en nombres entiers (GPNE) pour une résolution à l'optimum du problème étudié. Ce modèle GPNE peut s'écrire comme suit.

Variables de décision

Ce modèle utilise des variables de décision indexées temps.

- $x_{j,i}$: variable binaire égale à 1 si l'employé e_i est affecté à la tâche n_j ; 0 sinon.
- $y_{j,i,s}$: variable entière (allant de 0 à 10) qui indique le nombre de demi-journées effectuées de la tâche n_j par l'employé e_j durant la semaine h_s.
- $z_{j,i,s}$: variable binaire égale à 1 si $y_{j,i,s}$ est supérieur à 0 ; 0 sinon.

- F_j : date de fin de la tâche n_j.
- T_j : retard de la tâche n_j.
- $u^+_{j,s}$: déviation supérieure à $c^{\min}_j$ associée à la tâche n_j et à la semaine h_s.
- $u^-_{j,s}$: déviation inférieure à $c^{\min}_j$ associée à la tâche n_j et à la semaine h_s.
- $o^+_{i,s}$: déviation supérieure à b associée à l'employé e_i et à la semaine h_s.
- $o^-_{i,s}$: déviation inférieure à b associée à l'employé e_i et à la semaine h_s.

Contraintes dures

Les contraintes dures de la formulation sont décrites dans ce qui suit.

$$\sum_{i=1}^{I} x_{j,i} \ = 1, \quad j = 1, \ldots, J \tag{4.1}$$

Cette contrainte garantit qu'un seul employé doit être affecté à chaque tâche.

$$x_{j,i} \ \leq m_{j,i}, \quad j = 1, \ldots, J; \quad i =, \ldots, I \tag{4.2}$$

Cette contrainte garantit qu'un employé ne doit pas travailler sur une tâche qu'il ne maîtrise pas.

$$\sum_{s=r_j}^{H} y_{j,i,s} \ = p_{j,i} * x_{j,i} \quad j = 1, \ldots, J; \quad i = 1, \ldots, I \tag{4.3}$$

Cette contrainte garantit que si un employé est affecté à une tâche, il doit l'exécuter en intégralité.

$$\sum_{i=1}^{I} y_{j,i,s} \ \leq \sum_{i=1}^{I} c^{\max}_j * x_{j,i}, \quad j = 1, \ldots, J; \quad s = r_j, \ldots, H \tag{4.4}$$

Cette contrainte garantit qu'au maximum $c^{\max}_j$ à exécuter de la tâche n_j dans chaque semaine.

$$\sum_{i=1}^{I} z_{j,i,s} \ \leq \sum_{i=1}^{I} x_{j,i} \quad j = 1, \ldots, J; s = r_j, \ldots, H \tag{4.5}$$

$$\sum_{i=1}^{I} 10 * z_{j,i,s} \geq \sum_{i=1}^{I} y_{j,i,s}, \quad j = 1, \ldots, J; \quad s = r_j, \ldots, H \tag{4.6}$$

Les contraintes (4.5) et (4.6) garantissent que chaque tâche est exécutée par le même employé tout au long de son exécution. La contrainte (4.6) permet de vérifier, pour chaque tâche, si l'employé qui y est affecté a travaillé sur cette tâche au cours d'une semaine donnée.

$$\sum_{j=1}^{J} y_{j,i,s} \leq D_{i,s}, \quad i = 1, \ldots, I; \quad s = 1, \ldots, H \tag{4.7}$$

Cette contrainte garantit que, pour une semaine donnée, aucun employé ne doit dépasser sa disponibilité.

$$\sum_{j \in N_l} y_{j,i,s} \leq Q_{i,l} * D_{i,s}, \quad i = 1, \ldots, I; \quad s = 1, \ldots, H; \quad l = 1, \ldots, L \tag{4.8}$$

Cette contrainte garantit que, pour une semaine donnée, aucun employé ne doit dépasser sa quotité sur chaque projet.

$$\sum_{i=1}^{I} y_{j,i,s} \geq \sum_{i=1}^{I} c_j^{\min} * z_{j,i,s}, \quad j = 1, \ldots, J; \quad s = r_j, \ldots, H \tag{4.9}$$

Cette contrainte garantit que si un employé travaille sur une tâche pendant une semaine donnée, il doit effectuer de cette tâche au moins sa charge minimale.

$$\sum_{j=1}^{J} z_{j,i,s} \leq b, \quad i = 1, \ldots, I; \quad s = r_j, \ldots, H \tag{4.10}$$

Cette contrainte garantit qu'au cours d'une semaine donnée, le nombre de tâches différentes sur lesquelles un employé travaille doit être inférieur ou égal au nombre maximum autorisé (b).

$$F_j \geq \sum_{i=1}^{I} s * z_{j,i,s}, \quad j = 1, \ldots, J; \quad s = r_j, \ldots, H; \tag{4.11}$$

$$F_j - T_j \leq d_j, \quad F_j, T_j \geq 0 \tag{4.12}$$

Les contraintes (4.11) et (4.12) déclarent respectivement, pour chaque tâche, la date de fin et le retard.

Contraintes souples

Selon la description du problème, les contraintes (4.9) et (4.10) sont des contraintes souples qui peuvent être violées lorsqu'il n'est pas possible d'obtenir un ordonnancement réalisable. Par conséquent, nous incorporons la possibilité de relaxer ces contraintes souples en ajoutant des variables de déviation dans la formulation. Tandis que, ces variables de déviation sont calculées et ajoutées à la fonction objectif en tant que pénalités. Après l'ajout de ces variable, les contraintes souples décrites précédemment dans les équations (4.9) et (4.10) sont devenues, respectivement :

$$\sum_{i=1}^{I} y_{j,i,s} - u_{j,s}^{+} + u_{j,s}^{-} = \sum_{i=1}^{I} c_j^{\min} * z_{j,i,s}, \quad j = 1, \ldots, J; \quad s = r_j, \ldots, H \tag{4.13}$$

$$\sum_{j=1}^{J} z_{j,i,s} - o_{i,s}^{+} + o_{i,s}^{-} = b, \quad i = 1, \ldots, I; \quad s = 1, \ldots, H \tag{4.14}$$

Fonction objectif

Dans notre modèle, une solution doit respecter les contraintes souples autant que possible avant de minimiser la somme des retards pondérés. Ainsi, les objectifs liés aux déviations par rapport aux contraintes souples sont pondérés dans la fonction objectif finale (voir l'équation 7.18) par α et β, de sorte que chaque poids soit supérieur au coût des retards maximaux. Notez que le coût des retards maximaux est calculé en supposant que toutes les tâches se terminent au cours de la dernière semaine de l'horizon de planification. En outre, parmi les deux contraintes souples, la contrainte sur la charge minimale est légèrement plus importante que la contrainte sur le nombre de tâches effectuées par un employé dans la semaine. Par conséquent, α est légèrement supérieur à β. Les trois objectifs différents sont énumérés comme suit (de la plus haute à la plus basse priorité).

Objectif 1 : l'objectif associé à la contrainte (7.15) est d'éviter autant que possible que, pour une semaine donnée, l'employé affecté à une tâche exécute de cette tâche moins que sa charge minimale prédéfinie. Pour cette raison, seule la déviation négative par rapport à cet objectif est minimisée dans l'équation suivante.

$$\text{Min} \sum_{j=1}^{J} \sum_{s=r_j}^{H} u_{j,s}^{-} \tag{4.15}$$

Objectif 2 : l'objectif associé à la contrainte (7.16) est de limiter la perte de temps des employés due au passage d'une tâche à l'autre. À cette fin, aucun employé ne doit travailler sur plus que le nombre maximum de tâches par semaine. Par conséquent, seule la déviation positive par rapport à ce but est minimisée dans la fonction objectif suivante.

$$\text{Min} \sum_{i=1}^{I} \sum_{s=1}^{H} o_{i,s}^{+} \tag{4.16}$$

Objectif 3 : la fonction objectif initiale dans l'équation ci-dessous cherche à minimiser la somme des retards pondérés des tâches.

$$\text{Min} \sum_{j=1}^{J} w_j T_j \tag{4.17}$$

Après avoir incorporé les objectifs des contraintes souples, la fonction objectif finale peut être écrite comme suit.

$$\text{Min} \sum_{j=1}^{J} \sum_{s=r_j}^{H} \alpha u_{j,s}^{-} + \sum_{i=1}^{I} \sum_{s=1}^{H} \beta o_{i,s}^{+} + \sum_{j=1}^{J} w_j T_j \tag{4.18}$$

Les résultats expérimentaux du modèle GPNE montrent que, bien qu'il soit utile pour résoudre des instances de tailles petites, il n'est pas capable de résoudre des instances de taille réelle. Par exemple, lorsque nous essayons de résoudre une instance de 100 tâches, le solveur n'arrive pas à retourner une solution optimale dans un temps de calcul raisonnable. Par conséquent, nous optons pour des méthodes approchées (heuristiques et métaheuristiques) afin de calculer des solutions de bonne qualité pour des instances de grande taille dans un temps raisonnable.

4.3 Algorithmes gloutons

Dans cette section, nous proposons des heuristiques gloutonnes à trois phases, que nous appelons TPGH, pour générer des solutions initiales.

La méthode TPGH est appliquée pour calculer une solution initiale. La première phase de cette méthode, appelée “Phase constructive”, cherche à affecter les employés aux tâches de manière appropriée. Sur la base de ces affectations, la deuxième phase, appelée “Phase d'évaluation”, cherche à planifier les tâches de chaque employé sur l'horizon temporel fixé. L'objectif est d'identifier pour chaque employé, la charge à réaliser de chaque tâche au cours de chaque semaine. La dernière phase, appelée “Phase d'ajustement”, vise à équilibrer la charge de travail des projets entre les employés.

Nous détaillons chacune des phases dans les trois sections suivantes.

4.3.1 Phase constructive

L'objectif de cette phase est d'affecter les employés aux tâches. Cela signifie que pour chaque tâche, nous devons choisir l'employé approprié parmi les employés qui la maîtrise. Nous procédons de la manière suivante : d'abord, nous utilisons une des règles de priorité détaillées ci-dessous pour déterminer un ordre de sélection des tâches. Ensuite, nous utilisons une méthode heuristique pour sélectionner l'employé approprié pour effectuer chaque tâche dans l'ordre précédemment déterminé. Le processus se répète jusqu'à ce que toutes les tâches soient affectées.

Plusieurs règles de priorité ont été testées pour la sélection des tâches. Nous listons dans ce qui suit les plus adaptées à notre modèle d'ordonnancement.

1. Weighted Longest Processing Time (WLPT) : ordonne les tâches par ordre décroissant de leur charge nominale pondérée par l'efficacité moyenne ;
2. Weighted Shortest Processing Time (WSPT) : ordonne les tâches dans l'ordre non décroissant de leurs charges nominales pondérées par l'efficacité moyenne ;
3. Weighted Earliest Due Dates (WEDD) : ordonne les tâches dans l'ordre non décroissant de leurs dates de fin souhaitées pondérées par l'efficacité moyenne.

Une fois que nous avons défini l'ordre selon lequel les tâches seront sélectionnées, l'étape suivante consiste à choisir, pour chaque tâche, l'employé qui en sera chargé parmi les employés qui la maîtrisent. Ce problème peut être considéré comme un cas général du problème de bin-packing Bernhard and Jens (2006), où un nombre donné d'éléments (tâches) de différentes tailles doivent être assignés à des bins (employés) de même capacité, de sorte que le nombre total de bins utilisés soit minimisé.

Pour affecter les employés aux tâches, nous avons adapté deux algorithmes différents utilisés pour traiter le problème de bin-packing (ou de remplissage de sacs). Ces algorithmes sont l'algorithme First-fit et l'algorithme Best-fit. Les détails de ces algorithmes sont donnés dans Alg. 2 et Alg. 3.

Dans chacune des adaptations de ces algorithmes, l'entrée est la liste des tâches (notée N) ordonnée avec l'une des règles de priorité précédentes. Pour chaque tâche n_j prise dans l'ordre de la liste N, on obtient l'ensemble des employés E_j maîtrisant cette tâche et affectés au projet k_l (la tâche n_j fait partie du projet k_l). Pour chaque employé $e_i \in E_j$ on obtient $\tau_{i,l}^{tot}$, qui correspond à la somme de temps de travail maximum de cet employé alloué au projet k_l. On obtient également $\tau_{i,l}^{act}$ qui correspond à la charge de travail du projet k_l déjà attribuée à l'employé e_i. Dans l'algorithme FF, le premier employé trouvé avec un temps de travail suffisant pour réaliser la tâche (c'est-à-dire $\tau_{i,l}^{act} + p_{j,i} \leq \tau_{i,l}^{tot}$) sera affecté à celle-ci. Alors que dans l'algorithme BF, on calcule cette somme pour chaque employé de la liste E_j, puis on choisit celui dont la disponibilité est la plus petite et suffisante pour compléter la tâche. Dans les deux algorithmes, nous mettons à jour la disponibilité de l'employé sélectionné. Cela signifie que le choix de l'employé pour la prochaine tâche est influencé par la charge de travail qui lui est déjà assignée.

Algorithm 2 : Pseudo-code de la procédure First-fit (FF)

Entrée : N : Ensemble des tâches ordonnées selon une règle de priorité
Sortie : $\sigma : \phi$ Affectation des employés aux tâches
$\tau = \phi$: Temps de travail maximum des employés sur chaque projet pendant chaque semaine
$\tau^{act} = \phi$: Temps de travail réel des employés sur les projets
$\tau^{tot} = \phi$: Temps de travail total des employés sur chaque projet pendant tout l'horizon
pour chaque Employé $e_i \in E$ **faire**
  pour chaque Projet $k_l \in K$ **faire**
    pour chaque Semaine $h_s \in H$ **faire**
      $\tau_{i,s,l} = D_{i,s} * Q_{i,l}$
      $\tau^{tot}_{i,l} += D_{i,s} * Q_{i,l}$
    fin pour
  fin pour
fin pour
pour chaque Tâche $n_j \in N$ **faire**
  k_l : Projet auquel la tâche n_j appartient
  E_j : Liste des employés maîtrisant la tâche n_j
  pour chaque employé $e_i \in E_j$ **faire**
    $temp = 0$
    $fin = faux$
    pour $(s = r_j; s \leq d_j \& fin = faux; s++)$ **faire**
      $temp += \tau_{i,s,l}$
      si $(p_{j,i} \leq temp)$ et $(\tau^{act}_{i,l} + p_{j,i} \leq \tau^{tot}_{i,l})$ **alors**
        Ajouter n_j à $\sigma[i]$
        $\tau^{act}_{i,l} += p_{j,i}$ // Mettre à jour la disponibilité de l'employé
        $fin = vrai$
      fin si
    fin pour
  fin pour
fin pour
retoune σ

Algorithm 3 : Pseudo-code de la méthode Best-fit (BF)

Entrée : N : Ensemble de toutes les tâches ordonnées avec une règle de priorité
Sortie : $\sigma : \phi$ Affectation des employés aux tâches
$\tau = \phi$: Temps de travail maximum des employés sur chaque projet durant chaque semaine
$\tau^{act} = \phi$: Temps de travail réel des employés sur les projets
pour chaque Employé $e_i \in E$ **faire**
  pour chaque Projet $k_l \in K$ **faire**
    pour chaque Semaine $h_s \in H$ **faire**
      $\tau_{i,s,l} = D_{i,s} * Q_{i,l}$
    fin pour
  fin pour
fin pour
pour chaque Tâche $n_j \in N$ **faire**
  k_l : Projet auquel la tâche n_j appartient
  E_j : Liste des employés maîtrisant la tâche n_j
  $TrinomeList = \phi$
  pour chaque Employé $e_i \in E_j$ **faire**
    $temp = 0$
    pour $(s = r_j; s \leq d_j; s++)$ **faire**
      $temp += \tau_{i,s,l}$
    fin pour
    Ajouter $(e_i, temp, \tau^{act}_{i,l})$ à $TrinomeList$
  fin pour
  Trier $TrinomeList$ dans l'ordre croissant de $temp$
  Trier $TrinomeList$ dans l'ordre croissant de τ^{act}.
  Affecter n_j à l'employé $TrinomeList[0][0]$ // l'employé avec la plus petite disponibilité
  Ajouter n_j à $\sigma[TrinomeList[0][0]]$
  $\tau^{act}_{i,l} += p_{j,i}$ // mettre à jour la disponibilité de l'employé
fin pour
retourne σ

4.3.2 Phase d'évaluation

Cette phase est le cœur de notre heuristique. L'objectif est de déterminer la charge que chaque employé doit effectuer par semaine de chacune de ses tâches, de sorte que la somme des retards pondérés soit minimisée. Comme mentionné dans la section 4.1.5, ce problème est $\mathcal{NP}$-difficile puisqu'il s'agit d'un cas général du problème d'ordonnancement à une seule machine $1 \mid r_j, pmtn \mid \sum w_j T_j$. Nous utilisons un modèle de flot maximum à coût minimum (ou maximum flow with minimum cost (MFMC)) (Ahuja et al. (1993)) pour construire une solution approchée à ce problème.

Le problème de MFMC consiste à faire passer le maximum de flot possible dans un réseau, de sorte que le coût requis soit minimisé. Dans notre approche, le flot représente la charge des tâches, tandis que le coût requis est la somme des pénalités des retards.

Nous utilisons l'exemple précédent (voir Section 4.1.6) pour illustrer la structure du graphe utilisé pour obtenir un ordonnancement des tâches de chaque employé. Comme l'affectation de chaque employé est indépendante des autres employés, nous construisons ce graphe pour chaque employé indépendamment de l'affectation des autres employés. La structure du graphe proposé est la suivante :

– L'ensemble de nœuds se compose de trois sous-ensembles de nœuds : {Tâche}, {Projet, Semaine}, et {Semaine}. Deux nœuds fictifs (S et P) sont ajoutés pour indiquer le début et la fin des projets, respectivement.

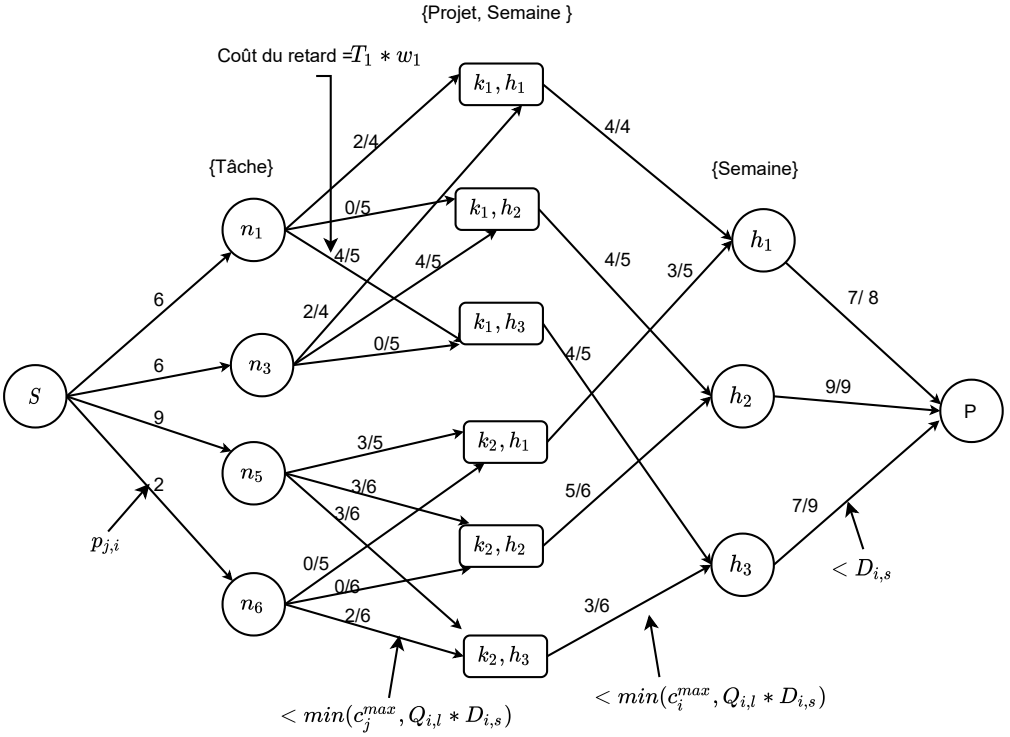


Fig. 4.1 : Graphe de flot pour l'affectation de l'employé e_1.

– La capacité d'un arc entre les nœuds {Source (S)} et {Tâche} représente le temps de traitement de la tâche ($p_{j,i}$). La capacité d'un arc entre les nœuds ({Tâche} et {Projet, Semaine})

ou entre les nœuds ({Projet, Semaine} et {Semaine}) représente le temps maximum que l'employé peut consacrer à la tâche correspondante par semaine. Il s'agit donc du minimum entre la charge maximale de la tâche ($c_j^{\max}$) par semaine et le temps de travail maximum de l'employé sur le projet correspondant ($D_{i,s} \times Q_{i,l}$). Enfin, entre les nœuds {Semaine} et le nœud {Puit (P)}, la capacité de l'arc représente la disponibilité de l'employé pendant la semaine h_s ($D_{i,s}$).

– Les coûts des arcs sont nuls, sauf pour les arcs entre ({Tâche},{Projet, Semaine}) qui représentent le nombre de semaines des retards pondérés par la pénalité de la tâche. Par exemple, le coût de l'arc $(n_1, \{k_1, h_3\})$ est calculé comme $w_1 \times T_1$, où $w_1 = 10$, et $T_1 = 1$.

La Fig. 4.1 montre le graphe utilisé pour obtenir un ordonnancement pour les tâches de l'employé e_1. Notons que toutes les contraintes dures sont modélisées dans ce graphe. Cependant, la solution obtenue peut violer les contraintes souples, car elles ne sont pas modélisées par ce graphe. Comme dans la fonction objectif du modèle mathématique, les violations des contraintes souples sont minimisées dans la fonction objectif avec les mêmes poids (voir Section 4.2).

Pour résoudre le problème du flot maximum à coût minimum pour le graphe proposé, nous utilisons l'algorithme polynomial d'Edmonds-Karp (Edmonds and Karp (1972)). Une solution est faisable si le flot maximum est égal à la somme totale des charges de tâches.

4.3.3 Phase d'ajustement

L'idée de cette phase est d'équilibrer la charge de travail des projets entre les employés pour augmenter la chance de trouver une bonne solution. Cette phase est donc appliquée lorsqu'une solution avec des tâches incomplètes ou en retard est retournée par la phase d'évaluation. Une tâche est dite "incomplète" lorsque le flot sur l'arc entre le nœud {Source} et le nœud {Tâche} est strictement inférieur à la capacité de l'arc (c'est-à-dire inférieur au temps de traitement de la tâche). En modifiant l'affectation des employés surchargés, on cherche à utiliser la disponibilité supplémentaire sur certains projets où elle est le plus nécessaire. L'ensemble du processus de cette phase est décrit plus formellement dans Algo. 4.

Considérons à nouveau la solution de l'exemple précédent. L'employé e_2 est disponible 9 demi-journées pendant la semaine h_1, pendant cette période il a effectué 4 demi-journées du projet k_1 et rien du projet k_2 (c'est-à-dire que l'employé e_2 a encore 5 demi-journées sur le projet k_2). D'autre part, l'employé e_1 a une tâche en retard (n_5), dont le temps de traitement est inférieur à 5, peut être commencée à la semaine h_1 et appartient au projet k_2. En changeant l'affectation de cette tâche, on peut obtenir une solution améliorante, avec un coût égal à 40. Cet exemple est bien illustré aux Figures 4.8 et 4.9.

	h_1	h_2	h_3
n_1	2	0	4 ×
n_3	2	4	0
n_6	0	2	0

✓ indique une nouvelle affectation
× indique une tâche en retard

Tab. 4.8 : Ordonnancement amélioré pour l'employé e_1.

	h_1	h_2	h_3
n_2	0	0	3 ×
n_4	4	5	0
n_5	4	2 ✓	
n_7	0	3	3 ×
n_8	0	0	3

Tab. 4.9 : Ordonnancement amélioré pour l'employé e_2.

4.4 Méthodes de résolution approchée itératives

Pour améliorer la meilleure solution initiale obtenue, nous introduisons deux méthodes itératives : un algorithme de recherche locale et une recherche tabou. Les deux méthodes itératives

Algorithm 4 : Pseudo-code de la phase d'ajustement

```
1 : Entrée : ξ : Liste des tâches incomplètes ou en retard ; D^act : Temps de travail
    effectif des employés sur les projets pendant l'horizon de planification
2 : Sortie : σ la nouvelle affectation des employés aux tâches
3 : LisTrinome = ϕ
4 : pour chaque Employé e_i ∈ E faire
5 :     pour chaque Semaine h_s ∈ H faire
6 :         si D^act_{i,s} < D_{i,s} alors
7 :             pour chaque Projet k_l ∈ K faire
8 :                 si (D^act_{i,s} * Q_{i,l} < D_{i,s} * Q_{i,l}) alors
9 :                     Ajouter le trinôme (e_i, h_s, k_l) à LisTrinome
10 :                fin si
11 :            fin pour
12 :        fin si
13 :    fin pour
14 : fin pour
15 : pour chaque trinomial ∈ LisTrinome faire
16 :    tant que D^act_{i,s} < D_{i,s} faire
17 :        n_j = ξ[cpt]
18 :        si (r_j ≤ h_s ≤ d_j) et (n_j ∈ N_l) et (p_{j,i} < D_{i,s} − D^act_{i,s}) alors
19 :            Affecter e_i à n_j
20 :            Mettre à jour σ
21 :            D^act_{i,s} = D^act_{i,s} + p_{j,i}
22 :        fin si
23 :    fin tant que
24 : fin pour
25 : retourne σ
```

utilisent des fonctions de voisinage pour générer l'espace de recherche à partir d'une solution donnée. Dans ce qui suit, nous décrivons les métaheuristiques proposées.

4.4.1 Codage d'une solution et fonctions de voisinage

Cette section décrit le codage des solutions et fournit une description détaillée des différentes fonctions de voisinages utilisées pour construire des espaces de recherche de voisinages.

4.4.1.1 Codage de la solution

Une solution complète du problème est définie par l'affectation des employés aux tâches (notée σ), la charge réalisée des tâches au cours des semaine (notée π), et le retard correspondant (noté ω). En considérant la solution de l'exemple précédent, le codage de la solution est le suivant :

$\sigma = \{\{n_1, n_3, n_5, n_6\}, \{n_2, n_4, n_7, n_8\}\}$. Cela signifie que l'employé e_1 est affecté aux tâches $\{n_1, n_3, n_5, n_6\}$ et e_2 est affecté aux tâches $\{n_2, n_4, n_7, n_8\}$.

$\pi = \{\{\{2, 2, 3, 0\}, \{0, 4, 3, 2\}, \{4, 0, 3, 0\}\}, \{\{0, 4, 0, 0\}\{0, 5, 3, 0\}, \{3, 0, 3, 3\}\}\}$. Cela signifie que l'employé e_1 a effectué, pendant la semaine h_1, 2 demi-journées de n_1, 2 demi-journées de n_3, et 3 demi-journées de n_5. Pendant la semaine h_2, le même employé a effectué 4 demi-journées de n_3, 3 demi-journées de n_5 et 2 demi-journées de n_6, et ainsi de suit.

$\omega = \{\{1, 0, 1, 0\}, \{1, 0, 1, 0\}\}$. Une valeur de 1, par exemple, signifie que la tâche est en retard et 0 signifie qu'elle est à l'heure. Pour une tâche incomplète, nous fixons la valeur à -1.

4.4.1.2 Fonctions de recherche de voisinages

À chaque itération des méthodes itératives, un nombre limité d'affectations appelé "espace de voisinage" est généré à partir d'une solution donnée (solution courante). Cet espace de voisinage est généré en appliquant trois fonctions de voisinage. Ces différentes fonctions reposent sur la modification de l'affectation des employés aux tâches. Ensuite, pour chaque affectation de tâche

modifiée, une liste d'employés maîtrisant cette tâche est établie. Cette liste est ordonnée en appelant l'algorithme de best-fit.

Pour illustrer le fonctionnement de ces fonctions de voisinage, considérons à nouveau la solution de l'exemple précédent. Selon la solution partielle π_0, les tâches n_1, n_2, n_5 et n_7 ont chacune une semaine de retard. La première fonction de voisinage, notée V1, consiste à réaffecter les tâches en retard ou incomplètes à un autre employé.

Figure 7.8c présente les affectations voisines générées à l'aide de la fonction V1. Les nouvelles affectations σ_1 et σ_3 sont obtenues, respectivement, en modifiant l'affectation des tâches n_1 et n_6 de l'employé e_1 à l'employé e_2. Les nouvelles affectations σ_2 et σ_4 sont obtenues en changeant l'affectation des tâches n_2 et n_7 de l'employé e_2 à l'employé e_1. Notons que les employés e_1 et e_2 ont tous deux les compétences nécessaires pour effectuer les nouvelles tâches ajoutées à leurs affectations.

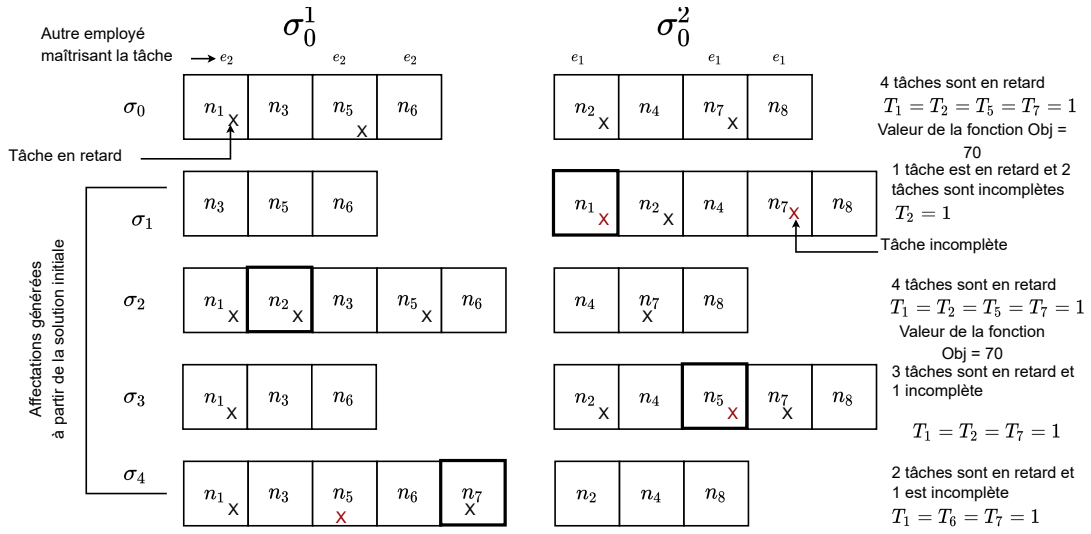


Fig. 4.2 : Affectations générées avec la fonction de voisinages V1 à partir de la solution π_0.

La deuxième fonction de voisinage, notée V2, tente de libérer du temps pour un employé effectuant une tâche en retard ou incomplète. Il s'agit de réaffecter une tâche terminée à l'heure à un autre employé qui la maîtrise. Pour bien comprendre cette fonction de voisinage, considérons la solution π_0. Notamment, π_0^1 (la solution de l'employé e_1), on peut voir que les tâches n_3 et n_6 sont à l'heure. Alors que la tâche n_6 est la seule réalisée par e_1 et maîtrisée par e_2. En modifiant l'affectation de la tâche n_6, une seule nouvelle affectation peut être générée (σ_1). Notez qu'il n'est pas possible de générer de nouvelles affectations en modifiant l'affectation des tâches terminées à l'heure réalisées par l'employé e_2 (n_4 et n_8). En effet, l'employé e_1 ne possède pas la compétence requise par n_4 et la modification de l'affectation de n_8 ne libère pas du temps pour l'employé e_2.

La troisième fonction de voisinage, notée V3, consiste à échanger l'affectation d'une tâche en retard ou incomplète avec une autre tâche. La permutation n'est possible que si les deux tâches appartiennent au même projet et leur intervalle de temps d'exécution se chevauchent. Figure 4.4 montre un exemple de génération des voisinages avec cette fonction. Dans cet exemple, nous échangeons l'affectation des tâches en retard n_1 et n_5 effectuées par l'employé e_1 avec n_2 et n_7 effectuées par l'employé e_2, respectivement. Selon les intervalles de temps d'exécution définis dans la table 4.2 (ligne (r_i, d_i)), n_1 affectée à l'employé e_1 chevauche avec n_2 et n_4 effectuées par l'employé e_2. Cependant, e_1 n'est pas compétent pour effectuer n_4. n_5 ne chevauche qu'avec n_7, qui peut

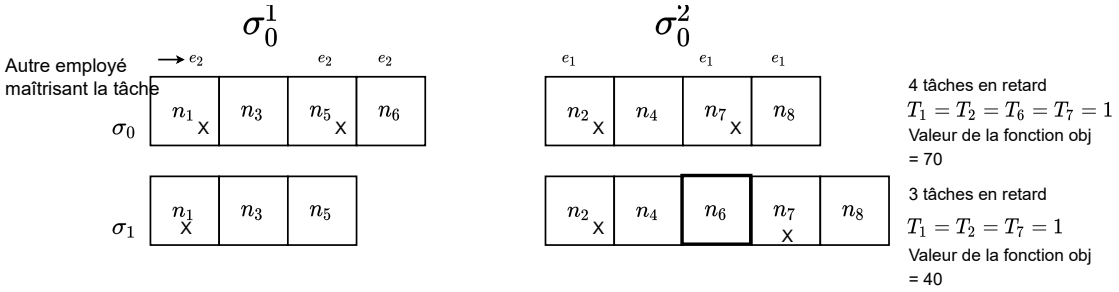


Fig. 4.3 : Affectations générées avec la fonction V2 à partir de la solution π_0.

être effectuée par l'employé e_1. Afin de déterminer dans quel ordre nous appelons ces fonctions de

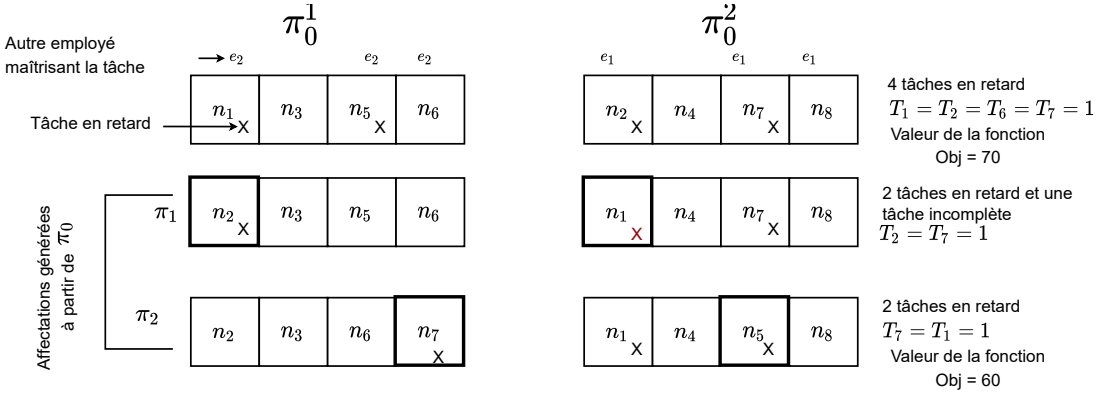


Fig. 4.4 : Affectations générées avec la fonction V3 à partir de la solution π_0.

voisinage dans les méthodes itératives, nous avons fait plusieurs expériences et les résultats finaux sont donnés dans le Tableau 4.17.

4.4.2 Algorithme de recherche locale

Nous avons développé un algorithme de recherche locale (ou local search (LS)) pour améliorer itérativement une solution en examinant ses voisins. Plusieurs stratégies de recherche locale ont déjà été proposées pour résoudre des problèmes d'optimisation difficiles, et la forte performance de ce type d'algorithmes a été soulignée dans la littérature (voir par exemple, Ishibuchi and Murata (1998) ; Tseng and Lin (2009) ; Choi and Choi (2002)).

Afin d'expliquer facilement notre implémentation de l'algorithme LS, considérons V^{fst}, V^{snd} et V^{trd} nos trois fonctions de voisinage et supposant que V^{fst} domine V^{snd} qui domine à son tour V^{trd} en terme de meilleures solutions trouvées.

Comme mentionné précédemment, notre procédure LS commence avec la meilleure solution initiale obtenue en appliquant l'algorithme TPGH. A l'itération zéro, la solution initiale est la solution courante (notée S^{cur}). A chaque itération, l'algorithme LS commence par exploiter $V^{fst}(S^{cur})$. Ensuite, la phase d'évaluation est appliquée pour examiner les affectations fraîchement générées. La solution du meilleur voisin est identifiée par la fonction objectif produisant une valeur meilleure que celle des autres voisins. Si une amélioration est observée, la solution voisine devient la nouvelle

solution courante et le point de départ de l'itération suivante. Sinon, après que l'évaluation de $V^{fst}(S^{cur})$ soit terminée, l'évaluation du voisinage $V^{snd}(S^{cur})$ commence. Si une amélioration est observée, l'exploitation du voisinage $V^{fst}(S^{cur})$ est reprise. Autrement, on exploite le voisinage $V^{trd}(S^{cur})$, et ainsi de suite. Le LS utilise la phase d'ajustement de la méthode TPGH sur une solution courante pour trouver une nouvelle solution améliorante. Ce processus de génération et d'évaluation des solutions voisines est répété jusqu'à ce qu'aucune fonction de voisinage ne puisse améliorer la solution courante.

La recherche locale a deux paramètres à définir. Le premier paramètre désigné par λ^{adjus}, contrôle la fréquence à laquelle la procédure de la phase d'ajustement est invoquée. Une valeur de 20 par exemple, signifie que la phase d'ajustement est invoquée toutes les 20 solutions. Le deuxième paramètre désigné par λ^{neigh}, correspond à la taille de la fonction de voisinage (c'est-à-dire le nombre maximum d'affectations générées par une fonction de voisinage).

4.4.3 Algorithme de recherche tabou

La procédure générale d'un algorithme de recherche tabou a été présentée dans la Section 2.2.2. Nous commençons notre procédure TS à partir de la meilleure solution initiale donnée par la méthode TPGH. A chaque itération, le TS examine l'ensemble des voisins de la solution courante (S^{cur}) et choisit la meilleure solution non classée tabou. La meilleure solution obtenue est retenue, même si elle n'améliore pas la fonction objectif courante. Elle sera considérée comme la solution courante de l'itération suivante. La recherche est guidée par la valeur de la fonction objectif. La procédure est répétée tant que le critère d'arrêt n'est pas satisfait.

Les voisins de la solution courante sont générés par les fonctions de voisinage présentées précédemment dans la Section 4.4.1.2. Les différentes fonctions sont également utilisées dans le même ordre que dans l'algorithme LS. Toutes les affectations générées sont examinées séparément en appelant l'algorithme MFMC. Comme l'algorithme LS, la procédure TS s'arrête lorsqu'aucune fonction de voisinage ne peut améliorer la solution actuelle.

Pour éviter de revenir à une affectation qui a déjà été évaluée, nous stockons les affectations récemment visitées dans une liste tabou. Nous attribuons à chaque nouvelle affectation ajoutée dans la liste tabou deux nombres. Le premier nombre représente une valeur de hachage spécifique à chaque affectation. Nous utilisons une fonction de hachage pour calculer cette valeur sur la base de l'encodage de la solution correspondante. Le deuxième nombre correspond au nombre d'itérations pour lesquelles l'affectation est maintenue dans la liste tabou, noté λ^{erase} et est donc un paramètre de la méthode. La taille de la liste tabou est fixée par un paramètre de la méthode noté λ^{tlsize}.

En utilisant une technique de diversification, la procédure TS passe à une nouvelle partie de l'espace de recherche, lorsque la recherche se bloque autour d'un optimum local. Nous appliquons la technique de diversification après un nombre donné de solutions successives (noté λ^{divers}) sans aucune amélioration de la solution actuelle. Dans ce cas, la recherche recommence à partir d'une des solutions calculées par la méthode TPGH.

Contrairement à la technique de diversification, l'intensification est appliquée lorsque l'espace de voisinage de la solution courante est prometteur. Ainsi, la recherche doit être intensifiée autour de cet espace. Ce processus est appliqué si plus de λ^{intens} solutions améliorent la solution courante lors d'une itération donnée, où λ^{intens} est un paramètre de la méthode.

4.5 Résultats expérimentaux

Nous présentons dans cette section les caractéristiques des instances et les résultats expérimentaux des différentes approches de résolution proposées. Nos expérimentations se composent de deux parties. Tout d'abord, nous voulons analyser la sensibilité des paramètres les plus impor-

tants du problème pour en tirer des meilleurs stratégies de gestion. La deuxième partie concerne la performance des algorithmes proposés quand il s'agit de résoudre des instances de taille réelle.

Les expérimentations ont été menées sur un Intel® core™ i7-1.9 GHz avec 16 Go de RAM sous Windows 10. Tous les algorithmes ont été écrits en Python. Le solveur CPLEX 12.8.0 associé à l'API Python a été utilisé pour résoudre le modèle GPNE, en utilisant les paramètres du solveur par défaut à l'exception du paramètre temps.

4.5.1 Caractéristiques des instances

La réalisation des deux parties des expérimentations implique l'utilisation d'instances industrielles. Cependant, nous n'avons pas pu obtenir suffisamment de données réelles pour des raisons de confidentialité. Sur la base de la description du problème et des quelques instances réelles fournies par l'entreprise, nous avons généré 7 ensembles d'instances ($T1$–$T7$) avec 40 instances par ensemble. L'objectif principal est de tester le comportement des algorithmes proposés et l'effet de certains paramètres importants sur la résolution du problème.

Chaque instance d'un ensemble contient 2 projets qui doivent être planifiés sur une période de 4 semaines, 50 tâches dont la charge nominale varie entre 3 et 5 jours, et 8 employés multi-compétences. Le nombre de tâches par projet suit une distribution uniforme entre 20 et 30 tâches. Le nombre de compétences nécessaires à l'exécution de toutes les tâches est compris entre 3 et 5 ; et chaque employé peut disposer de 2 à 3 compétences. Les dates de début au plut tôt de 30% des tâches (choisies au hasard) sont comprises entre 2 et 3. Et pour les 70% des tâches restantes, leur date de début au plus tôt ont été fixées à 0. Les dates de fin souhaitées de 50% des tâches (également choisies au hasard) sont comprises entre 2 et 3, et ainsi 4 pour les autres tâches. Pour chaque tâche parmi les 30% des tâches choisies aléatoirement, la valeur de sa charge maximale (respectivement minimale) est générée aléatoirement entre 2 et sa charge nominale (respectivement maximale). La disponibilité de chaque employé pendant chaque semaine est comprise entre 2 et 10. Les quotités maximales des employés sur les projets suivent une distribution uniforme entre 10% et 100%.

Les résultats préliminaires ont montré que la difficulté de résoudre une instance est principalement liée à un équilibre approprié entre l'horizon de planification, la charge de travail des projets, le nombre d'employés et le nombre de compétences par employé. Si la valeur d'un de ces paramètres domine les valeurs des autres paramètres, une instance peut être facile à résoudre (c'est-à-dire que l'optimalité ou l'infaisabilité peut être facilement prouvée). Alors qu'une instance peut être difficile lorsque les valeurs de ces paramètres sont équilibrées. De plus, la difficulté de résolution d'une instance peut augmenter de manière significative (même si le problème est déjà $\mathcal{NP}$-difficile) lorsque nous considérons certains autres paramètres importants, tels que les charges minimales et maximales des tâches, la disponibilité des employés et leurs quotités sur les projets, le nombre maximal de tâches par employé et par semaine, et le nombre maximal de compétences par employé. Afin d'analyser l'effet de ces paramètres sur la résolution des problèmes, nous changeons leurs valeurs par défaut en fonction de l'ensemble des instances.

Le tableau 4.10 présente la distribution spécifique des valeurs de ces paramètres sur chaque ensemble d'instances. De gauche à droite, les colonnes indiquent le type d'instance, le pourcentage de tâches avec des charges minimales, le pourcentage de tâches avec des charges maximales, le pourcentage d'employés ayant une disponibilité inférieure ou égale à 5 pendant au moins une semaine, le pourcentage d'employés ayant des quotités inférieures ou égales à 50% sur au moins un projet, le nombre de tâches que chaque employé ne doit pas dépasser par semaine, et la dernière colonne fait référence au nombre maximum de compétences par employé.

Comme vous pouvez le voir dans ce tableau, nous changeons la valeur d'un paramètre tout en gardant les autres paramètres fixes. Par exemple, pour évaluer l'effet des charges minimales des tâches sur la qualité des solutions, nous définissons pour 70% de tâches (au lieu de 30%) des charges minimales entre 2 et les charges maximales de ces tâches.

Ensemble d'instances	$c_j^{\min}$	$c_i^{\max}$	$D_{i,s}(\leq 5)$	$Q_{i,l}(\leq 5)$	b	Le nombre maximum de compétences
T1	70%	30%	30%	20%	4	3
T2	30%	70%	30%	20%	4	3
T3	30%	30%	70%	20%	4	3
T4	30%	30%	30%	20%	2	3
T5	30%	30%	30%	40%	2	3
T6	30%	30%	30%	20%	4	1
T7	30%	30%	30%	20%	4	3

Tab. 4.10 : Valeurs de certains paramètres par ensemble d'instances

4.5.2 Résultats du modèle mathématique

Le modèle mathématique (GPNE) a été testé sur les différents ensembles d'instances. Les tableaux 4.11–4.13 comparent les résultats obtenus. Dans ces tableaux, de gauche à droite, les colonnes font référence au type d'instances, au nombre d'instances réalisables, au nombre de solutions qui sont prouvées comme étant les solutions optimales, au temps moyen nécessaire pour prouver l'optimalité, et aux déviations moyennes par rapport à l'optimalité pour les instances qui ne sont pas résolues de manière optimale.

Comme mentionné dans la section précédente, les expérimentation pour le GPNE sont réalisées pour analyser l'influence de certains paramètres importants du problème sur la qualité des solutions retournées. De plus, nous avons voulu voir comment le temps de résolution influence la déviation par rapport à l'optimalité. Pour atteindre ce dernier objectif, nous avons récupéré les résultats du modèle GPNE après 10 minutes puis après 30 minutes de temps de calcul par instance. Dans les deux cas, nous comparons la valeur de la meilleure solution obtenue avec la solution optimale. Notons que, pour obtenir les solutions optimales, nous avons exécuté le modèle GPNE sans limite de temps jusqu'à ce qu'une solution réalisable soit retournée.

Ensemble d'instances	Nombre d'instances réalisables	Nombre d'instances résolues jusqu'à l'optimalité	Temps moyen pour atteindre l'optimalité (s)	Déviation moyen par rapport aux objectifs des contraintes		
				Objectif 1	Objectif 2	Objectif 3
T1	23	5	493	8.4%	3.2%	5.2%
T2	25	4	478.8	7.2%	3.6%	6.2%
T3	9	3	385.3	4.4%	6.3%	8.2%
T4	16	10	143.2	4.6	4.7%	7.6%
T5	2	0	-	12.6	7.4%	9.2%
T6	10	8	148.8	4.7%	5.7%	6.2%
T7	36	20	93.2	3.2%	0%	5.2%

Tab. 4.11 : Résultats GPNE après un temps de calcul limité à 10 min

Instance set	Nombre d'instances réalisables	Nombre d'instances résolues jusqu'à l'optimalité	Temps moyen pour atteindre l'optimalité (s)	Déviation moyen par rapport aux objectifs des contraintes		
				Objectif 1	Objectif 2	Objectif 3
T1	27	19	617.2	5.4%	2%	4.1%
T2	29	17	638.6	4.2%	0%	3.7%
T3	12	7	59.2	1.1%	3,1%	5.9%
T4	17	15	385.6	5,3	2,8%	6.1%
T5	3	0	-	7.2	5.8%	10,2%
T6	18	12	203.5	2.8%	5.3%	5.1%
T7	38	22	133.5	1.5%	0%	4.5%

Tab. 4.12 : Résultats GPNE après un temps de calcul limité à 30 min

Ensemble d'instance	Nombre d'instances réalisables	Nombre d'instances résolues jusqu'à l'optimalité	Temps moyen pour atteindre l'optimalité (s)	Déviation moyen par rapport aux objectifs des contraintes		
				Objectif 1	Objectif 2	Objectif 3
T1	35	24	199.5	6.9%	2.6%	6.3%
T2	37	25	146.2	5.3%	0%	5.8%
T3	12	7	245.8	3.2%	3.7%	10.6%
T4	16	13	91.8	3.4	3.8%	6.6%
T5	2	0	-	12.6	7.4%	8.1%
T6	14	10	91.2	2.2%	4.5%	7.2%
T7	40	33	67.3	2.4%	0%	4.4%

Tab. 4.13 : Résultats GPNE après un temps de calcul limité à 10 min en utilisant des solutions de démarrage (Warm start)

Tab. 4.14 : Comparaison des résultats du GPNE

Méthodes	Nombre d'instances réalisables	Nombre d'instances résolues jusqu'à l'optimalité	Temps moyen pour atteindre l'optimalité
GPNE après 10 min	121	50	290.3
GPNE après 30 min	144	92	339.6
GPNE après 10 min avec des solutions de démarrages	156	112	140.3

Pour un temps de calcul par instance limité à 10 minutes, le tableau 4.11 résume l'effet de certains paramètres importants sur la résolution du problème. Les résultats présentés dans ce tableau montrent que la performance du modèle GPNE est généralement plus affectée par les charges minimales et maximales des tâches, la disponibilité des employés et leurs quotités sur les projets, le nombre maximal de compétences par employé. Pour les instances des ensembles T1 et T2, le modèle a une meilleur chance de trouver une solution réalisable, mais semble avoir quelques difficultés pour prouver l'optimalité des solutions trouvées (par exemple, seules 4 instances de l'ensemble T2 sont résolues de manière optimale sur 25 d'instances réalisables). Pour les instances des ensembles $T3$, $T5$ et $T6$, le modèle semble avoir encore plus de difficultés à trouver une solution réalisable. Le modèle n'obtient que respectivement, 2, 9 et 10 solutions réalisables pour les instances des ensembles T5, T3 et T6. De plus, si nous analysons les résultats en termes d'optimalité, nous pouvons clairement observer que le modèle a quelques difficultés à prouver l'optimalité pour les instances sur les ensembles $T3$ et $T5$. Le modèle n'obtient que 3 solutions optimales pour les instances sur l'ensemble $T3$ et ne parvient pas à atteindre une solution optimale pour l'instance sur l'ensemble $T5$. Les expériences montrent également que le degré de la satisfaction de l'objectif 1 semble décroître lorsque le pourcentage de tâches avec des charges minimales ou maximales devient

plus important (voir les résultats des ensembles T1 et T2). D'un autre coté, le degré de satisfaction de l'objectif 2 semble décroître lorsque les employés sont moins qualifiés ou moins disponibles (voir les résultats des ensembles $T3$ et $T6$). Enfin, si nous comparons les résultats numériques de l'ensemble $T7$ avec ceux des ensembles $T1 - -T6$, nous pouvons clairement voir que les instances des ensembles T1–T6 sont plus difficiles que les instances de l'ensemble T7. Ceci nous permet d'affirmer que chacun de ces paramètres a un effet significatif sur la complexité du problème.

Le tableau 4.12 présente les résultats du modèle après 30 min de calcul par instance. Ce tableau montre que les performances du modèle augmentent lorsque le temps de résolution est plus important. Cela est dû au fait que le nombre de solutions réalisables ainsi que le nombre de solutions optimales a augmenté dans tous les ensembles d'instances. On peut également constater que la déviation des solutions a diminué dans la plupart des ensembles d'instances.

Afin de réduire le temps de calcul du modèle, nous générons des solutions initiales que nous utilisons comme solutions de départ (warm start) pour résoudre le modèle GPNE. Nous avons utilisés la méthode TPGH pour générer les solution de départs du modèle. Le tableau 4.13 présente les résultats pour un temps de calcul limité à 10 minutes par instance. Si on compare les résultats de ce tableau avec les résultats du modèle présenté dans le Tableau (4.11) (c'est-à-dire ceux sans les solutions de départ), on constate que l'utilisation des solutions de départ a amélioré modérément les performances du modèle GPNE en termes de faisabilité et d'optimalité. Cependant, l'utilisation des solutions de départ semble diminuer significativement le temps moyen nécessaire pour atteindre l'optimalité.

Finalement, le Tableau 4.14 résume et compare les résultats numériques du modèle GPNE en utilisant les trois configurations (c'est-à-dire après 10 min de calcul par instance, après 30 min de calcul, et après 10 min de calcul en utilisant des solutions de départ). Les résultats de chaque configuration sont affichés en trois colonnes : la première colonne contient le nombre total de solutions réalisables, la deuxième colonne contient le nombre total de solutions optimales, et la dernière colonne contient le temps moyen nécessaire pour atteindre l'optimalité. Cet ensemble de résultats numériques est basé sur la somme des 6 ensembles d'instances ($T1, T2, T3, T4, T6$ et $T7$). Nous pouvons clairement observer que l'utilisation des solutions de départ améliore les performances du modèle GPNE en termes de qualité et de temps d'exécution. Le nombre total de solutions réalisables est passé de 121, 144 à 156. Le nombre total de solutions optimales est passé de 50, 92 à 112. Le temps moyen pour atteindre l'optimalité a significativement diminué de 290, 340 à 140 secondes.

4.5.3 Analyse des performances des méthodes approchées

Dans cette section, nous évaluons expérimentalement les performances des méthodes heuristiques proposées. Nous comparons les performances des algorithmes heuristiques à la formulation GPNE en utilisant des instances dérivées de données réelles.

4.5.3.1 Instances réelles

Nous générons deux ensembles d'instances (T_1^{rel}, T_2^{rel}) inspirés de quelques données réelles fournies par notre partenaire informatique. Nous observons que dans ces instances réelles, les projets à réaliser peuvent être décomposés en lots. Chaque lot (ensemble de tâches) est caractérisé par trois paramètres : un intervalle d'exécution moyen (la différence entre la date de début au plus tôt de la première tâche et la date de fin souhaité de la dernière tâche du lot), une charge de travail nominale moyenne, et des besoins moyens en compétences. En faisant varier ces trois paramètres, nous générons des instances simulées qui ont la même structure que les instances réelles. Dans le Tableau 4.15, nous présentons un exemple de la structure générale d'une instance de taille réelle.

Chacun des deux ensembles d'instances de taille réelle contient un total de 40 instances avec

les mêmes caractéristiques et un nombre différent de tâches. Chaque instance sur l'ensemble T_1^{rel} (resp. T_2^{rel}) contient 120 (resp. 130) tâches avec des charges nominales allant jusqu'à 20 (resp. 30). Chaque instance sur les deux ensembles a 4 projets, 16 employés, et un horizon de planification de 16 semaines. Le nombre total de compétences requises pour exécuter les tâches est de 8, et chaque employé maîtrise 2 à 4 compétences. Pour 80% des employés, nous fixons la disponibilité à 10 pendant chaque semaine de l'horizon de planification. La disponibilité des employés restants se situe entre 2 et 7. Les quotités maximales des employés sur les projets sont comprises entre 50% et 100%.

Tab. 4.15 : Structure générale d'une instance réelle

	Caractéristiques des lots			
Délivrance	Intervalles de temps d'exécution	Nombre de tâches	Charge nominale(homme-jour)	
Lot 1	[1,5]	27	393	
Lot 2	[4,10]	30	542	
Lot 3	[8,12]	35	458	
Lot 4	[11,16]	28	382	
	Exigences moyennes des lots pour les compétences			
	Lot 1	Lot 2	Lot 3	Lot 4
Compétence 1	26%		18%	10%
Compétence 2		28%	16%	
Compétence 3	15%		17%	
Compétence 4		24%		31%
Compétence 5	16%			27%
Compétence 6		32%	17%	17%
Compétence 7	10%		32%	15%
Compétence 8	33%	16%		

4.5.3.2 Paramétrage des méthodes itératives

Des expérimentation ont été réalisées pour ajuster les paramètres de recherche des méthodes TS et LS, l'objectif étant de maintenir un rapport plus faible entre le temps d'exécution et les qualités des solution retournées par ces méthodes. La méthode TS a les paramètres suivants : λ^{neigh} est la taille de l'espace de voisinage, λ^{tlsize} est la taille de la liste tabou, λ^{erase} est le nombre d'itérations pour lesquelles une affectation est conservée dans la liste tabou, λ^{divers} définit la fréquence à laquelle la procédure de diversification est invoquée (donnée en nombre de solutions), λ^{intens} contrôle la fréquence à laquelle la procédure d'intensification est invoquée (donnée en nombre de solutions). La recherche locale a les paramètres suivants : λ^{adjus} contrôle la fréquence à laquelle la procédure de phase d'ajustement est invoquée et λ^{neigh} correspond à la taille de l'espace de voisinage.

Différentes combinaisons de valeurs ont été testées, et seule les meilleures valeurs sont retenues ici. Afin d'obtenir la meilleure combinaison de ces paramètres, l'ensemble $T7$ est utilisé pour tester l'impact de tous les paramètres en changeant la valeur d'un paramètre tout en maintenant les autres paramètres fixes. Les résultats préliminaires ont montré que la taille de la fonction de voisinage a un impact significatif sur les performances du TS. La convergence du TS s'établit après environ 300 affectations générées par chaque fonction de voisinage. Cependant, ce paramètre n'a pas d'effet significatif sur la méthode LS. Ceci peut être expliqué par le fait que la méthode LS n'a pas besoin d'exploiter un grand nombre de voisinages pour améliorer la solution courante.

Les valeurs finales de ces paramètres et leurs plages sont présentées dans le tableau 4.16. Notez que le temps de calcul pour obtenir les meilleures valeurs des paramètres est inférieur à 2 minutes. Le nombre d'affectations générées par une fonction de voisinage est lié à la taille de l'instance. Par exemple, une instance de 50 tâches et 8 employés peut aboutir à la génération de 400 affectations à chaque itération.

Paramètre	Plage	Valeur finale
λ^{neigh}	[100, 600]	300
λ^{adjus}	[0, 50]	10
λ^{tlsize}	[0, 50]	20
λ^{erase}	[10, 50]	15
λ^{divers}	[0, 300]	50
λ^{intens}	[0, 300]	10

Tab. 4.16 : Les valeurs des paramètres utilisés par les méthodes TS et LS.

4.5.3.3 Résultats expérimentaux des heuristiques et métaheuristiques

Pour démontrer le comportement des heuristiques proposées, nous avons mené des expérimentations en deux parties. Dans la première partie, nous voulons évaluer les performances de nos heuristiques en calculant les déviations des solutions retournées par rapport aux solutions optimales trouvées par le modèle GPNE. Nous avons utilisé uniquement les instances résolues de manière optimale par le modèle sur les ensembles $T1, T2, T3, T4, T6$ et $T7$. Les résultats correspondants sont présentés dans le tableau 4.17. La formule utilisée pour calculer la déviation est la suivante :

$$GAP = \frac{\text{Somme des valeurs heuristiques des objectifs} - \text{Somme des valeurs optimales des objectifs}}{\text{Somme des valeurs heuristiques des objectifs}} \times 100\%.$$

Dans la deuxième partie, nous voulons évaluer les performances de ces heuristiques sur des instances de taille réelle (i.e., ensemble T_{real}). Dans les tableaux 4.18 et 4.19, nous présentons les résultats correspondants.

		Écart moyen entre les solutions heuristiques et les solutions optimales					
		Set T1	Set T2	Set T3	Set T4	Set T6	Set T7
Phase constructive	WSPT	11.3%	10.7%	6.6%	8.1%	7.9%	8.8%
	WLPT	10.4%	11.2%	5.7%	8.2%	7.4%	8.2%
	WEDD	9.3%	9.2%	5.6%	7.4%	6.2%	7.4%
	First fit	10.3%	9.8%	6.9%	6.8%	6.9%	8.2%
	Best fit	9.2%	9.2%	5.6%	7.4%	6.2%	7.4%
Fonctions de voisinages	V1	5.7%	4.8%	4.2%	3.6%	4.3%	4.1%
	V2	5.3%	5.2%	5.9%	4.7%	4.6%	5.2%
	V3	5.2%	6.3%	5.5%	4.6%	4.8%	4.5%
Méthodes itératives	LS	4.7%	4.3%	4.8%	4.2%	3.2%	3.9%
	TS	**2.5**%	**3.2**%	**1.5**%	**2.7**%	**2.9**%	**2.1**%

Tab. 4.17 : Comparaison des heuristiques en utilisant les instances résolues de manière optimale par le modèle GPNE.

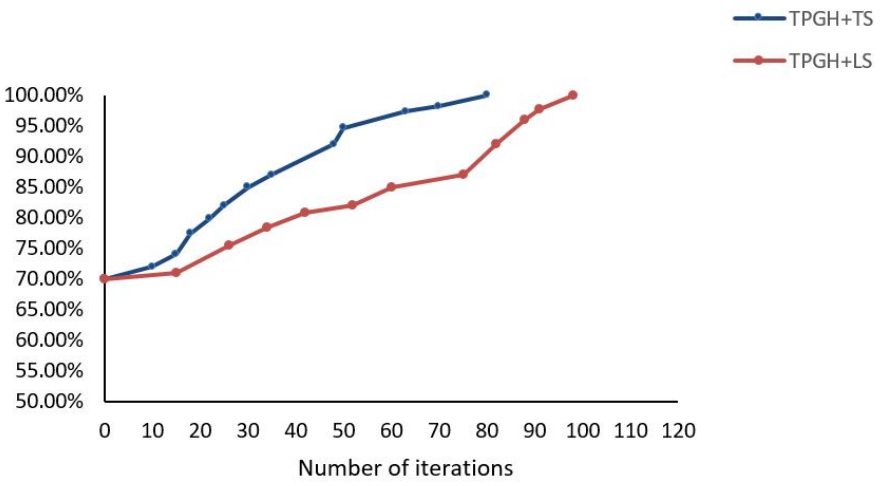


Fig. 4.5 : Nombre d'itérations nécessaires pour que le LS et le TS convergent vers une solution réalisable, avec des instances de taille réelle

Tab. 4.18 : Comparaison des méthodes sur des instances de taille réelle résolues par le GPNE

	Nombre des instances faisables	Nombre de solutions respectant les contraintes souples		Écart moyen par rapport aux objectifs des contraintes		
		Constr souple 1	Constr souple 2	Goal 1	Goal 2	Goal 3
TPGH	28	16	12	15.3%	13.3%	25.9%
LS	40	27	23	7.2%	4.6%	11.3%
TS	40	32	29	5.3%	3.2%	8.4%
GPNE (après 10min)	9	-	-	9.8%	8.2%	23.8%

Tab. 4.19 : Comparaison des méthodes sur des instances de taille réelle non résolues par le GPNE

	Nombre des instances faisables	Nombre de solutions respectant les contraintes souples		Écart de retard supplémentaire
		Contr souples 1	Contr souples 2	
TPGH	25	10	11	23.7%
LS	40	28	21	9.5%
TS	40	29	26	8.2%

Le Tableau 4.17 rapporte trois comparaisons différentes des heuristiques. La première comparaison concerne les règles de priorité et les algorithmes utilisés pour la construction des solutions initiales. Les résultats comparant les différentes règles de priorité ont été obtenus en triant la liste des tâches par l'une des trois règles de priorité, puis nous avons appliqué l'algorithme best-fit pour la sélection des employés. Ensuite, en utilisant la meilleure règle de priorité pour ordonner la liste des tâches, nous avons évalué les performances des deux algorithmes de sélection des employés (les algorithmes best-fit et first-fit). La deuxième comparaison est faite pour déterminer le meilleur ordre dans lequel nous appellerons les fonctions de voisinage dans les méthodes itératives. Les résultats ont été obtenus en appliquant la méthode TS à la meilleure solution initiale trouvée par la

méthode TPGH. La dernière comparaison concerne les méthodes LS et TS. Pour chaque méthode, nous avons déterminé les solutions initiales en utilisant la règle de priorité WEDD combinée à l'algorithme best-fit car ils ont donné les meilleurs résultats.

La Figure 4.5 illustre le nombre d'itérations requises pour que les méthodes LS et TS convergent vers une solution faisables. Elle illustre les méthodes LS et TS en fonction de l'amélioration du pourcentage de solutions faisables. Les expériences ont été réalisées sur deux ensembles d'instances de taille réelle. Le pourcentage de solutions réalisables est indiqué sur l'axe vertical. Le nombre d'itérations est indiqué sur l'axe horizontal.

Le Tableau 4.18 résume la qualité des solutions retournées par les méthodes TPGH, LS, TS et le modèle GPNE (en limitant le temps de calcul à 10 minutes par instance). Nous n'avons utilisé que les instances résolues par le modèle GPNE sur l'ensemble T_1^{real}. Ces valeurs correspondent au nombre de solutions respectant les contraintes souples et à l'écart moyen entre les solutions obtenues et les meilleures solutions obtenues par le modèle GPNE pour un temps de calcul limité à 2h/instance (le solveur a obtenu une solution faisable pour toutes les instances de l'ensemble T_1^{real}). Il est à noter qu'un temps de calcul supérieur à 10 min ne répond pas aux exigences de l'entreprise.

Le Tableau 4.19 montre les résultats numériques pour les instances sur l'ensemble T_2^{real}, où la GPNE n'a pas pu trouver de solutions faisables dans le temps limité à 2 heures. Les performances des trois méthodes sont mesurées en fonction du nombre de solutions réalisables, du nombre de solutions réalisables respectant les contraintes souples et du retard supplémentaire moyen des projets (mesuré par rapport aux dates de fin souhaitées). Les valeurs du retard des projets sont très pertinentes car elles donnent une idée de la manière dont les retards correspondants affectent l'avancement global des projets. Par exemple, si la date de fin souhaitée d'un projet est de 10 semaines et que l'heuristique trouve un coût de 2000 pour un retard de 2 semaines, nous avons un coût supplémentaire de 2000, un retard de deux semaines et une augmentation de la durée du projet de $2/10 = 20\%$.

Analyse expérimentale

Dans le Tableau 4.17, les résultats conduisent aux points suivants : la combinaison de l'algorithme best-fit avec la règle de priorité WEDD semble donner l'écart moyen le plus faible. L'implémentation de l'algorithme best-fit, qui sélectionne pour chaque tâche l'employé ayant la plus faible disponibilité suffisante, permet de planifier des tâches à forte charge, ce qui augmente les chances de trouver une solution faisable de bonne qualité. La fonction de voisinage $V1$ semble être légèrement plus performante que les autres fonctions de voisinage ($V2$ et $V3$). Pour la plupart des ensembles d'instances, l'écart moyen le plus faible est donné par $V1$ puis $V2$ puis $V3$. Nous observons également que la méthode TS est plus performante que la méthode LS. Si nous comparons les résultats obtenus par la méthode TPGH avec les résultats obtenus par la méthode TS, nous observons clairement que la méthode TS a considérablement amélioré la déviation des solutions initiales obtenues par la méthode TPGH. A noter que le temps de calcul des deux méthodes est inférieur à 3 minutes par instance, ce qui est très raisonnable pour une méthode itérative.

Le Tableau 4.18 montre que l'écart moyen du modèle GPNE (après 10 minutes de calcul) est inférieur à celui obtenu par la méthode TPGH. Cependant, si on compare ces résultats en terme de faisabilité, on observe que la méthode TPGH obtient un plus grand nombre de solutions faisables (le modèle n'obtient que 9 de solutions faisables sur 40). La méthode TPGH a été capable de trouver des solutions réalisables pour plus de la moitié des instances (28 de solutions réalisables sur 40). Pour le reste des instances, on peut affirmer que la méthode TPGH est capable d'atteindre des solutions avec moins de violation de contraintes lorsqu'elles ne sont pas réalisables, ce qui est confirmé par les résultats de la Figure 4.5. Ces solutions sont néanmoins utilisées comme solutions initiales pour les approches itératives. Ce qui est à souligner dans ce cas en observant la Figure

4.5 que les méthodes LS et TS n'ont pas eu besoin de faire beaucoup d'itérations pour atteindre une solution réalisable (chaque solution réalisable est obtenue avant 100 itérations). Le même tableau montre également que les solutions obtenues par la méthode TPGH sont considérablement améliorées par les méthodes TS et LS. En effet, l'écart moyen entre les solutions obtenues et les meilleures solutions obtenues avec le modèle GPNE (après 2 heures de calcul) est réduit d'au moins la moitié. Nous observons également que les méthodes itératives ont permis de respecter les contraintes souples pour un grand nombre d'instances. Notons que le temps de calcul pour obtenir ces valeurs est inférieur à 3 minutes. Nous observons également que les résultats des méthodes itératives surpassent les résultats du modèle GPNE après 10 min de calcul.

Dans les résultats numériques présentés dans le Tableau 4.19, les méthodes LS et TS ont pu obtenir des solutions réalisables pour toutes les instances T_2^{real}, ce qui confirme la pertinence de ces méthodes. Nous notons que le degré de respect des contraintes souples est très acceptable et que le retard moyen des projets est inférieur à 10% pour les deux méthodes.

Enfin, tous les résultats numériques montrent que la méthode TS est légèrement plus performante que la méthode LS. Ce qui confirme les avantages de l'utilisation de la recherche tabou dans ce contexte.

4.6 Conclusion

Nous avons étudié ici un problème d'ordonnancement multi-projet et d'affectation d'employés multi-compétences aux tâches des projets. Le problème a été soulevé par notre partenaire, la société d'IT Alfa-conseil, qui partage un pool d'employés entre plusieurs projets afin d'assurer leurs exécutions simultanées dans un horizon fixé. Un employé peut par conséquence être impliqué dans plus d'un projet en même temps, avec une quotité maximale allouée à chaque projet. Ces quotités sont le résultat d'une procédure de négociation entre les chefs de projet. Comme nous avons étudié un modèle d'ordonnancement réel, nous avons considéré les tâches et les employés à différents niveaux de détail. La solution doit respecter autant que possible des contraintes souples spécifiques, telles qu'une charge minimale de la tâche par semaine et un nombre limité de tâches différentes que chaque employé ne doit pas dépasser par semaine.

Pour résoudre le problème, nous avons proposé un modèle de programmation par objectif mixte en nombres entiers et une méthode heuristique gloutonne à trois phases qui est améliorée en appliquant un algorithme de recherche locale et un algorithme de recherche tabou. Nous avons comparé les résultats de ces méthodes en termes de qualité de solution en les appliquant à des instances de taille réelle dérivées de données réelles. Tous les algorithmes trouvent rapidement de bonnes solutions à des instances de taille réelle. Les expériences montrent ainsi l'avantage d'utiliser la méthode de recherche tabou dans ce contexte.

De plus, les résultats numériques ont prouvé l'impact majeur des quotités maximales sur la qualité des solutions retournées. Dans la suite du manuscrit, nous nous intéressons au cas où les quotas de temps des employés sur les projets ne sont pas fixés, et que plusieurs chefs de projet collaborent pour décider quel employé affecter à quel projet. Dans ce cas, chaque chef de projet doit négocier pour obtenir les ressources nécessaires pour satisfaire les contraintes du problème et optimiser ses propres critères. Ce problème peut être défini comme un problème d'ordonnancement multi-agent (Agnetis et al. (2014)) dans un environnement multi-compétence et multi-projet.

Deuxième partie

Ordonnancement de projets multi-agent

Chapitre 5

Présentation du contexte de travail

Contents

Ce chapitre étant une suite du chapitre 2, est consacré à une présentation du contexte général des problèmes évoqués dans cette partie de la thèse. Les travaux abordés dans la suite de ce document concernent les problèmes d'ordonnancement multi-agents. Dans la section 5.1, nous définissons un problème d'ordonnancement multi-agent. Dans cette même section, nous définissons un problème d'ordonnancement multi-critère afin d'éclairer les similitudes et les différences entre ces deux problèmes d'optimisation. Les notions nécessaires à la compréhension des problèmes multi-agents sont présentées dans la section 5.2. Dans la section 5.3, nous présentons les différentes classes d'ordonnancement multi-agent distinguées dans la littérature. Les principales métriques de performance proposées pour des problèmes d'optimisation multi-critères sont discutées dans la section 5.5. Dans les sections 5.6 et 5.7, nous discutons des différentes approches de résolution appliquées dans la littérature. Nous clôturons ce chapitre par une synthèse présentée dans la section 5.8.

5.1 Introduction

Le contexte actuel d'ordonnancement multi-projet est entré dans un nouvel environnement où différents acteurs (agents) sont impliqués dans le processus de l'ordonnancement. Dans certaines situations réelles, les tâches à ordonnancer peuvent appartenir à des sous-ensembles différents, pour lesquels des mesures non identiques sont à optimiser. Telle peut être la situation d'une organisation qui traite des projets pour lesquels les clients n'ont pas les mêmes particularités. Par exemple, certains clients peuvent être plus exigeants sur les délais, certains sur le coût, d'autres sur les deux en même temps, etc. De même, comme les agents partagent les mêmes ressources, il arrive que chacun souhaite optimiser un même critère appliqué sur ses tâches. À titre d'exemple, deux agents souhaitent chacun minimiser son makespan. La prise en compte de toutes ces particularités a impliqué l'introduction d'une nouvelle extension de problèmes d'ordonnancement multi-projet, dans laquelle des mesures sont appliquées sur certaines tâches et pas sur l'ensemble des tâches. De plus, les sous-ensembles des tâches sont en concurrence pour l'utilisation des ressources, ce qui peut provoquer des conflits. On parle de l'ordonnancement des sous-ensembles de tâches concurrentiels. Ce type de problèmes est appelé dans la littérature " problèmes d'ordonnancement multi-agent" (Agnetis et al. (2004)).

D'un point de vue théorique, l'ordonnancement multi-agent est une classe particulière de l'ordonnancement multi-critère (voir T'kindt and Billaut (2001)). Ce dernier est défini comme un problème d'ordonnancement dans lequel un ensemble des critères sont appliqués sur la totalité des tâches. Nous définissons plus formellement les problèmes d'optimisation multi-critères (ou multi-objectifs) ci-dessous.

Soit m le nombre des variables de décision dont les valeurs sont à choisir dans un problème d'optimisation. Ces variables sont notées x_i, $i \in \{1, ..., m\}$. On cherche la solution ou le vecteur de variables de décision $\mathbf{x} = (x_1, x_2, ..., x_m)$ qui satisfait un ensemble de contraintes et optimise simultanément un ensemble de $n \geq 2$ fonctions objectifs $(f^1, f^2, ..., f^n)$. À chaque solution $\mathbf{x}$ est affecté un vecteur objectif $\mathbf{f}$ représenté par : $\mathbf{f}(\mathbf{x}) = (f^1(\mathbf{x}), f^2(\mathbf{x}), ..., f^n(\mathbf{x}))$. Sans perte de généralité, nous supposons que les n fonctions objectifs doivent être minimisées. Sachant qu'un problème de maximisation peut être ramené simplement à un problème de minimisation : maximiser f_i revient à minimiser $-f_i$.

$$Minimiser \quad \mathbf{f}(\mathbf{x}) = (f(\mathbf{x})^1, f(\mathbf{x})^1, ..., f(\mathbf{x})^n) \tag{5.1}$$

$$s.c. \quad \mathbf{x} \in X \tag{5.2}$$

X représente l'ensemble des solutions réalisables, celui-ci peut être défini par un ensemble de contraintes vérifiées par chaque solution $\mathbf{x}$.

Comme mentionnée avant, la particularité des problèmes d'ordonnancement multi-agent réside dans le fait qu'il y a au moins un critère qui est appliqué uniquement à un sous-ensemble des tâches et non à la totalité. Tout comme les problèmes d'ordonnancement multi-critère, l'objectif est de trouver une solution de compromis (ou une solution de Pareto) entre les différents agents. En principe, identifier une ou les solutions de Pareto non dominées est un élément clé dans le processus de résolution d'un problème d'ordonnancement multi-agent.

5.2 Concepts de base et terminologie

Cette section présente la terminologie de base couramment utilisée dans la littérature des problèmes multi-critères, sachant que la même terminologie est également utilisée pour les problèmes multi-agents.

5.2.1 Notion de dominance

Comme indiqué précédemment, un problème d'optimisation multi-critère se distingue par la prise en compte explicite de plusieurs critères à optimiser en même temps. De plus, ces critères présentent souvent un aspect conflictuel. Par exemple, maximiser la performance de la voiture et minimiser sa consommation de carburant. De ce fait, on peut dire qu'il n'existe pas de critère d'optimalité unique permettant de mesurer la qualité de la solution. Cette dernière est déterminée par la valeur de la solution pour chaque critère. Une solution ne peut donc être considérée comme la meilleure de toutes les autres. Toutefois, on peut rechercher le ou les solutions qui ne sont pas dominées par aucune autre solution, dans le sens où il n'existe pas une autre solution réalisable qui atteint des meilleures valeurs pour toutes les fonctions objectifs. Ces solutions non-dominées, aussi désignées par solutions optimales de Pareto, correspondent aux meilleurs solutions de compromis. Le concept de dominance a été introduit par Vilfredo Pareto au 19ème siècle. Nous définissons plus formellement cette notion de dominance ci-dessous.

Definition 6. ***Pareto-dominance*** *Pour simplifier la présentation, on considère que A est un problème d'optimisation multi-critère et que $\boldsymbol{x}$ et $\boldsymbol{y}$ sont deux solutions à ce problème. On dit qu'une solution $\boldsymbol{x}$ domine une autre solution $\boldsymbol{y}$, si et seulement si $\boldsymbol{x}$ n'est pas pire que $\boldsymbol{y}$ pour n'importe quel objectif et si $\boldsymbol{x}$ est meilleure que $\boldsymbol{y}$ pour au moins un seul objectif. Cette relation est notée par $\boldsymbol{x} \succ \boldsymbol{y}$.*

Definition 7. ***Dominance faible de Pareto*** *Soit $\boldsymbol{x}$ et $\boldsymbol{y}$ deux solutions pour A. On dit que $\boldsymbol{x}$ domine faiblement $\boldsymbol{y}$, si et seulement si $\forall i \in \{1, 2, ..., n\}$, $f^i(\boldsymbol{x}) \leq f^i(\boldsymbol{y})$ et $\exists j \in \{1, 2, ..., n\}$ tel que $f^j(\boldsymbol{x}) < f^j(\boldsymbol{y})$. Cette relation est désignée par $\boldsymbol{x} \preceq \boldsymbol{y}$.*

Definition 8. ***Dominance stricte de Pareto*** *Soit $\boldsymbol{x}$ et $\boldsymbol{y}$ deux solutions pour A. On dit que $\boldsymbol{x}$ domine strictement $\boldsymbol{y}$, si et seulement si $\forall i \in \{1, 2, ..., n\}$, $f^i(\boldsymbol{x}) < f^i(\boldsymbol{y})$. Cette relation est désignée par $\boldsymbol{x} \prec \boldsymbol{y}$.*

Definition 9. ***Solution optimale de Pareto*** *Une solution $\boldsymbol{x^*} \in X$ est dite optimale de Pareto (ou solution efficace), si elle n'est dominée par aucune autre solution dans l'espace des solutions. Une autre définition de ce concept stipule que $\boldsymbol{x^*}$ est optimale si aucune autre solution x' ne peut améliorer la valeur d'un objectif sans entraîner simultanément une détérioration de la valeur d'un autre objectif*

5.2.2 Quelques définitions

Dans ce qui suit, nous donnons quelques définitions générales liées à l'optimisation multi-critère :

Definition 10. ***Point idéal*** *Le point idéal correspond au vecteur objectif constitué de la meilleure valeur pour chaque objectif.*

Definition 11. ***Point nadir*** *Contrairement au point idéal, le point nadir correspond au vecteur objectif composé de la pire valeur pour chaque objectif. Généralement, ce point ne correspond pas à une solution réelle. Ainsi, un point maximum est souvent utilisé comme une estimation du point nadir et mis à jour si un point plus mauvais est obtenu.*

Definition 12. ***Solutions supportées*** *Les solutions optimales de Pareto qui sont situées sur l'enveloppe convexe sont appelées solutions supportées. Ces solutions sont plus faciles à calculer. Les autres solutions optimales de Pareto sont appelées solutions non-supportées. Les solutions supportées situées aux sommets de l'enveloppe convexe sont appelées solutions extrêmes.*

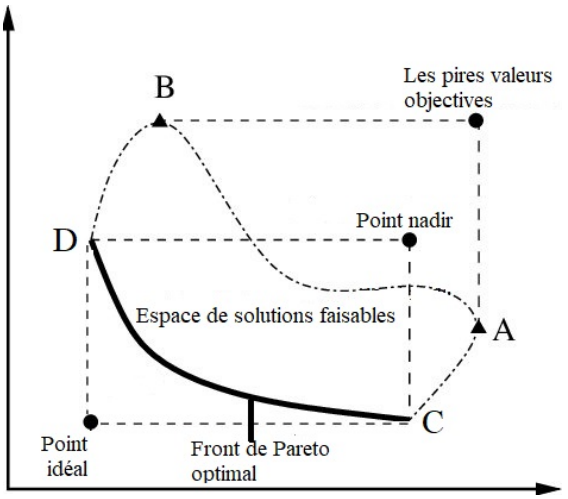


Fig. 5.1 : Illustration du point idéal et du point nadir (De et al. (2011)).

5.3 Classes d'ordonnancement multi-agent

Dans l'ouvrage de Agnetis et al. (2014), quatre classes d'ordonnancement multi-agent sont identifiées selon la relation entre les sous-ensembles des tâches des agents (s'ils partagent ou non certaines tâches). Ci-dessous, nous décrivons brièvement chacune de ces quatre classes :

- Symétrique : c'est le cas classique de l'ordonnancement multi-critère. Dans ce type de problème, chaque agent est intéressé à l'ordonnancement de toutes les tâches, car son sous-ensemble des tâches est égal à l'ensemble de toutes les tâches. Cependant, les critères des agents ne peut pas être identiques, sinon on revient vers un problème monocritère.

- Compétition : dans cette classe, chaque agent est responsable d'un sous-ensemble de tâches, qui est entièrement distinct de ceux des autres agents. Autrement dit, les agents ne partagent pas les même tâches. Les agents sont appelés par fois "agents locaux". Chaque agent local a une fonction objectif (fonction locale) qui peut être différente de celles des autres agents, et qui est appliquée uniquement sur son sous-ensemble de tâches. Toutefois, les agents locaux sont tous en concurrence pour l'utilisation de toutes ou d'une partie des ressources. Ce problème a été introduit pour la première fois par Baker and Smith (2003) et Agnetis et al. (2004).

- Interférant : c'est le cas global de la classe "Compétition", dans lequel un agent global en charge de l'ensemble des tâches des agents est considéré. Cela signifie que l'ordonnancement de toutes les tâches doit optimiser un critère (fonction globale) appliqué sur la totalité des tâches. C'est le cas par exemple du chef d'atelier souhaitant optimiser ses ressources.

- Non-Disjoint : c'est le cas le plus général, dans lequel les agents peuvent disposer des tâches en commun. En d'autres termes, l'intersection de deux sous-ensembles de tâches appartenant à deux agents quelconques n'est pas forcement vide. C'est le cas par exemple de chefs de projets qui partagent le même environnement de production.

Dans la figure 5.2 sont présentées les classes d'ordonnancement multi-agent. Seuls deux agents (1 et 2) sont considérés dans cet exemple, N_1 et N_2 sont les sous-ensembles des tâches des agents, et $objf^1$ et $objf^1$ sont les fonctions objectifs.

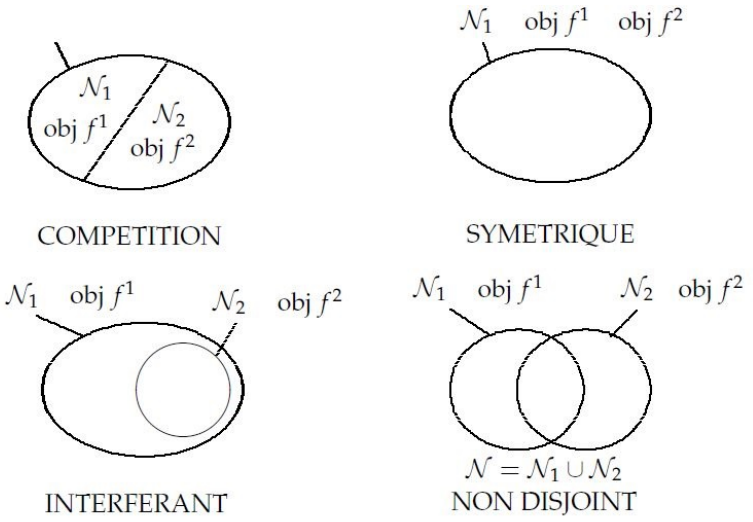


Fig. 5.2 : Les différentes classes d'ordonnancement multi-agent Agnetis et al. (2014)

5.4 Notation des classes d'ordonnancement multi-agent

Agnetis et al. (Agnetis et al. (2004)) étendent la notation de Graham pour inclure le cas d'ordonnancement multi-agent. Ils divisent le champs *beta* en deux partie : $\beta = \beta_1\beta_2$, où le champ β_1 décrit les contraintes associées aux agents (relations entre agents) et le champ β_2 décrit les contraintes associées aux tâches. Les différentes valeurs qui peuvent prendre le champ β_1 sont les suivantes.

- $\alpha \mid CO, \beta_2 \mid \gamma$: où CO représente le cas de sous-ensembles disjoints en compétition.
- $\alpha \mid BI, \beta_2 \mid \gamma$: où BI indique un problème d'ordonnancement bicritère.
- $\alpha \mid MU, \beta_2 \mid \gamma$: où MU indique un problème d'ordonnancement multicritère symétrique.
- $\alpha \mid IN, \beta_2 \mid \gamma$: où IN représente un problème d'ordonnancement interférant.
- $\alpha \mid ND, \beta_2 \mid \gamma$: où ND représente un problème d'ordonnancement non disjoint.

5.5 Évaluation de performance

Tout comme dans l'optimisation multi-critère, la résolution à l'optimal d'un problème d'optimisation multi-agent consiste à l'identification de l'ensemble des solutions non-dominées (ou front de Pareto optimal). Mais la génération de cet ensemble est très souvent impossible, en raison de la complexité du problème ou du grand nombre de solutions admissibles. Par conséquent, l'objectif global est souvent d'identifier une bonne approximation efficace de cet ensemble, d'où l'utilité des méthodes approchées.

La performance d'une méthode approchée est mesurée en comparant la qualité du front de Pareto approché, déterminé par celle-ci, avec le front de Pareto exacte. Selon Zitzler et al. (2000), cette comparaison s'effectue sur la base des critères suivants :

- La distance entre le front de Pareto approché et le font de Pareto exact. Plus cette distance est minimale plus le front de Pareto approché est bon.

– La diversité des points du front de Pareto approché, c'est-à-dire qu'un sous-ensemble représentatif du front total est souhaité.

– L'étendue du front approché correspondant doit être maximisée, c'est-à-dire qu'une large gamme des valeurs doit être couverte par les points non-dominés.

Plusieurs métriques (ou indicateurs) de performance sont déjà proposées dans la littérature, et des états-de-l'art sur ces indicateurs peuvent être trouvées par exemple dans Zitzler et al. (2008) ; Li and Yao (2019). Parmi les métriques de performance les plus utilisées, on trouve les suivantes.

5.5.1 Distances générationnelle

Cette métrique (proposée par Veldhuizen and Lamont (2000)) permet de mesurer à quelle distance se situe un ensemble de solutions obtenues (un front approché A, par exemple) par rapport à un ensemble de référence qui est dans l'idéal le front exact ($\mathcal{F}^*$). Pour chaque solution de A, la distance moyenne par rapport à la plus proche solution de $\mathcal{F}^*$ est calculée. La distance générationnelle (GD) entre les deux ensembles de solutions $(A, \mathcal{F}^*)$, correspond alors à la moyenne de ces distances. La formule générale pour calculer cette distance est comme suit :

$$GD(A, \mathcal{F}^*) = \frac{1}{|A|}(\sum_{a \in A}(d_a)^l)^{1/l}) \tag{5.3}$$

avec $|A|$ est le nombre de solutions dans A. d_a est la distance euclidienne entre le point $a \in A$ et le point le plus proche de $\mathcal{F}^*$, $l = 2$.

5.5.2 Hypervolume

La métrique d'hypervolume (proposée pour la première fois par Zitzler and Thiele (1998)) mesure la taille de la portion de l'espace des solutions dominées collectivement par celles-ci. Son calcule nécessite la définition d'un point de référence (point de pire cas). Par exemple, le point origine peut être utilisé comme point de référence.

Comme illustré dans la Figure 5.3, les points avec le marqueur x sont des solutions dans le front de Pareto (solutions non dominées). Pour chaque solution dont la valeur est donnée par (f^A, f^A) dans un cas bi-critère, on calcule la surface du rectangle défini par le point de référence (l'origine (0,0)) et le point (f^A, f^B). La somme de ces surfaces (zone en gris) est l'hypervolume du front. Pour comparer deux fronts de Pareto, la différence de leur hypervolume donne une indication de la distance entre ces deux fronts. Le meilleur front a un hypervolume plus élevé ou une surface plus grande.

5.5.3 Taille du front de Pareto

La valeur de cette métrique est égale à la taille du front de Pareto (c'est-à-dire le nombre de solutions uniques dans le front de Pareto). Une valeur plus élevée signifie une meilleure diversité des solutions, et la maximisation de cette valeur est donc souhaitée. Il convient de noter que cette métrique ne s'intéresse pas à la proximité des solutions obtenues par rapport au front optimal de Pareto ni à la proximité des solutions entre elles. Néanmoins, sa valeur peut être facilement interprétée et est directement liée au nombre de solutions uniques fournies.

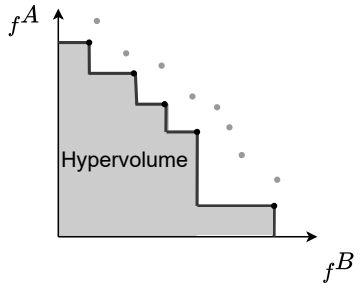


Fig. 5.3 : Métrique de performance hypervolume.
Les points noires appartiennent du même front de Pareto.

5.6 Approches de résolution considérées

Il existe de nombreuses méthodes pour trouver une ou l'ensemble des solutions de Pareto. Ces méthodes peuvent être classées dans les catégories suivantes :

- Méthodes à posteriori : elles utilisent l'information sur les préférences du décideur relatives à chaque objectif et génèrent itérativement un ensemble de solutions optimales de Pareto.
- Méthodes à priori : ces méthodes utilisent davantage les informations sur les préférences des objectifs. Elles transforment le problème en un problème mono-critère, en appliquant une agrégation des critères ou en désignant un seul critère à optimiser alors que les autres sont contraints.
- Méthodes interactives : ces méthodes utilisent les informations, de préférence de manière progressive au cours du processus d'optimisation. Cela signifie que le décideur est sollicité progressivement pour affiner son choix.
- Méthodes sans préférence : ces méthodes ne supposent aucune information sur l'importance des objectifs. Il s'agit uniquement d'une méthode de résolution qui recherche une solution faisable, sans nécessairement avoir un critère à optimiser.

Nous présentons ci-dessous uniquement les méthodes qui nous intéressent et qui sont utilisées dans la suite de cette thèse.

5.6.1 Méthode ε-contrainte

La méthode ε-contrainte a été proposée pour la première fois par Haimes HAIMES (1971). L'idée de cette méthode est de trouver une solution minimisant un seul objectif, tout en gardant les restes des objectifs bornées par certaines valeurs spécifiées par le ou les décideurs. Pour simplifier la présentation, considérons le cas de deux agents A et B. On veut minimiser les retards des tâches de l'agent A (Minimiser f^A) sous-contrainte que les retards des tâches de l'agent B soient inférieurs à une valeur Q_B donnée ($f^B \leq Q_B$). En effet, pour chercher une solution optimale de Pareto avec une valeur Q_B donnée, on peut procéder comme suit. (i) Résoudre le problème en minimisant f^A, sous contrainte que $f^B \leq Q_B$. La solution trouvée est optimale pour le critère A, notée (f^{*A}, Q_B). (ii) Résoudre une deuxième fois le problème en minimisant f^B, sous contrainte que $f^A \leq f^{*A}$. La solution ainsi obtenue est optimale au sens de Pareto strict. La modification systématique de la valeur de Q_B conduit à la génération du front optimal de Pareto. Figure 5.4 montre un exemple de ce processus.

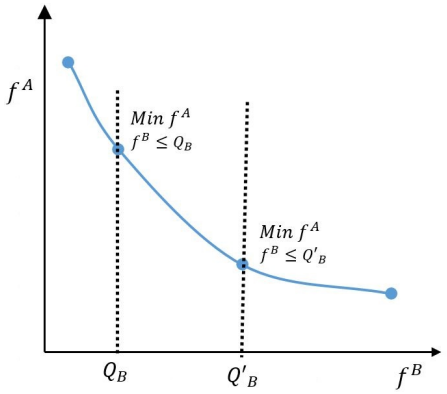


Fig. 5.4 : Méthode ε-contrainte pour deux objectifs.

5.6.2 Méthode de combinaison linéaire des critères

La méthode de combinaison linéaire des critères a été proposée pour la première fois par Geoffrion en 1968 (Geoffrion (1968)). Cette méthode consiste à ramener un problème d'optimisation multi-critère à un problème d'optimisation mono-critère, en appliquant une agrégation des critères. Le problème mono-critère résultant va optimiser donc une seule fonction objectif qui est la somme pondérée de tous les objectifs. Pour un problème de minimisation des objectifs des agents, la fonction objectif s'exprime de la façon suivante (par exemple, cas de deux agents A et B) :

$$Minimiser \quad F = \alpha f^A + (1-\alpha) f^B \tag{5.4}$$

Le coefficient α ($\alpha \in [0,1]$) est choisi en fonction de l'importance des objectifs, ils répondent aux préférences des décideurs concernant les objectifs.

Cependant, les solutions qui peuvent être obtenues par cette méthode ne sont qu'un sous-ensemble de l'ensemble des solutions optimales de Pareto (approche ne peut pas déterminer les solutions non supportées). En effet, il peut être impossible de fixer les poids des critères de manière à obtenir toutes les solutions optimales de Pareto Agnetis et al. (2014).

5.7 Algorithme génétique : NSGA-II

L'algorithme NSGA-II (pour Non-dominated Sorting Genetic Algorithm II) est un algorithme génétique bien connu comme l'un des algorithmes les plus efficaces et populaires pour résoudre des problèmes d'optimisation multi-critères. Le NSGA-II est proposé par Deb et al. (2002) sur la base de l'algorithme génétique NSGA Srinivas and Deb (1994). La version standard de NSGA-II, telle qu'elle est définie dans Deb et al. (2002), peut être décrite comme suit. Initialement, une population parente $P(0)$ de N individus est générée (soit aléatoirement ou par des heuristiques). Ensuite à chaque itération t, une population enfant $Q(t)$ (de taille N) est crée à partir de la population parente en appliquant des opérateurs de sélection, de croisement, et de mutation. La population parente $P(t)$ et la population enfant $Q(t)$ sont ensuite fusionnées en $R(t)$ (de taille $2N$). Les individus (ou les solutions) de $R(t)$ sont classés sur la base du principe de non-dominance dans un certain nombre de rangs en utilisant une méthode de tri efficace. L'algorithme génétique associe une valeur de performance (fitness) à chaque individu ce qui permet à l'algorithme de comparer les performances (qualités) des solutions. Toutes les solutions non-dominées de la population sont placées dans le premier front (rang 1), puis elles sont supprimées de la population $R(t)$. Ensuite, toutes les solutions

de la population non-dominées sont placées dans le second front (rang 2), puis elles sont supprimées également de la population $R(t)$. Et ainsi de suite. Ce processus de classification des individus est répété jusqu'à ce que chaque individu dans la population $R(t)$ soit associé à un rang. Afin de choisir N individus à croiser et à muter pour générer la nouvelle population parente $P(t+1)$, un processus de sélection élitiste est appliqué sur la population $R(t)$. Ce processus sélectionne chaque individu sur la base du degré de domination de son front. Autrement dit, les individus de rang 1 sont sélectionnés en premier (le rang 1 est le meilleur) puis les individus de rang 2, et ainsi de suite. Les individus de chaque front sélectionné sont ajoutés entièrement à la population $P(t+1)$ jusqu'à ce que la taille de $P(t+1)$ soit égale à N. Il peut arriver que les individus du dernier front sélectionné ne puissent pas intégrer entièrement la population $P(t+1)$. Cela est dû au fait que la taille de $P(t+1)$ ne doit pas dépasser N individus. Afin de choisir exactement N individus, une autre mesure de tri est appliquée pour évaluer tous les individus du dernier front sélectionné et insérer les meilleurs dans la population $P(t+1)$. Ce processus de tri des individus (appelé crowding distance) mesure la densité des solutions autour d'un individu. Plus un individu est loin des autres, plus il est favorisé. La crowding distance associe à chaque individu un rang en fonction de la proximité de son fitness par rapport aux individus ayant les fitness les plus proches. Comme le montre la Figure 5.5, la crowding distance calcule la distance moyenne sur chaque objectif, entre les deux points les plus proches situés de part et d'autre de l'individu. L'individu avec la crowding distance la plus élevée est sélectionné. La nouvelle population $P(t+1)$ est maintenant utilisée pour la sélection, le croisement et la mutation pour créer une nouvelle population $Q(t+1)$ également de taille N. Les différentes étapes de cet algorithme sont présentées dans la figure 5.6.

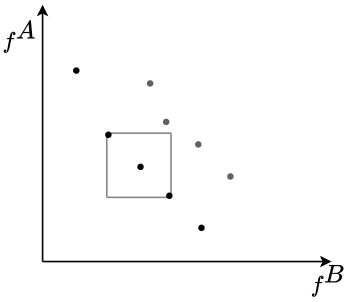


Fig. 5.5 : Tri selon la crowding distance.

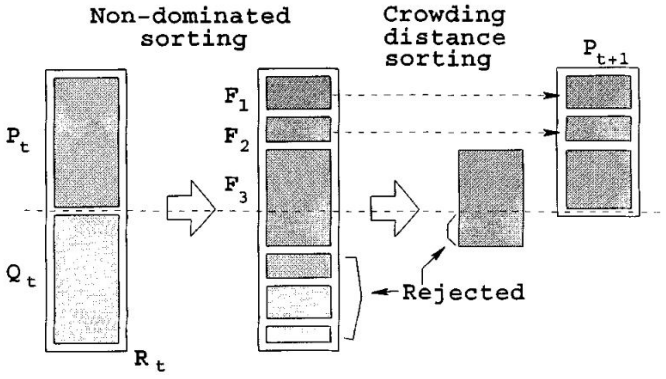


Fig. 5.6 : Procédure de NSGA-II (Deb et al. (2002)).

5.8 Conclusion

À travers ce chapitre, nous avons présenté le contexte général des problèmes d'ordonnancement multi-agent. Tout d'abord, nous avons défini les problèmes d'ordonnancement multi-critère et les problèmes d'ordonnancement multi-agent dans la section 5.1. Puis, nous avons précisé les similitudes et les différences entre ces deux problèmes d'optimisation. En outre, nous avons présenté dans la section 5.2 les notions, définitions et notations utilisées dans la littérature des problèmes multi-agents. Dans la section 5.3, nous avons souligné les différentes classes d'ordonnancement multi-agent distinguées dans la littérature. Les principales métriques de performance proposées pour des problèmes d'optimisation multi-critère ont été abordées dans la section 5.5. Enfin, nous avons discuté des différentes approches de résolution appliquées dans la littérature dans les sections 5.6 et 5.7. Comme approche de résolution, nous avons souhaité rappeler en particulier la méthode NSGA-II qui a montré ses performances pour la résolution des problèmes d'optimisation multi-critère. Cette approche que nous avons utilisée pour résoudre le problème d'ordonnancement multi-agent que nous avons identifié.

Dans le chapitre suivant, nous présentons une revue de la littérature variée portant sur les problèmes d'ordonnancement à deux agents.

Chapitre 6

État de l'art : problèmes d'ordonnancement à deux agents

Contents

Ce chapitre est consacré à une revue de la littérature des problèmes connexes à ceux que nous étudions dans la deuxième partie de la thèse. Pour réaliser cette revue, nous avons regroupé les travaux existants selon le nombre de machines considérées. A savoir, le cas où les deux agents sont en concurrence pour exécuter leurs tâches sur une seule machine et le cas où ils sont en concurrence sur plusieurs machines.

6.1 Problèmes avec deux agents en compétition sur une seule machine

Dans ce type de problèmes d'ordonnancement à deux agents, chaque agent gère un sous-ensemble de tâches qui sont entièrement distinctes de celles des autres agents, c'est-à-dire que les agents n'ont pas de tâches en commun. L'objectif est de déterminer les solutions de compromis susceptibles de satisfaire les critères de chaque agent. Nous faisons face donc à une autre variante des problèmes d'ordonnancement multi-critère.

Un nombre croissant de travaux impliquant diverses contraintes et critères de performance ont été largement étudiés ces derniers années. Compte tenu de l'ampleur des recherches sur ce type de

problèmes d'ordonnancement, nous subdivisons les études correspondantes en quatre catégories, auxquelles nous ferons référence plus en détail ci-dessous.

6.1.1 Objectifs liés aux avances et aux retards

Les pionniers de la recherche sur l'ordonnancement multi-agent sont Baker et Smith Baker and Smith (2003) et Agnetis et al. Agnetis et al. (2004). Ils s'intéressent à des problèmes d'ordonnancement à deux agents en compétition sur une seule machine, avec des tâches non-préemptives, des dates de fin souhaitées et un critère à optimiser par agent. Plus précisément, Baker et Smith se focalisent sur la minimisation d'une fonction objectif qui combine les objectifs des deux agents, notamment le makespan C_{max}, le retard algébrique maximum L_{max} et la somme pondérée des dates de fin d'exécution. Cependant, Agnetis et al. abordent plusieurs problèmes de type optimisation sous contraintes, où l'objectif est de minimiser le critère d'un agent, tout en maintenant la valeur du critère de l'autre agent à un niveau inférieur ou égal à un seuil donné. Les fonctions objectifs considérées, comprennent la somme pondérée des dates de fin des tâches $\sum w_j C_j$, le nombre de tâches en retards $\sum U_j$, et le maximum de certaines fonctions régulières (associées à chaque tâche) f_{max} (où $f_{max} = max_j(f_j(C_j))$ et $f_j(C_j)$ est une fonction non décroissante par rapport à la date de fin d'exécution C_j). Ng et al. (2006) étudient un modèle similaire au modèle traité par Agnetis et al. (2004), ayant comme objectif la minimisation de la somme des dates de fin des tâches du premier agent, sous contrainte d'une borne supérieure sur le nombre de tâches en retard du second agent. Le même modèle est considéré récemment par Li et al. Li et al. (2021a), ayant comme objectif la minimisation du nombre pondéré de tâches en retard du premier agent, à condition que le nombre de tâches en retard de l'autre agent ne dépasse pas un seuil donné. Les auteurs de Lee et al. (2015) abordent un modèle à deux agents en compétition sur une seule machine, où l'objectif est de minimiser la somme des retards des tâches d'un agent, sous contrainte que le retard absolu maximum T_{max} des tâches de l'autre agent ne peut pas dépasser une limite supérieure. Wang et al. Wang et al. (2017a)) abordent également un modèle similaire ayant pour objectif la minimisation de la somme des charges de tâches en retard du premier agent, avec une borne supérieure sur le retard maximum L_{max} du second agent. Dans Zhang and Wang (2017), les auteurs étudient le problème de la minimisation de la somme pondérée des charges de tâches en retard ($\sum w_j V_j$) pour le premier agent, sous condition que la fonction régulière du coût maximum (f_{max}) du deuxième agent ne dépasse pas un seuil donné. Plus récemment, Zhang (2021) s'est intéressé à un modèle similaire aux deux modèles ci-dessus, ayant pour objectif de minimiser la somme pondérée des charges de tâches en retard ($\sum w_j V_j$) pour le premier agent, tout en maintenant la somme des dates de fin des tâches du second agent inférieure à un seuil donné. Dans Cheng et al. (2006), les auteurs étudient principalement le problème de faisabilité d'ordonnancement à deux agents en compétition sur une seule machine, où l'objectif de chaque agent est de minimiser le nombre pondéré de tâches en retard ($\sum w_j U_j$). Le problème de la faisabilité cherche à vérifier s'il existe un ordonnancement qui maintient le nombre de tâches en retard pour chaque agent A en dessous du nombre toléré ($\sum w_j U_j \leq Q_A$). Les auteurs montrent que le problème est $\mathcal{NP}$-complet au sens fort. Cheng et al. C.E.Cheng et al. (2011) étendent le travail de Ng et al. (2006) pour intégrer l'effet de l'apprentissage. L'objectif dans leur étude est de minimiser la somme pondérée des dates de fin des tâches du premier agent, en interdisant les retards des tâches du second agent. Les auteurs de Yin et al. (2012a) considèrent deux agents concurrents sur une seule machine, avec des dates de fin à attribuer aux tâches. L'objectif est d'attribuer à chaque tâche une date de fin parmi un ensemble donné de dates de fin et une position dans la séquence d'ordonnancement, tout en minimisant la somme pondérée des objectifs des deux agents. Plusieurs combinaisons de fonctions objectifs sont considérées, notamment le retard algébrique maximum L_{max}, la somme (pondérée) des retards absolus $\sum T_j$ et $\sum w_j T_j$ ou encore le nombre total (pondéré) de tâches en retard $\sum U_j$ et $\sum w_j U_j$. Wang et al. Wang et al. (2015) étendent le travail présenté dans Yin et al. (2012a) pour minimiser l'objectif du premier agent sous une contrainte sur l'objectif du second agent. Plusieurs problèmes

sont étudiés provenant de différentes combinaisons de fonctions objectifs, notamment le retard maximum L_{max}, la somme des retards (pondérés) $\sum T_j$ ($\sum w_j T_j$), le nombre (total pondéré) de tâches en retard $\sum U_j$ ($\sum w_j U_j$), et la somme pondérée des retards et des avances ($\sum w_j(E_j+T_j)$). Les auteurs de Gerstl and Mosheiov (2013) étudient plusieurs problèmes d'ordonnancement avec deux agents en compétition sur une seule machine et sur m machines parallèles. Toutes les tâches des agents ont des durées unitaires ($p_j = 1$) et une date de fin souhaitée commune. Dans tous les problèmes traités, l'objectif est de minimiser la somme pondérée des retards et des avances des tâches du premier agent, sous contrainte d'une borne supérieure sur la déviation maximale (par rapport à la date de fin souhaitée) des tâche du second agent. Récemment, dans Choi et al. (2020), les tâches en retard ont été soumises à des pénalités qui peuvent être évitées en compressant les durées de certaines tâches, ce qui a ajouté un coût supplémentaire. L'objectif de chaque agent est de minimiser le coût total des pénalités des retards plus le coût total de la compression des durées des tâches. Deux problèmes ont été considérés : le premier minimise la somme pondérée des valeurs des deux objectifs, et le second minimise la valeur de l'objectif d'un agent à condition que la valeur de l'objectif de l'autre agent ne dépasse pas un seuil donné. Plus récemment, les auteurs de Li et al. (2021b) ont considéré un modèle d'ordonnancement à deux agents en compétition sur une seule machine, dans lequel les tâches sont associées à des revenus et à des coûts de réalisation. L'acceptation de toutes les tâches peut entraîner des retards de livraison des autres tâches et une perte des revenus attendues. Par conséquent, certaines tâches peuvent être rejetées afin de maximiser les revenus des tâches acceptées. L'objectif dans leur modèle est de maximiser le revenu net, tout en maintenant le nombre pondéré de tâches en retard $\sum U_j$ pour le deuxième agent en dessous d'une valeur prédéterminée.

6.1.2 Objectifs liés aux dates de fin et aux makespans

Agnetis et al. Agnetis et al. (2009) abordent plusieurs problèmes consistant à minimiser la somme pondérée des dates de fin d'exécution des tâches du premier agent, avec une limite sur la fonction objectif du second agent. Cette dernière peut être : (i) la somme pondérée des dates de fin d'exécution, (ii) le retard algébrique maximum L_{max}, (iii) le makespan C_{max}. Les auteurs de Li et al. (2016) s'intéressent à un modèle d'ordonnancement à deux agents avec des setup times entre les tâches des agents. À savoir, un setup time est nécessaire lorsqu'une tâche d'un agent est traitée après la tâche d'un autre agent. Plusieurs combinaisons de fonctions objectifs sont considérées, notamment le retard algébrique maximum L_{max}, la somme (pondérée) des dates de fin d'exécution $\sum C_j$ ($\sum w_j C_j$) et le nombre (pondéré) des tâches en retard $\sum U_j$ ($\sum w_j U_j$). Chaque combinaison de fonctions objectifs conduit à un problème dont l'objectif est de minimiser la fonction objectif d'un agent, à condition que la valeur de la fonction objectif de l'autre agent ne dépasse pas un seuil donné. Sahu et al Sahu et al. (2018) considèrent également des setup times entre les tâches de différentes familles. L'objectif dans leur étude consiste à déterminer un ordonnancement qui minimise la somme pondérée des dates de fin des tâches du premier agent, sous réserve d'une borne supérieure sur le makespan du second agent.

6.1.3 Travaux avec une seule machine à traitement par batch

Mor et al. Mor and Mosheiov (2011) étudient un problème d'ordonnancement à deux agents sur une machine à traitement par batch en série [1] (notée *"s-batch"*). Les auteurs considèrent que les tâches ont des durées unitaires ($p_j = 1$) et que le lancement de chaque batch nécessite un setup time identique pour tous les batches. L'objectif est de minimiser la somme des dates de fin des tâches d'un agent $\sum C_j^A$, sous contrainte d'une borne supérieure sur les dates de fin des tâches

[1] n tâches doivent être traitées dans des batches sur une seule machine. La date de fin d'une tâche est donnée par la somme des durées de toutes les tâches constituant le batch.

de l'autre agent $\sum C_j^A \leq Q$. Les auteurs de Sabouni and Jolai (2010) considèrent une machine de type "p-batch" (batch parallèle) [2]. Différents problèmes sont envisagés où la capacité du batch est non bornée et les tâches sont compatibles, et où la capacité du batch est bornée et les tâches des deux agents sont compatibles et non compatibles. Le concept de tâches compatibles signifie qu'un batch peut contenir des tâches qui appartiennent aux différents agents. Les critères considérés sont la minimisation du makespan C_{max} et du retard maximum L_{max}. Li et al. Li and Yuan (2012) considèrent une seule machine à traitement par batch parallèle non-bornée. Ils considèrent également les cas où les deux familles de tâches sont respectivement compatibles et incompatibles. Différentes fonctions objectifs sont combinées, notamment le coût maximum f_{max}, le coût total $\sum f_j$ et le nombre des tâche en retard $\sum U_j$. Dans tous les problèmes étudiés, où chacun résulte d'une combinaison de deux fonctions objectifs, le problème consiste à trouver un ordonnancent qui minimise la fonction objectif d'un agent, en maintenant la valeur de la fonction objectif de l'autre agent en dessous d'un seuil donné. Les auteurs de Yin et al. (2013b) étendent les modèles ci-dessus pour inclure des coûts de livraison associés aux batches, des pénalités des avances des tâches, des pénalités des attentes des tâches, et des fenêtres d'exécution communes aux tâches à déterminer par le processus de l'ordonnancement. L'objectif est de trouver la taille optimale et l'emplacement optimal de la fenêtre, la date optimale de la livraison de chaque tâche et une séquence optimale de tâches pour minimiser une fonction de coût basée sur l'avance, le retard, le temps d'attente, l'emplacement de la fenêtre, la taille de la fenêtre et la livraison par batch. Un modèle similaire au modèle ci-dessus est considéré dans Yin et al. (2016b). Plusieurs problèmes sont étudiés provenant de différentes combinaisons de fonctions objectifs, y compris la somme des dates de fin d'exécution $\sum C_j$, le nombre pondéré de tâches en retard $\sum w_i U_j$, et le retard maximum L_{max} plus la somme des coûts de livraison des batches. L'objectif dans chacun des problèmes étudiés est de minimiser la fonction objectif d'un agent, avec une limite supérieure sur la valeur de la fonction objectif de l'autre agent. Kovalyov et al. Kovalyov et al. (2015) considèrent l'ordonnancement des tâches incompatibles sur une seule machine à traitement par batch en série non-bornée. Un setup time spécifique à chaque agent est considéré avant l'exécution de la première tâche de chaque batch d'un agent. Les tâches sont associées à des dates de fin souhaitées et des pénalités de retard. Plusieurs type de fonctions objectifs sont considérés, notamment le makespan C_{max}, le retard algébrique maximum L_{max}, le coût maximum f_{max}, la somme pondérée des dates de fin $\sum w_j C_j$, et le nombre pondéré des tâches en retard $\sum w_j U_j$. Dans tous les problèmes étudiés, dont chacun résulte d'une combinaison de fonctions objectif, le problème consiste à trouver un ordonnancement qui minimise la fonction objectif d'un agent, à condition que la valeur de la fonction objectif de l'autre agent ne dépasse pas un seuil donné. Dans Yin et al. (2021), les auteurs étendent le travail publié dans Yin et al. (2013b) pour intégrer des dates de fin des tâches. Ces dernières sont attribuées par le décideur en utilisant certains modèles d'affectation des dates de fin. De plus, les auteurs considèrent également un setup time avant l'exécution de la première tâche de chaque batch. Les fonctions objectifs des agents sont : (i) le coût total comprenant les coûts des avances, des retards, de la livraison des batches et les temps des attentes des tâches avant leurs livraisons ; (ii) le coût total comprenant les avances, le nombre pondéré de tâches en retard, et les coûts de livraison des batches. L'objectif dans leur étude est également de minimiser la fonction objectif d'un agent, sous contrainte d'une borne supérieure sur la fonction objectif de l'autre agent. Fan et al. Fan et al. (2013) considèrent une machine de type "p-batch" (batch parallèle). Ils se concentrent sur la minimisation du makespan ou la somme des dates de fin des tâches d'un agent, sous contrainte d'une limite supérieure sur le makespan de l'autre agent. Wang et al. Wang et al. (2017c) considèrent un modèle similaire, avec des tâches de durées d'exécution unitaires ($p_j = 1$). L'objectif est de minimiser le makespan du premier agent sous réserve d'une limite supérieure sur le makespan de l'autre agent.

[2] Une machine de traitement par batch en parallèle. Elle peut traiter jusqu'à n nombre de tâches simultanément dans un batch. Le temps de traitement de chaque batch est égal au temps de traitement le plus long des tâches de ce batch.

6.1.4 Travaux avec des dates de début

Wu et al. Wu et al. (2013a) s'intéressent à un modèle à deux agents en compétition sur une seule machine, avec des dates de début au plus tôt. L'objectif est de minimiser la somme des dates de fin d'exécution du premier agent, avec la restriction que la somme des dates de fin d'exécution du second agent ne dépasse pas un seuil donné. Liu et al. Liu et al. (2019) étudient un modèle similaire, où l'objectif est d'ordonnancer toutes les tâches de telle sorte qu'une combinaison linéaire des makespans de chaque agent soit minimisée. Les auteurs de Lee et al. (2012) étendent le problème ci-dessus pour inclure des dates de fin souhaitées. L'objectif est de minimiser la somme des retards absolus $\sum T_j$ du premier agent, étant donné que le retard maximum (L_{max}) du second agent ne dépasse pas une limite supérieure. Cheng et al. Cheng et al. (2013) étudient également un modèle similaire au modèle précédent, ayant comme objectif la minimisation de la somme pondérée des dates de fin d'exécution des tâches du premier agent, sous contrainte que le retard maximum L_{max} du second agent ne dépasse pas un seuil donné. Un modèle similaire est étudié également dans Yin et al. (2013a), ayant comme objectif la minimisation du nombre de tâches en retard $\sum U_j$ d'un agent, sous la contrainte que le retard maximum L_{max} des tâches de l'autre agent ne dépasse pas une valeur donnée. Yin et al. Yin et al. (2012c) considèrent des dates de début arbitraires et de fin souhaitées. L'objectif est de minimiser la somme des retards absolus $\sum T_j$ d'un agent, tout en maintenant le retard algébrique maximum L_{max} de l'autre agent en dessous ou à un niveau fixe. Wang et al. Wang et al. (2017b) considèrent des dates de début arbitraires. L'objectif du premier agent est de minimiser la date de fin maximale pondérée w_jC_j de ses tâches, tandis que l'objectif du second agent est de minimiser la somme pondérée des dates de fin de ses tâches. Les auteurs de Gawiejnowicz and Suwalski (2014) considèrent un modèle d'ordonnancement à deux agents avec des dates de début au plut tôt communes. L'objectif dans leur étude est de minimiser simultanément la somme de dates de fin d'exécution $\sum w_jC_j$ et le retard maximum L_{max}. Wan et al. Wan et al. (2010) considèrent plusieurs problèmes d'ordonnancement à deux agents avec des tâches préemptives et contrôlables. Une tâche contrôlable signifie que sa durée peut être compressée (en lui allouant des ressources supplémentaires). Les durées des tâches du première agent sont contrôlables avec un coût supplémentaire ajouté, alors que les durées des tâches du deuxième agent ne le sont pas. Chaque tâche du deuxième agent est soumise à une fonction de pénalité qui dépend de la date de fin de la tâche. L'objectif de ce dernier agent est de garder la fonction de coût f_{max} en dessous d'une valeur donnée. Cependant le première agent considère plusieurs fonctions objectifs, notamment les dates de fin d'exécution des tâches plus le coût de compression, le retard absolu maximum T_{max} plus le coût de compression, et le retard algébrique maximum (L_{max}) plus le coût de compression. Le problème consiste à minimiser la fonction objectif du première agent, sous contrainte d'une borne supérieure sur la fonction objectif du deuxième agent.

6.1.5 Travaux avec des détériorations linéaires des durées de tâches

D'autres travaux ont intégré le concept de détérioration linéaire dans des modèles d'ordonnancement à deux agents sur une seule machine. Le concept de détérioration linéaire suppose que la durée d'une tâche est une fonction linéaire habituellement croissante de sa date de début. Autrement dit, la durée de la tâche augmente avec le retard de sa date de début.

Les travaux de Liu et al. Liu et al. (2011) intègrent des détérioration linéaires des tâches. Les auteurs étudient deux modèles d'ordonnancement à deux agents avec des différentes combinaisons des fonctions objectifs : makespan C_max, retard maximum L_{max}, coût maximum et la somme des dates de fin $\sum C_j$. Wu et al. Wu et al. (2013b) envisagent également un modèle similaire ayant comme objectif la minimisation du nombre pondéré de tâches en retard du premier agent, à condition que le retard maximal du second agent ne dépasse pas un seuil donné. À la suite des travaux Liu et al. (2011), Yin et al. Yin et al. (2015) étudient les différentes combinaisons des fonctions objectifs régulières, notamment le coût maximum f_{max}, la somme pondérée des dates de

fin d'exécution $\sum wjC_j$, le coût de l'avance maximale, la somme des avances $\sum E_j$ et la somme pondérée des avances. L'objectif dans chacun des problèmes étudiés est de minimiser la fonction objectif d'un agent, sous une limite supérieure sur la valeur de la fonction objectif de l'autre agent. Les auteurs de Gawiejnowicz and Suwalski (2014) considèrent en plus des dates de début au plus tôt communes et une fonction objectif minimisant la somme pondérée des deux critères. Notamment la somme des dates de fin d'exécution $\sum w_jC_j$ et le retard maximum L_{max}. Les auteurs de Liu et al. (2013) étudient une nouvelle variante d'ordonnancement à deux agents avec des détériorations cumulatives des tâches. Par détérioration cumulative les auteurs entendent que la durée nominale d'une tâche est une fonction linéaire croissante de la somme des durées des tâches déjà traitées sur la même machine. L'objectif dans leur étude est de minimiser la somme des dates de fin d'exécution d'un agent, tandis que le makespan de l'autre agent ne peut pas dépasser une limite supérieure donnée. Récemment, Chen et al. Chen and Li (2019) ont étudié le même modèle, avec les différentes combinaisons des fonctions objectifs réguliers, notamment le coût maximum f_{max}, la somme des dates de fin $\sum C_j$ et le nombre (pondéré) de tâches en retard $\sum U_j$ ($\sum w_jU_j$). Les auteurs de Yin et al. (2012b) s'intéressent à un modèle similaire au modèle étudié dans Liu et al. (2011) avec des détériorations linéaires non-croissantes. Dans leur modèle, toutes les tâches doivent être achevées au plus tôt à une date de fin commune. En effet, une tâche peut être trivialement ordonnancée suffisamment tard afin d'éviter le coût de l'avance associé à cette tâche. Trois fonctions objectifs différentes sont adoptées pour un agent, à savoir le coût maximal de l'avance f_{max}, la somme des coûts des avances et la somme pondérée des coûts des avances, tout en maintenant le coût maximal de l'avance de l'autre agent à un niveau inférieur ou égal à un niveau fixe.

6.1.6 Synthèse

Dans le tableau 6.1, nous présentons une synthèse des travaux rapportés ci-dessus. Comme mentionné précédemment, tous ces travaux se concentrent sur des modèles avec deux agents en compétition sur une seule machine. Certains peuvent être généralisés au cas de plusieurs agents. Cependant, ils peuvent être différenciés selon les caractéristiques de la machine, des tâches ou des objectifs. La première colonne du tableau représente la référence de l'article. En adoptant la notation à trois champs introduite par Agnetis et al. Agnetis et al. (2004), la deuxième colonne indique la nature du problème traité. La troisième colonne montre les méthodes de résolution développées, suivis des instances utilisées dans la colonne 4. La dernière colonne indique si l'étude considère d'autres caractéristiques non indiquées dans la colonne 2.

Le tableau 6.1 utilise de nombreux acronymes et abréviations pour indiquer soit les contraintes, l'objectif ou la méthode de résolution. Certains d'entre eux ont déjà été présentés dans la Section 3.3.3 du Chapitre 3. Nous ne présentons ici que les nouveaux résultats.

Concernant les contraintes : D_1 : b-batch (machine à traitement par batch non bornée), $agent-spc$ (batch spécifique à l'agent) ; $D_2 : b-batch$, $D_3 : s-bacth$ (machine à traitement par batch en série) ; $D_4 : p_j = 1, p-batch$ (machine à traitement par batch en parallèle) ; $D_5 : p-batch, agent-spc$; D_6 : $pmtn$ (tâches prépemtives), $\bar{d}$ (présence des deadlines), r_j : présence de dates de début au plus tôt ; $D_7 : p_j = 1, d_j = d$.

Concernant les objectifs : f_{max} : fonction du coût maximum ; T_max : le retard absolu maximum ; $\sum U_j$ nombre de tâches en retard ; $\sum f_j$: somme des coûts des tâches ; $\sum w_jU_j$ nombre pondéré de tâches en retard ; $\sum w_jV_j$: somme pondérée des charges de tâches en retard ; $\sum E_j$: somme des avances des tâches ; $\sum w_jE_j$: somme des pénalités des avances des tâches ; $\sum co_jx_j^B$: somme des coûts des compressions : $\sum R_j$: somme des revenues des tâches ; $\sum Z$: somme des fonctions des coûts ; γ_0^A et γ_0^B $\in \{L_{max}, \sum T_j, \sum w_jT_j, \sum U_j, \sum w_jU_j\}$; γ_1^A et γ_1^B $\in \{L_{max}, \sum C_j, \sum U_j, \sum w_jU_j\}$; γ_2^A et γ_2^B $\in \{f_{max}, C_{max}, \sum C_j, \sum w_jU_j\}$; γ_3^A $\in \{f_{max}, \sum E_j, \sum w_jE_j\}$; γ_4^A et γ_4^B $\in \{f_{max}, \sum E_j, \sum w_jE_j, \sum w_jC_j\}$; γ_5^A et γ_5^B $\in \{\sum T_j, \sum w_jT_j, L_{max}, \sum U_j, \sum w_jU_j, \sum w_j(E_j +$

$T_j)\}$; γ_6^A et $\gamma_6^B \in \{\sum L_{max}, \sum C_j, \sum w_j C_j, \sum U_j, \sum w_j U_j\}$; $\gamma_7^A \in \{C_j, x_j, \sum(x_j + C_j), L_{max} + x_j, L_{max} + x_j\}$; γ_8^A et $\gamma_8^B \in \{C_{max}, L_{max}, f_{max}, \sum C_j, \sum w_j U_j\}$; γ_9^A et $\gamma_9^B \in \{f_{max}, \sum f_j, \sum U_j\}$; $\gamma_1 0^A$ et $\gamma_1 0^B \in \{\sum w_j C_j, C_{max}, L_{max}\}$; $\gamma_1 1^A$ et $\gamma_1 1^B \in \{\sum C_j, \sum U_j, \sum w_j C_j, L_{max}\}$.

Concernant les méthodes : AO : algorithme optimal, AP : algorithme approximative, H : heuristique, MH : métaheuristique, AGs : algorithmes génétiques, A-Pseudo : algorithme pseudo-polynomial, A-Poly : algorithme polynomial, FPTAS : $fully\ polynomial - time\ approximation\ scheme$, PSO : $particle\ swarm\ optimization$, PD-EA : $dynamic\ programming\ based\ exact\ algorithm$, P-PDPA : $pseudo - polynomial\ dynamic\ programming\ algorithm$, APC : $approximation\ Pareto\ curve$, RSH : $reserved - space\ heuristic$, DMH : $dynamic - mix\ heuristic$, LB : $lower\ bound$, MBO : $marriage\ in\ honey - bees\ optimization$, AC : $ant\ colonyalgorithm$.

6.2 Problèmes avec deux agents en compétition sur plusieurs machines

Dans ce qui suit, nous passons en revue les travaux portant sur l'ordonnancement à deux agents avec plus d'une machine. Contrairement aux travaux considérant deux agents et une seule machine, les travaux considérant le cas de plusieurs machines sont peu nombreux. On peut regrouper ces travaux selon le type de machines utilisées, à savoir, des machines identiques, des machines uniformes ou des machines non reliées (ou générales).

6.2.1 Travaux avec des machines parallèles identiques

Ce modèle d'ordonnancement considère $m \quad (m \geq 2)$ machines parallèles identiques et N tâches à exécutées. Les durées des tâches sont indépendantes des machines. Chaque tâches doit être exécutée sur une ou plusieurs machines. Chaque tâche doit être traitée sur une seule machine sans interruption ou avec interruption.

Balasubramanian et al Balasubramanian et al. (2009) ont été les premiers à considérer un environnement à machines parallèles identiques. Ils étudient le cas où le critère de l'un des agents est de minimiser le makespan C_{max} et celui de l'autre est de minimiser la somme des dates de fin de ses tâches $\sum C_j$. Les auteurs de Leung et al. (2010) étudient plusieurs problèmes avec des machines identiques dans lesquels les tâches sont préemptives et ont des dates de début au plus tôt différentes. Les critères considérés dans leur travail sont de types : la somme des dates de fin d'exécution $\sum C_j$, le nombre des tâches en retard $\sum U_j$ et une fonction régulière de type f_{max}. Chaque combinaison de fonctions objectifs conduit à un problème dont l'objectif est de minimiser la fonction objectif d'un agent, à condition que la valeur objective de l'autre agent ne dépasse pas un seuil donné. Wan et al. Wan et al. (2010) étudient un problème d'ordonnancement à deux agents où les agents partagent deux machines identiques. Les auteurs considèrent le cas des tâches préemptives et contrôlables. L'objectif du premier agent est de minimiser les dates de fin d'exécution de ces tâches plus le coût de compression, tandis que le second agent veut garder sa fonction de coût maximum f_{max} en dessous d'une valeur donnée. Les auteurs de Zhao and Lu (2013) étudient deux problèmes d'ordonnancement à deux agents en compétition sur des machines parallèles identiques où l'objectif est de minimiser soit le makespan, soit la somme des dates de fin d'un agent, sous contrainte d'une borne supérieure sur le makespan de l'autre agent. Lee et al Lee et al. (2016) s'intéressent également à un modèle similaire ayant comme objectif la minimisation de la somme pondérée des dates de fin d'un agent, avec une borne supérieure sur le makespan de l'autre agent. Les auteurs de Choi and Park (2017) considèrent un modèle similaire dans lequel le nombre de tâches exécutées simultanément du second agent est limité, bien que certaines machines soient disponibles. L'objectif est de minimiser la somme des dates de

Auteurs	Problème	Méthodes	Jeux de données	Autres
Li et al. (2021a)	$1 \mid\mid \sum w_j U_j^A, \sum w_j U_j^B \leq Q$	PSO, TS, PD-EA	Propre	
Li et al. (2021b)	$1 \mid\mid \sum R_j^A, \sum w_j U_j^B \leq Q$	MILP, H, PSO	Reisi-Nafchi and Moslehi (2015)	Rejet éventuel des tâches
Yin et al. (2021)		FPTAS, P-PDPA	Étude de la complexité	Setup times entre les batches, dates de fin à attribuer aux tâches
Zhang (2021)	$1 \mid\mid \sum w_j V_j^A, \sum C_j^B \leq Q$	A-Pseudo, APC, AP	Étude de la complexité	
Choi et al. (2020)	$1 \mid\mid \sum w_j^A + \sum co_j x_j^B$; $1 \mid\mid w_j^A, \sum co_j x_j^B \leq Q$	A-Poly	Étude de la complexité	Durées contrôlables
Chen and Li (2019)	$1 \mid\mid \gamma_2^A, \gamma_2^B \leq Q$	A-Poly, A-Pseudo	Étude de la complexité	Détériorations cumulatives
Liu et al. (2019)	$1 \mid r_i \mid C_{max}^A + \theta C_{max}^B$	FPTAS, AP	Propre	
Sahu et al. (2018)	$1 \mid\mid \sum w_j C_j^A, C_{max}^B \leq Q$	AO, H, PSO	Propre	Setup times entre les agents
Wang et al. (2017c)	$1 \mid D_4 \mid C_{max}^A, C_{max}^B \leq Q$	LB, RSH, DMH	Propre	
Wang et al. (2017a)	$1 \mid\mid \sum V_j^A, L_{max}^B \leq Q$	P-PDPA, BB, TS	Propre	
Wang et al. (2017b)	$1 \mid r_j \mid w_j C_j + \sum w_j C_j$	BB	Propre	
Zhang and Wang (2017)	$1 \mid\mid \sum w_j V_j^A, f_{max}^B \leq Q$	A-Poly, A-Pseudo	Étude de la complexité	
Li et al. (2016)	$1 \mid\mid \gamma_6^A, \gamma_6^B \leq Q$	A-Poly, A-Pseudo	Étude de la complexité	Setup times
Yin et al. (2016b)	$1 \mid D_3 \mid \gamma_1^A + \gamma_2^B \leq Q$	A-Poly,MILP, P-PDPA, FPTAS	Propre	Setup times entre les batches
Kovalyov et al. (2015)	$1 \mid D_1 \mid \gamma_8^A, \gamma_8^B \leq Q$	PDPA, P-PDPA	Étude de la complexité	Setup times
Lee et al. (2015)	$1 \mid\mid \sum C_j^A, T_{max}^B \leq Q$	BB, GA	Propre	Effet d'apprentissage
Wang et al. (2015)	$1 \mid\mid \gamma_5^A, \gamma_5^B \leq Q$	A-Poly, FPTAS	Étude de la complexité	Dates des fin à attribuer aux tâches
Yin et al. (2015)	$1 \mid\mid \gamma_0^A, \gamma_0^B \leq Q$	A-Poly	Étude de la complexité	Détériorations linéaires
Gawiejnowicz and Suwalski (2014)	$1 \mid r_j \mid \sum w_j C_j + \theta L_{max}$	MH,BB	Propre	Détériorations linéaires
Cheng et al. (2013)	$1 \mid r_i \mid \sum w_j C_j^A, L_{max}^B \leq Q$	BB, SA	Propre	
Fan et al. (2013)	$1 \mid D_5 \mid \sum C_j^A, C_{max}^B \leq Q$, $1 \mid D_5 \mid C_{max}^A, C_{max}^B \leq B$	A-Poly, FPTAS	Étude de la complexité	
Gerstl and Mosheiov (2013)	$1 \mid D_7 \mid \sum E_j + T_j, max(E_j + T_j)^B \leq Q$	Poly-A	JàT	
Liu et al. (2013)	$1 \mid\mid \sum C_j^A, C_{max}^B \leq Q$	A-Poly		Détériorations cumulatives
Wu et al. (2013b)	$1 \mid\mid \sum w_j U_j^A, L_{max}^B \leq Q$	BB, TS	Propre	Détériorations linéaires
Wu et al. (2013a)	$1 \mid r_j \mid \sum C_j^A, C_j^B \leq Q$	BB, AC, AGs	Propre	
Yin et al. (2013b)	$1 \mid D_2 \mid \sum Z$	PograD Poly	Propre	Intervalles d'exécution communs, coûts de livraison des batches
Yin et al. (2013a)	$1 \mid r_j \mid \sum U_j^A, L_{max}^B \leq Q$	BB, SA	Propre	
Yin et al. (2012c)	$1 \mid r_j \mid \sum T_j^A, L_{max}^B \leq Q$	MILP, BB, MBO	Propre	
Lee et al. (2012)	$1 \mid r_j \mid \sum T_j^A, T_{max}^B \leq Q$	BB, AGs	Propre	
Li and Yuan (2012)	$1 \mid D_1 \mid \gamma_9^A, \gamma_9^B \leq Q$	A-Poly et A-Pseudo	Étude de la complexité	Setup times
Yin et al. (2012a)	$1 \mid\mid \gamma^A + \theta\gamma^B$	A-Poly, A-Pseudo, P-PDPA	Propre	Dates des fin à attribuer aux tâches
Yin et al. (2012b)	$1 \mid\mid \gamma_3^A, f_{max}^B \leq Q$	A-Poly	Étude de la complexité	Détériorations linéaires non-croissantes
Liu et al. (2011)	$1 \mid\mid \sum C_j^A, f_{max}^B \leq Q$, $1 \mid\mid L_{max}^A, C_{max}^B \leq Q$	A-Poly	Étude de la complexité	Détériorations linéaires
C.E.Cheng et al. (2011)	$1 \mid\mid \sum w_j C_j^A, \sum U_j^B \leq 0$	BB, SA	Propre	Effet d'apprentissage
Mor and Mosheiov (2011)	$1 \mid p = 1 \mid \sum C_j^A, \sum C_j^B \leq Q$	A-Poly	Propre	Setup times identiques entre les batches
Wan et al. (2010)	$1 \mid D_6 \mid \gamma_7^A, f_{max}^B \leq Q$	A-Poly	Étude de la complexité	Durées contrôlables
Agnetis et al. (2009)	$1 \mid\mid \gamma_1 0^A, \gamma_1 0^B \leq Q$	BBs	Propre	
Cheng et al. (2006)	$1 \mid\mid \gamma_1 1^A, \gamma_1 1^B \leq Q$	A-Poly, A-Pseudo FPTAS	Étude de la complexité	
Ng et al. (2006)	$1 \mid\mid \sum C_j^A, \sum U_j^B, \leq Q$	A-Pseudo	Étude de la complexité	

Tab. 6.1 : Problèmes d'ordonnancement à deux agents en compétition sur une seule machine

fin du premier agent, sous condition que la somme pondérée des tâches traitées du second agent, sur chaque machine m_i, soit égal ou supérieur à une valeur λ_i. Zhao et Lu Zhao and Lu (2016) abordent également un modèle similaire où l'objectif est de minimiser le makespan du premier agent tout en maintenant le makespan de second agent sous une valeur donnée. Yu et al Yu et al. (2018) adressent un problème à deux agents qui partagent deux machines parallèles identiques. L'objectif est de minimiser simultanément la somme des retards du première agent et le makespan du second agent. Li and Lu Li and Lu (2017) étudient également un problème à deux agents sur deux machines parallèles identiques, sous l'hypothèse qu'une tâche peut être rejetée avec une pénalité encourue. La fonction objectif de chaque agent est la somme du coût d'ordonnancement plus la somme des coûts des pénalités de rejet. Les auteurs considèrent plusieurs combinaisons de fonctions, notamment le makespan, le retard maximal, la somme des dates de fin et le nombre pondéré de tâches en retard. Pei et al. Pei et al. (2020) considèrent un problème d'ordonnancement à deux agents en concurrence sur des machines identiques à traitement par batch en parallèle, caractérisé par des détérioration cumulatives et des capacités limitées des machines. L'objectif est de minimiser le makespan d'un agent avec une contrainte que le makespan de l'autre agent ne dépasse pas un seuil donné. Sadi and Soukhal (2017) étudient un problème d'ordonnancement des travaux concurrents sur machines parallèles identiques. Il s'agit d'un problème d'ordonnancement multi-agent interférant. De nouveaux résultats de complexité ont été élaborés lorsque les travaux sont de durées identiques. Certains problèmes sont montrés polynomiaux où des algorithmes de résolution exactes sont développés et d'autres sont montrés $\mathcal{NP}$-difficiles.

6.2.2 Travaux avec des machines parallèles uniformes

Dans ce modèle, chaque machine à sa propre vitesse qui est indépendante de la tâche et potentiellement différente de celles des autres machines. Cependant, la durée de chaque tâche varie en fonction de la vitesse de la machine sur laquelle cette tâche est traitée. Étant données que p_j est la durée de la tâche n_j et s_i la vitesse de la machine m_i, la durée de la tâche n_j sur la machine m_i est la suivante $p_{i,j} = p_j/s_i$.

Les auteurs de Elvikis et al. (2011) s'intéressent à l'énumération totale du front de Pareto strict pour un problème d'ordonnancement à deux agents en compétition sur des machines uniformes. Ils considèrent que toutes les tâches ont la même durée. L'objectif de l'agent A est de minimiser une fonction de coût non-décroissante, alors que c'est le makespan qui doit être minimisé pour l'agent B. Elvikis et T'kindt Elvikis and T'kindt (2014) considèrent le même modèle ou les deux fonctions objectifs sont de type f_{max}. Gerstl et Mosheiov Gerstl and Mosheiov (2013) considèrent un problème d'ordonnancement de type juste-à-temps (JàT) avec des tâches de durées unitaires. Ils étudient le cas où les machines sont identiques et aussi le cas où elles sont uniformes. L'objectif est de minimiser la somme pondérée des retards et des avances pour un agent, sous une contrainte d'une borne supérieure sur la déviation maximale (par rapport à une date de fin commune) des tâches de l'autre agent.

6.2.3 Travaux avec des machines parallèles non reliées

C'est le cas le plus général, dans lequel les performances des machines dépendent des tâches à exécuter. Les travaux intégrant une telle configuration dans un contexte à deux agents sont très rares.

Sadi et al. (2014) considèrent un environnement d'ordonnancement avec m ($m \geq 1$) machines parallèles. L'ensemble des travaux à ordonnancer est divisé en K sous-ensembles disjoints. Chaque sous-ensemble de travaux est associé à un agent. Les K agents sont en compétition pour exécuter leurs tâches sur des ressources communes. L'objectif est de trouver un ordonnancement qui minimise une fonction objectif globale f^0, tout en maintenant la fonction objectif régulière de chaque

agent f^k à une valeur fixe ε_k ($f^k \in \{f^k_{max}, \sum f^k\}, k = 0, \ldots, K$). Ce problème est appelé *problème d'ordonnancement multi-agent avec une fonction objectif globale*. Dans cet article, les auteurs étudient le cas où la préemption est autorisée et le cas où la préemption est interdite. Les tâches sont à exécuter sur machines parallèles générales. Si la préemption est autorisée, les auteurs montrent que le problème est polynomial. Si la préemption n'est pas autorisée, de nouveaux résultats de complexité sont élaborés et des approches de résolution basées sur la programmation dynamique sont élaborées. Yin et al. Yin et al. (2016a) considèrent un problème d'ordonnancement de type juste-à-temps. Le première agent désire maximiser le nombre pondéré de ses tâches qui sont achevées exactement à leurs dates de fin souhaitées (c'est-à-dire les tâches juste-à-temps), tandis que le second vise à maximiser soit le revenu maximum résultant de ses tâches terminées juste-à-temps, soit le nombre pondéré de ses tâches terminées juste-à-temps. Dans Yin et al. (2019), les auteurs traitent un problème d'ordonnancement à deux agents qui partagent des machines parallèles non reliées. L'objectif global est de minimiser la somme des dates de fin des tâches d'un agent, sous contrainte d'une borne supérieure sur le nombre pondéré des tâches en retard de l'autre agent.

6.2.4 Synthèse

Le Tableau 6.2 est une synthèse des travaux cités dans cette section. Les nouveaux acronymes et abréviations introduits dans ce tableau sont présentés ci-dessous.

Contraintes : o : le nombre de tâches exécutées simultanément d'un agent est limité. Objectifs : $\sum W_i$: la somme pondérée des tâches traitées sur la machine m_i, $\sum R$: nombre de tâches rejetées, $\gamma_1^B \in \{C_{max}, \sum L_{max}, \sum C_j, \sum U_j\}$; $\gamma_2^A \in \{\sum C_j, \sum U_j, f_{max}\}, \sum co_j x_j^B$: somme des coûts de compression ; $\sum w_j JT_j$: le nombre pondéré des tâches terminées JàT.

Méthodes : IP : *integer programming*, MCPEA : *minimal complete Pareto set enumeration algorithm*, CGS : *column generation scheme*, GH : *greedy heuristic*, BP : $branch - and - price$.

Les problèmes d'ordonnancement à deux agents étudiés dans cette partie sont des cas particuliers du problème $\mathcal{NP}$-difficile $Qm \mid r_j, pmtn \mid \sum w_j T_j^A, \sum w_j T_j^B$. En effet, nous introduisons pour la première fois des aspects multi-compétences dans un contexte d'ordonnancement multi-agent, ce qui représente une contribution importante dans ce domaine. On peut citer plusieurs caractéristiques distinguant notre travail de ceux existant dans la littérature. Notre modèle intègre le cas de ressources humaines multi-compétences avec différents niveaux de détails, par exemple, les types de compétences, le niveau de maîtrise des compétences et la disponibilité des ressources. Notre modèle tient également en compte des différents aspects liés aux tâches, telles que la préemption, les dates de début et de fin souhaitées et la charge qui pourra être réalisée de chaque tâche par semaine. De plus, deux types de contraintes sont considérées, notamment des contraintes dures et des contraintes molles.

Auteurs	Problèmes	Méthodes	Autres
Problème dans le chapitre 7	$Qm \mid pmtn, m-skill, r_j \mid \sum w_j T_j^A, \sum w_j T_j^A \geq Q,$ $Qm \mid pmtn, m-skill, r_j \mid \sum w_j T_j^A, \sum w_j T_j^A$	MILP, NSGA-II, TS	Contraintes dures et molles
hline Pei et al. (2020)	$Pm \mid p-batch \mid C_{max}^A, C_{max}^B \geq Q$	Détériorations cumulatives	
Yin et al. (2019)	$Rm \mid\mid \sum C_j^A, \sum w_j U_j C_j^B \geq Q$	MILP,CGS,BP	
Yu et al. (2018)	$P2 \mid\mid \sum T_j, C_{max}$	H	
Choi and Park (2017)	$Pm \mid p_j = 1, o, \mid \sum C_j^A, W_i \geq \lambda_i$	Poly-A, Pseudo-A,PD	
Li and Lu (2017)	$P2 \mid\mid \sum C_j + \sum R_A, \gamma_1^B + \sum R_B \geq Q$	Poly-As,Pseudo-A,FPTAS	Rejet des tâches
Lee et al. (2016)	$Pm \mid\mid \sum C_j^A, C_{max}^B \geq Q$	BB,GA	
Zhao and Lu (2016)	$Pm \mid\mid \sum C_{max}^A, C_{max}^B \geq Q$	APs	
Yin et al. (2016a)	$Rm \mid\mid \sum w_j JT_j^A, \sum w_j JT_j^B, \sum w_j JT_j^A, max(w_j JT_j^B)$	Poly-A,Pseudo-A,FPTAS	JàT
Elvikis and T'kindt (2014)	$Qm \mid p_j = p \mid f_{max}^A, f_{max}^B$	Poly-A,MCPEA	
Gerstl and Mosheiov (2013)	$Pm \mid D_7 \mid \sum E_j + T_j, max(E_j + T_j)^B \leq Q, Qm \mid D_7 \mid \sum E_j + T_j, max(E_j + T_j)^B \leq Q$	Poly-A	JàT
Zhao and Lu (2013)	$Pm \mid\mid C_{max}^A, C_{max}^B \leq Q ;$ $Q \mid\mid \sum C_j^A, C_{max}^B \leq Q$	FPTAS	
Elvikis et al. (2011)	$Q \mid p_j = p \mid F^A, C_{max}^B$	Poly-A	
Leung et al. (2010)	$Pm \mid pmtn, r_j \mid \gamma_2^A, f_{max}^B \leq Q$	Étude de la complexité	
Wan et al. (2010)	$P2 \mid pmtn, ctrl \mid \sum (C_j + co_j x_j^B), f_{max}^B \leq Q$	A-Poly	
Balasubramanian et al. (2009)	$P \mid\mid C_{max}^A, \sum C_j^B$	IP,H,GA	

Tab. 6.2 : Problèmes d'ordonnancement à deux agents en compétition sur plusieurs machines

Chapitre 7

Problèmes d'ordonnancement multi-agent sous contraintes de ressources multi-compétences

Contents

Comme vu dans le Chapitre 4, les quotités des employés sur les projets ont un grand impact sur la qualité des solutions obtenues. Les expériences menées ont montré que l'aménagement efficace des quotités permet de réduire considérablement le coût et le temps de développement des projets. Dans cette partie, nous étendons le problème présenté dans le Chapitre 4 pour inclure le cas où les quotités des employés sur les projets ne sont pas fixées préalablement par le décideur. Nous considérons que plusieurs chefs de projet, chacun gérant un ou plusieurs projets, sont en concurrence pour obtenir les ressources humaines nécessaires à l'exécution de leurs projets. L'affectation des employés pour l'ordonnancement des tâches doit optimiser une fonction objectif de chaque chef de projets. Il s'agit d'un problème d'ordonnancement multi-agent dans un contexte de gestion de projets.

Nous abordons deux problèmes d'ordonnancement à deux et trois agents, où les agents (chefs de projets) sont en compétition sur l'utilisation des ressources humaines multi-compétences nécessaires pour la réalisation de leurs tâches. Chaque agent est responsable de son propre sous-ensemble de tâches et cherche à optimiser une fonction objectif qui dépend uniquement des dates de fin d'exécution de ses tâches. D'abord, nous considérons le cas de deux agents en compétition. Puis, nous

étudions le cas non-disjoint, c'est-à-dire deux agents en compétition sur l'utilisation des ressources souhaitant chacun réduire les coûts des retards pondérés de ses propres tâches et un un troisième agent, dit agent global. L'agent global en charge de gérer ses ressources humaines tout en veillant sur leurs conditions de travail pour une meilleure efficacité. Ce dernier cas d'étude ajoute donc au deuxième problème le faite que certaines contraintes peuvent être violées pour diminuer le coût du retard lorsqu'il n'est pas possible de trouver une solution réalisable pour un horizon de planification H donné. Cet objectif global cherche à minimiser la violation de certaines contraintes dites souples afin de réduire les déviations indésirables par rapport aux objectifs des agents "locaux". Cette nouvelle mesure de performance considère la totalité des tâches à réaliser. Principalement, deux approches de résolution sont considérées : ε-contrainte et l'énumération du front de Pareto.

7.1 Présentation des problèmes

Dans ce chapitre, les problèmes d'ordonnancement à l'étude peuvent être définis comme suit. Soit $K = \{k_1, \ldots, k_L\}$ un ensemble de L projets devant être réalisés impérativement avant une date d'échéance (horizon) commune H. Soit A et B deux chefs de projets, chacun responsable d'un sous-ensemble disjoint de projets, i.e. chaque tâche appartient à un seul agent. Chaque projet k_l est découpé à un ensemble de tâches indépendantes et préemptives. On ne considère pas des contraintes de précédence ni entre les projets ni entre les tâches. Soit $N = \{n_1, \ldots, n_J\}$ l'ensemble des tâches à réaliser. Comme les tâches sont indépendantes, nous supposons que l'ensemble $N_A = \{n_1, \ldots, n_{J_A}\}$ (resp. $N_B = \{n_{J_A+1}, \ldots, n_J\}$) est le sous ensemble de tâches de l'agent A (resp. B), où $N = N_A \cup N_B$ et $N_A \cap N_B = \phi$. L'unité de temps considérée ici est la demi-journée.

Pour chaque tâche n_j, on dispose d'une charge nominale c_j (exprimée en hommes-jours, d'une date de début au plus tôt r_j (donnée en nombre de semaines) et d'une charge minimale notée $c_j^{\min}$ (exprimée en demi-journées) pour quantifier le degré minimal de réalisation de cette tâche par semaine. Dans le cas où l'employé affecté à la tâche n_j travaille sur cette tâche pendant une semaine donnée, il doit réaliser au moins sa charge minimale $c_j^{\min}$. Ceci permet de limiter la fragmentation de la tâche sur l'horizon temporel. Dans chaque semaine, l'employé affecté à la tâche n_j ne doit pas dépasser la charge maximale $c_j^{\max}$ de cette tâche. Un employé ne peut travailler que sur une seule tâche durant une demi-journée donnée. Nous voulons éviter la perte de temps due au changement de contexte des employés qui est nécessaire lors du passage d'une tâche à une autre. Ainsi, au cours de chaque semaine, le nombre de tâches différentes sur lesquelles un employé travaille est inférieur à une valeur donnée b, fixée pour tout le projet et tous les employés.

Soit $E = \{e_1, \ldots, e_I\}$ un ensemble de I employés multi-compétences travaillant dans l'entreprise. Chaque employé a une disponibilité par semaine (temps de travail) allant de 0 à 10 demi-journées. Cette disponibilité est connue au début de l'ordonnancement et n'est pas sujette au changement. On désigne par $D_{i,s}$ la disponibilité de l'employé e_i durant la semaine h_s, où $h_s \in H$. Un employé peut intervenir dans plusieurs projets en parallèle, avec un taux maximum de participation par projet à déterminer par le calcul de l'ordonnancement. Ainsi, au cours de chaque semaine h_s, chaque employé e_i affecté au projet k_l ne peut pas consacrer à ce projet plus de $D_{i,s} \times Q_{i,l}$, où $Q_{i,l}$ est la quotité maximale de l'employé e_i sur le projet k_l.

Dans notre modèle, une fois qu'une tâche est attribuée à un employé possédant la compétence requise, elle le reste pendant tout le temps de son exécution. Les capacités d'exécution des tâches par les ressources sont présentées par une matrice de compétences binaire désignée par m, où $m_{j,i} = 1$ si l'employé e_i maîtrise la tâche n_j, et $m_{j,i} = 0$ sinon. Cela pour dire que tous les employés ne peuvent pas être affectés à une tâche. De plus, nous supposons que plusieurs employés peuvent avoir des niveaux d'efficacité différents pour une même compétence. Comme chaque tâche ne requiert qu'une seule compétence, on associe directement le niveau de compétence de l'employé à la tâche. Le gestionnaire estime le niveau d'efficacité des employés selon la classification standard de niveau d'expertise : junior, moyen et senior. Sur la base de ces estimations, nous attribuons un

coefficient d'efficacité respectivement égal à 0, 0.5 et 1 à un junior, un moyen et un senior. Ainsi, pour prendre en compte le niveau d'efficacité de l'employé dans le calcul du temps de traitement de la tâche, on suppose une simple formule linéaire entre la charge nominale de la tâche et l'efficacité de l'employé qui y est affecté. Nous appliquons cette formule pour convertir la charge nominale (c_j) de la tâche n_j en durée ($p_{j,i}$) en fonction du niveau d'efficacité ($v_{j,i}$) de l'employé e_i (on suppose que l'employé e_i maîtrise la tâche n_j) : $p_{j,i} = (2 - v_{j,i})c_j$. Comme la charge nominale de chaque tâche est donnée en nombre de jours, nous la multiplions par 2 pour la convertir en demi-journées.

Par l'utilisation de la notation discutée dans la Section 5.4, les problèmes abordés dans cette partie du manuscrit sont décrits par :

1. Problèmes 1 ($Pb1$) : $Qm, m-skill \mid CO, r_j, pmtn \mid f^A, f^B$ et $Qm, m-skill \mid CO, r_j, pmtn \mid (f^A/f^B)$ dans le cas où l'approche ε-contrainte est appliquée. $m-skill$ désigne des ressources multi-compétences

2. Problème 2 ($Pb2$) : $Qm, m-skill \mid CO-GA, r_j, pmtn \mid f^G, f^A, f^B$. f^G est le critère global appliqué sur la totalité des travaux des deux agents et f^A (resp. f^B) est la fonction objectif de l'agent local A (resp B). Problème aussi appelé dans la littérature "ordonnancement interférant".

Le second problème $Pb2$ ajoute aux précédents $Pb1$ la possibilité de violer certaines contraintes. Ces contraintes, nommées ici contraintes souples, comprennent les contraintes des charges minimales des tâches et les contraintes de nombre maximum de tâches effectuées par employé par semaine. Le décideur autorise la violation de ces contraintes pour diminuer le retard ou lorsqu'il n'est pas possible de trouver une solution réalisable pour $Pb1$. Ainsi, un objectif global consistant à minimiser la violation de ces contraintes est ajouté pour réduire les déviations indésirables par rapport aux conditions de travail "idéales".

Contrairement aux méthodes de résolution introduites pour résoudre les problèmes étudiés dans les chapitres précédents, les quotités sont des variables de décision déterminées de façon exacte ou approchée. Pour les heuristiques et métaheuristiques, les solutions sont constituées principalement de deux parties : (i.) La première partie spécifie les quotités des employés aux projets, notamment, quels employés doivent réaliser quel(s) projet(s) et combien de temps au maximum ces employés doivent y consacrer ; (ii.) La deuxième partie spécifie, à partir de ces quotités, une affectation compatible des employés aux différentes tâches des projets.

L'ensemble des méthodes proposées nous permettent donc de calculer le front de Pareto exact et approché. Pour déterminer le front exact, l'approche ε-contrainte est utilisée.

Dans le tableau 7.2, nous présentons les nouvelles notations introduites dans ce chapitre. Les notations déjà utilisées dans le chapitre précédent restent valables et sont rappelées dans la Table 7.1.

Nous décrivons ci-dessous plus formellement les fonctions objectifs :

- $f^X = \sum_{j \in N_X} w_j T_j$, où $X \in \{A, B\}$, T_j est le nombre de semaines de retard de la tâche n_j et w_j le coût de pénalité de retard pour cette tâche. On note que, si une tâche n_j prend au moins une demi-journée de la semaine $(d_j + 1)$ avant sa date de fin, elle est en retard d'une semaine ($T_j = 1$).

- $f^G = \sum_{j=1}^{J} \sum_{s=r_j}^{H} \alpha u_{j,s}^{-} + \sum_{i=1}^{I} \sum_{s=1}^{H} \beta o_{i,s}^{+}$, où $u_{j,s}^{-}$ est la déviation inférieure à $c_j^{\min}$, $o_{i,s}^{+}$ est la déviation supérieure à b, et α et β sont des poids calculés sur la base des préférences du décideur sur les contraintes. Plus le poids d'une contrainte est élevé, plus sa violation est pénalisée.

	Données générales
H	Horizon de planification des projets
K	Ensemble de projets, $K = \{k_1, \ldots, k_L\}$, $\lvert K\rvert{=}L$
E	Ensemble des employés, $E = \{e_1, \ldots, e_i\}$, $\lvert E\rvert{=}I$
N	Ensemble de toutes les tâches, $N = \{n_1, \ldots, n_J\}$, $\lvert N\rvert{=}J$
N_l	Ensemble des tâches appartenant au projet k_l
	Données sur les tâches
c_j	Charge nominale de la tâche n_j (mesurée en jours-homme)
r_j	Date de début au plus tôt de la tâche n_j (donnée en nombre de semaines)
d_j	Date de fin souhaitée de la tâche n_j (donnée en nombre de semaines)
$c_j^{\max}$	Charge maximale de la tâche n_j par semaine (mesurée en demi-journées)
$c_j^{\min}$	Charge minimale de la tâche n_j par semaine (mesurée en demi-journées)
w_j	Coût de pénalité de la tâche n_j
F_j	Date de fin de la tâche n_j (exprimée en nombre de semaines)
T_j	Retard de la tâche n_j (exprimé en nombre de semaines)
	Données sur les employés
$D_{i,s}$	Disponibilité de l'employé e_i durant la semaine h_s (en demi-journées)
b	Nombre maximum de tâches sur lesquelles un employé peut travailler durant chaque semaine
	Données sur les tâches et les employés
$v_{j,i}$	Niveau d'efficacité de l'employé e_i pour la tâche n_j
$p_{j,i}$	Temps de traitement de la tâche n_j en fonction du niveau d'efficacité de l'employé e_i (en demi-journées)
$m_{j,i}$	1 si l'employé e_i maîtrise la tâche n_j ; 0 sinon
$\overline{v_j}$	Efficacité moyenne des employées pour réaliser la tâche n_j.

Tab. 7.1 : Notations introduites dans le Chapitre 4

	Paramètres
X	l'indice de l'agent, $X \in A, B$
N_X	Sous ensemble des tâches de l'agent X, $\lvert N_X\rvert = J_X$ et $J = J_A + J_B$
WL_l^g	Charge nominale du projet k_l en compétence z_g
$Z_g^l = \{z_g \lvert g \in \{1, .., G\}\}$	Ensemble de compétences nécessaires pour la réalisation du projet k_l
E_g	Ensemble de employés maîtrisant la compétence z_g
$M_{i,g}$	1 si l'employé e_i maîtrise la compétence z_g requise par le projet k_l ; 0 sinon

Tab. 7.2 : Nouvelles notations introduites dans ce chapitre

7.2 Méthodes exactes basées sur la programmation linéaire

Dans cette section, principalement deux modèles mathématiques sont développés : (i.) résolution du problème d'ordonnancement du projet multi-agent (en compétition ou non-disjoint) basée sur l'approche ε-contrainte ; (ii.) calculs des taux d'intervention par semaine de chaque employé sur chaque projet. Ce dernier modèle est utilisé dans les méthodes approchées afin de déterminer une allocation des employés aux projets.

7.2.1 Approche ε-contrainte

Dans cette section, nous présentons un modèle mathématique basé sur la programmation linéaire en nombres entiers (PLNE). Ce modèle est basé sur l'approche ε-contrainte pour calculer une solution de Pareto strict aux problèmes étudiés dans ce chapitre $Qm, m{-}skill \mid CO, r_j, pmtn, f^B \leq Q_B \mid f^A$ et $Qm, m - skill \mid CO - GA, r_j, pmtn, f^A \leq Q_A, f^B \leq Q_B \mid f^G$.

Pour déterminer une solution optimale au sens de Pareto, chaque instance est résolue deux fois. Par exemple, lorsque nous avons deux agents en compétition CO, nous procédons comme suit :

1. Résoudre par le PLNE le problème CO en minimisant f^A sous contrainte que $f^B \leq Q_B$, où Q_B est un paramètre de la méthode. La solution trouvée est optimale pour le critère de l'agent A, notée (f^{*A}, Q_B).

2. Résoudre une deuxième fois le problème en minimisant f^B sous contrainte que $f^A \leq f^{*A}$. La solution, notée (f^{*A}, f^{*B}) ainsi obtenue est optimale au sens de Pareto.

Pour déterminer l'ensemble du front optimal, on utilise le PLNE d'une manière itérative en

faisant varier la borne $Q_B = f^{*B} - \epsilon$, où ϵ est un paramètre de la méthode. La procédure se répète jusqu'à ce que Q_B soit en dessous de λ, où λ est le coût des retards de l'agent B après avoir exécuté l'ensemble des tâches de l'agent A. Une des approches utilisées dans ce cas pour estimer λ est l'approche lexicographique : minimiser le critère de l'agent A en ne considérant que ses travaux dans le système, puis par l'approche ε-contrainte, résoudre le problème $Minf^B$ sous contrainte $f^A \leq f^{*A}$.

Avant de présenter nos PLNE indexés-temps, nous allons décrire les variables de décision.

Variables de décision

- $x_{j,i}$: Variable binaire égale à 1 si l'employé e_i est affecté à la tâche n_j ; 0 sinon.
- $y_{j,i,s}$: Variable entière dans $\{0, 1, 2, \ldots, 10\}$ qui indique le nombre de demi-journées effectuées de la tâche n_j par l'employé e_i durant la semaine h_s.
- $z_{j,i,s}$: Variable binaire égale à 1 si $y_{j,i,s}$ est supérieur à 0 ; 0 sinon.
- F_j : Date de fin de la tâche n_j.
- T_j : Retard de la tâche n_j.
- $Q_{i,l}$: Quotité maximale de l'employé e_j sur le projet k_l (pourcentage du temps).

PLNE-$Pb1$

Fonction objectif

$$\text{Minimiser} \sum_{j=1}^{J_A} w_j T_j \tag{7.1}$$

Contraintes

$$\sum_{i=1}^{I} x_{j,i} \ = 1, \quad j = 1, \ldots, J \tag{7.2}$$

$$x_{j,i} \ \leq m_{j,i}, \quad j = 1, \ldots, J; \quad i = 1, \ldots, I \tag{7.3}$$

$$\sum_{s=r_j}^{H} y_{j,i,s} \ = p_{j,i} \cdot x_{j,i} \quad j = 1, \ldots, J; \quad i = 1, \ldots, I \tag{7.4}$$

$$\sum_{i=1}^{I} y_{j,i,s} \ \leq \sum_{i=1}^{I} c_j^{\max} \cdot x_{j,i}, \quad j = 1, \ldots, J; \quad s = r_j, \ldots, H \tag{7.5}$$

$$\sum_{i=1}^{I} z_{j,i,s} \ \leq \sum_{i=1}^{I} x_{j,i} \quad j = 1, \ldots, J; s = r_j, \ldots, H \tag{7.6}$$

$$\sum_{i=1}^{I} 10 \cdot z_{j,i,s} \geq \sum_{i=1}^{I} y_{j,i,s}, \quad j = 1, \ldots, J; \quad s = r_j, \ldots, H \tag{7.7}$$

$$\sum_{j=1}^{J} y_{j,i,s} \leq D_{i,s}, \quad i = 1, \ldots, I; \quad s = 1, \ldots, H \tag{7.8}$$

$$\sum_{j \in N_l} y_{j,i,s} \leq Q_{i,l} \cdot D_{i,s}, \quad i = 1, \ldots, I; \quad s = 1, \ldots, H; \quad l = 1, \ldots, L \tag{7.9}$$

$$\sum_{i=1}^{I} y_{j,i,s} \geq \sum_{i=1}^{I} c_j^{\min} * z_{j,i,s}, \quad j = 1, \ldots, J; \quad s = r_j, \ldots, H \tag{7.10}$$

$$\sum_{j=1}^{J} z_{j,i,s} \leq b, \quad i = 1, \ldots, I; \quad s = 1, \ldots, H \tag{7.11}$$

$$F_j \geq \sum_{i=1}^{I} s \cdot z_{j,i,s}, \quad j = 1, \ldots, J; \quad s = r_j, \ldots, H; \tag{7.12}$$

$$F_j - T_j \leq d_j, \quad F_j, T_j \geq 0 \tag{7.13}$$

$$\sum_{j=J_A+1}^{J} w_j T_j \leq Q_B \tag{7.14}$$

La fonction objectif 7.1 minimise la somme des retards pondérés des tâches de l'agent A. La contrainte 7.2 garantit qu'un seul employé doit être affecté à chaque tâche. La contrainte 7.3 assure qu'un employé ne doit pas travailler sur une tâche qu'il ne maîtrise pas. La contrainte 7.4 impose que si un employé est affecté à une tâche, il doit l'exécuter jusqu'à son intégralité. La contrainte 7.5 garantit qu'au maximum $c_j^{\max}$ à exécuter de la tâche n_j durant chaque semaine. Les contraintes 7.6 et 7.7 garantissent que chaque tâche est exécutée par le même employé tout au long de son exécution. La contrainte 7.7 permet de vérifier, pour chaque tâche, si l'employé qui y est affecté a travaillé sur cette tâche au cours d'une semaine donnée. La contrainte 7.8 assure que, pour une semaine donnée, aucun employé ne doit dépasser sa disponibilité. La contrainte 7.9 assure que, durant une semaine donnée, aucun employé ne doit dépasser sa quotité sur chaque projet. La contrainte 7.10 garantit que si un employé travaille sur une tâche pendant une semaine donnée, il doit effectuer de cette tâche au moins à la hauteur de sa charge minimale. La contrainte 7.11 garantit qu'au cours d'une semaine donnée, le nombre de tâches différentes sur lesquelles un employé travaille doit être inférieur ou égal au nombre maximum autorisé (b). Les contraintes 7.12 et 7.13 déclarent respectivement, pour chaque tâche, la date de fin et le retard. Enfin, la contrainte 7.14 borne supérieurement la valeur de l'objectif de l'agent B (approche ε-contrainte).

PLNE-*Pb*2

Pour résoudre à l'optimum le problème $Qm, m-skill \mid CO-GA, r_j, pmtn \mid (f^G/f^A, f^B)$, la modélisation du problème *Pb*2 suppose une légère modification du PLNE-*Pb*1 7.2.1 afin de tenir compte de l'objectif de l'agent global et de la possibilité de violer les contraintes souples. Ces violation sont pénalisées dans la fonction objectif finale comme suit :

- $u^+_{j,s}$: déviation supérieure à $c^{\min}_j$. associée à la tâche n_j et à la semaine h_s.
- $u^-_{j,s}$: déviation inférieure à $c^{\min}_j$. associée à la tâche n_j et à la semaine h_s.
- $o^+_{i,s}$: déviation supérieure à b associée à l'employé e_i et à la semaine h_s.
- $o^-_{i,s}$: déviation inférieure à b associée à l'employé e_i et à la semaine h_s.

Les contraintes 7.10 et 7.11 du PLNE-*Pb*1 sont remplacées par les contraintes 7.15 et 7.16 suivantes :

$$\sum_{i=1}^{I} y_{j,i,s} - u^+_{j,s} + u^-_{j,s} = \sum_{i=1}^{I} c^{\min}_j * z_{j,i,s}, \quad j = 1, \ldots, J; \quad s = r_j, \ldots, H \tag{7.15}$$

$$\sum_{j=1}^{J} z_{j,i,s} - o^+_{i,s} + o^-_{i,s} = b, \quad i = 1, \ldots, I; \quad s = 1, \ldots, H \tag{7.16}$$

Lorsque l'approche ε-contrainte est utilisée, pour le PLNE-*Pb*2 nous introduisons également la nouvelle contrainte 7.17 :

$$\sum_{j=1}^{J_A} w_j T_j \leq Q_A \tag{7.17}$$

La fonction objectif du problème *Pb*2 est alors présentée par la formule 7.18.

$$\text{Minimiser} \sum_{i=1}^{I} \sum_{s=r_i}^{H} \alpha u^-_{i,s} + \sum_{j=1}^{J} \sum_{s=1}^{H} \beta o^+_{j,s} \tag{7.18}$$

7.2.2 Génération des quotités exactes

Dans cette sous-section, nous développons le PLNE noté PGQ_{exact}, qui permet de déterminer les quotités d'intervention de chaque employé aux projets. Ce PLNE a pour but d'affecter à chaque projet les employés possédant les compétences nécessaires pour sa réalisation. Il doit également préciser le temps de travail maximum que chaque employé consacre à chacun de ses projets.

Il s'agit d'un PLNE à indexation temporaire qui considère uniquement les contraintes liées aux disponibilités hebdomadaires des employés, ainsi que les charges des projets à réaliser, et qui peut être décrit comme suit.

On introduit des variables entières $Y_{i,l,s,g} \in \{0, 1, 2, \ldots, 10\}$. Cette variable binaire indique le temps effectif que l'employé e_i doit consacré au projet k_l durant la semaine h_s en travaillant sur la compétence z_g. La variable entière $Q_{i,l} \in \{0, 1, 2, \ldots, 10\}$ indique la quotité maximale de l'employé e_i affectée au projet k_l. La variable entière $d^-_{i,s}$ (entre 0 et $D_{i,s}$) qui est égale à la disponibilité non utilisée de l'employé e_i durant la semaine h_s.

PGQ_{exact} :

$$\text{Minimiser} \sum_{i=1}^{I} \sum_{s=1}^{H} d_{i,s}^{-} \tag{7.19}$$

s.c.

$$Y_{i,l,s,g} \leq 10 \cdot M_{i,g} \quad i = 1, \ldots, I; \quad s = 1, \ldots, H; \quad g = 1, \ldots, G; \quad = l, \ldots, L; \tag{7.20}$$

$$\sum_{g=1}^{G} Y_{i,l,s,g} \leq D_{i,s} \cdot Q_{i,l}, \quad i = 1, \ldots, I; \quad s = 1, \ldots, H; \quad l = 1, \ldots, L \tag{7.21}$$

$$\sum_{l=1}^{L} \sum_{g=1}^{G} Y_{i,l,s,g} + d_{i,s}^{-} = D_{i,s}, \quad i = 1, \ldots, I; \quad s = 1, \ldots, H \tag{7.22}$$

$$\sum_{i=1}^{I} \sum_{s=1}^{H} Y_{i,l,s,g} \; = WL_{l}^{g}, \quad l = 1, \ldots, L \tag{7.23}$$

La fonction objectif (7.19) évite le fait que les employés travaillent, durant chaque semaine, moins que leur disponibilité. Elle minimise la somme des disponibilités non employées des employés (temps de travail inactif). La contrainte (7.20) garantit qu'un employé ne doit pas travailler sur un projet dont il ne maîtrise aucune des compétences requises par ce projet. La contrainte (7.21) garantit qu'un employé doit respecter la charge allouée à un projet selon la quotité fixée et sa disponibilité. La contrainte (7.22) permet de respecter la disponibilité par semaine de chaque employé. La contrainte 7.23 impose que la charge nominale de chaque projet en chaque compétence va être réalisée entièrement.

7.3 Méthodes de résolution approchées

Comme les problèmes à résoudre sont $\mathcal{NP}$-difficiles au sens fort, cette section présente deux schémas de résolution approchées pour déterminer des solutions non dominées des problèmes $Pb1$ et $Pb2$: approche globale et approche itérative. Les deux schémas font appel à un algorithme génétique de type NSGA-II. La différence entre les deux schémas réside dans la partie dédiée aux calculs des quotités et affectations des employés aux tâches/projets.

- Une approche globale : le calcul des quotités et l'affectation des employés aux projets n'est pas remis en cause, et ils sont déterminés au fur et à mesure de la résolution du problème.
- Une approche itérative : on introduit un modèle itératif en deux phases. Dans la première phase, on calcule les quotités nécessaires des employés aux projets. Dans la deuxième phase, on résout le modèle. S'il existe des projets en retard, on ajuste les quotités et on résout de nouveau le modèle, et ainsi de suite.

Dans ce qui suit, nous détaillons les approches de résolution proposées.

7.3.1 Approche globale

Cette approche, désignée par AGL, se base sur une procédure exacte et une heuristique hybride pour résoudre de manière approchée les problèmes étudiés. La procédure exacte utilise le PLNE PGQ_{exact} pour générer les quotités initiales des employés et leur affectation aux projets. Ensuite, à partir de ces quotités, un algorithme glouton est appliqué pour déterminer des solutions initiales. Les solutions retournées par l'algorithme glouton sont ensuite améliorées par la recherche tabou proposée pour la version mono-objectif du problème du Chapitre 4 (cf. Section 4.4.3). Via des règles de priorité différentes, nous générons un premier ensemble de solutions qui constituera alors la première partie de la population initiale de l'algorithme NSGA-II permettant de déterminer un front de Pareto approché.

La Figure 7.1 présente le schéma général de AGL dont les principales étapes sont les suivantes :

Étape 1 : Calculer les quotités des employés aux projets en utilisant la procédure exacte PGQ_{exact} de type PLNE.

Étape 2 : Déterminer les solutions initiales, en utilisant l'algorithme glouton AG en modifiant la liste des priorités des tâches (choix de la tâche suivante à affecter et choix de l'employé).

Étape 3 : Appliquer l'algorithme recherche tabou pour améliorer les solutions initiales.

Étape 4 : À partir de la population initiale, appliquer l'algorithme NSGA-II pour calculer des bonnes solutions de compromis (front de Pareto approché).

Dans ce qui suit, nous détaillons l'approche AGL à partir de l'étape 2.

7.3.1.1 Algorithme glouton

Cet algorithme représente une adaptation de l'algorithme glouton à trois-phases introduit dans la Section 4.3 et conçu pour résoudre le problème étudié dans le Chapitre 4. Nous utilisons des règles de priorité ainsi que des heuristiques simples pour construire rapidement et efficacement des solutions initiales de bonnes qualité. Cette étape est très importante car une affectation appropriée peut aider l'algorithme à obtenir rapidement de bonnes solutions, tandis qu'une affectation inappropriée peut rendre difficile la recherche des solutions faisables.

Cet algorithme glouton renvoie toutes les solutions générées après un délai limité AG_{max}. Chaque solution initiale est générée en deux étapes. La première étape consiste à déterminer un ordre dans lequel les tâches seront sélectionnées et la deuxième étape consiste à affecter les tâches aux employés. Le choix de l'employé pour chaque tâche est conditionné par la charge de travail qui lui a déjà été attribuée. Une charge de travail équilibrée entre les employés augmente les chances de faisabilité de la solution. Nous détaillons ci-dessous les deux étapes de l'algorithme glouton.

1. La première étape consiste à retourner la liste des tâches ordonnées selon l'ordre croissant de nombre d'employés maîtrisant chaque tâche. Par exemple, une tâche maîtrisée par un seul employé doit être attribuée avant une autre tâche maîtrisée par deux employés. Cela permet de s'assurer que les employés les plus indispensables ne sont pas surchargés par d'autres tâches qui ont plusieurs options d'affectation. Lorsque deux tâches sont maîtrisées par le même nombre d'employés, dans ce cas, la première tâche à affecter est choisie selon la règle de priorité WLPT.

2. La deuxième étape consiste à affecter les employés aux tâches des projets. Pour réaliser cette étape, nous procédons comme suit. Premièrement, pour chaque tâche n_j prise dans l'ordre de la liste des tâches, les employés maîtrisant n_j sont sélectionnés et ordonnés selon l'ordre décroissant de leurs disponibilité sur le projet correspondant. Cela signifie que l'employé le

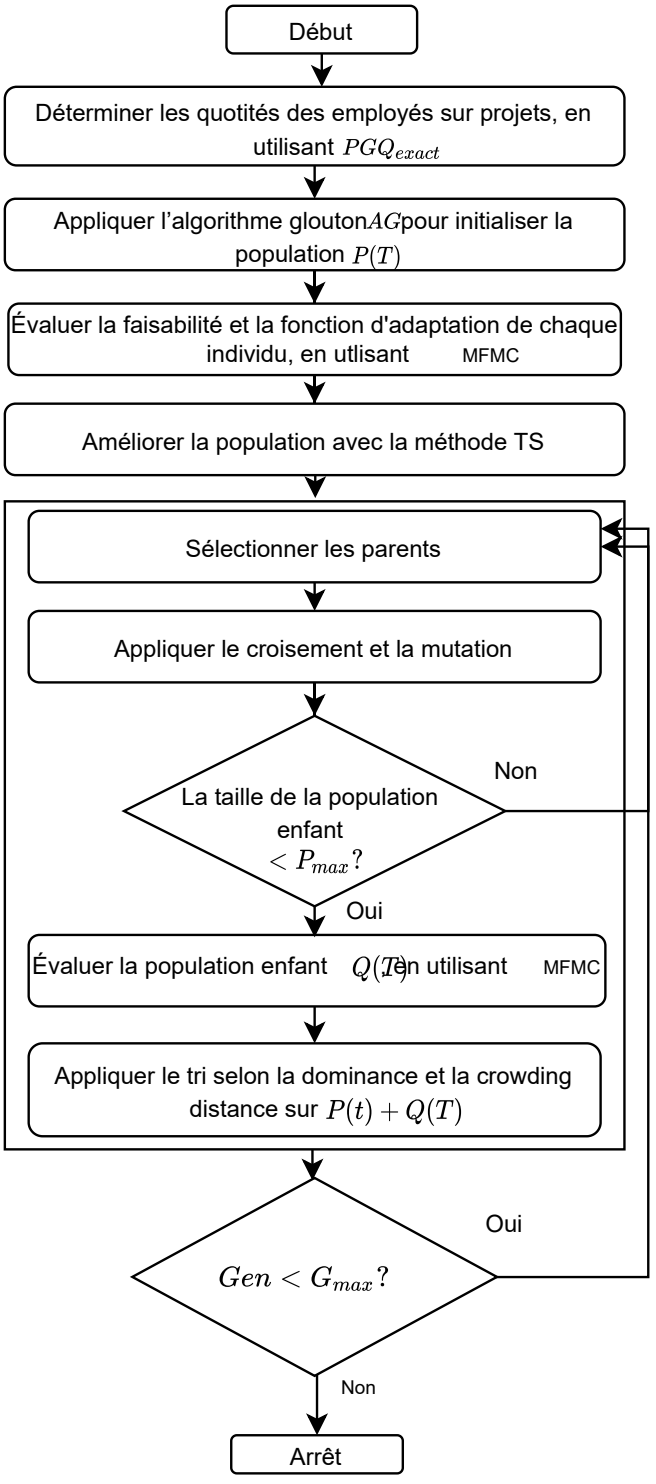


Fig. 7.1 : Schéma de l'approche globale AGL.

Algorithm 5 : Pseudo-code de l'algorithme glouton.

Entrée : N liste des tâches rangées selon l'ordre croissant du nombre d'employés maîtrisant chaque tâche ; τ^{tot} : la disponibilité des employés sur les projets
Sortie : σ l'affectation des employés aux tâches
pour chaque tâche $n_j \in N$ **faire**
    k_l : Projet auquel la tâche n_j appartient
    E_j : Liste des employés maîtrisant la tâche n_j
    $BinomiaList = \phi$
    pour chaque employé $e_i \in E_j$ **faire**
        Ajouter $(e_i, \tau^{tot}_{i,l})$ à $BinomiaList$
    fin pour
    Trier $BinomiaList$ dans l'ordre décroissant de τ^{tot}
    $NoAffecte =$ vrai, $cout = 0$
    tant que $NoAffecte$ **faire**
        si $BinomiaList[cout][1] - p_{j,i} \geq 0$ **alors**
            Affecter l'employé $BinomiaList[cout][0]$ à la tâche n_j
            $NoAffecte =$ faux
        fin si
        $cout++$
    fin tant que
    Mettre à jour la disponibilité de l'employé sélectionné
    Mettre à jour la liste d'affectation σ
fin pour
retourner σ

plus disponible sera affecté à n_j. On note qu'avant de sélectionner cet employé on soustrait à sa disponibilité $p_{j,i}$ le temps de traitement de la tâche. Ceci afin d'assurer qu'il dispose suffisamment de disponibilité pour accomplir la tâche. Autrement, si l'employé n'a pas assez de disponibilité, le deuxième employé de la liste sera sélectionné et ainsi de suite. Enfin, nous mettons à jour la disponibilité de l'employé sélectionné.

La première et la deuxième étapes génèrent une seule solution. Les solutions restantes sont générées itérativement en effectuant de simples opérations de mutation sur la liste des priorités des tâches, puis en répétant la deuxième étape. L'algorithme 5 présente plus formellement la procédure de génération des solutions initiales.

7.3.1.2 NSGA-II : adaptation aux problèmes à résoudre

La procédure générale d'une méthode NSGA-II est déjà présentée dans la Section 5.7. Nous rappelons ici les éléments généraux de cette méthode qui sont à définir. (i) le schéma de codage pour représenter un individu ou une solution, (ii) les opérateurs génétiques pour générer et modifier de nouveaux individus, (iii) la fonction d'adaptation (ou fitness function en anglais) pour mesurer et comparer les qualités des individus, (iv) les paramètres de l'algorithme (c'est -à-dire taille de la population P_{max}, le nombre des points de croisement P_c et de mutation P_m, critère d'arrêt G_{max}). Notre implémentation de cet algorithme applique ces éléments comme suit.

Représentation de l'individu

Le codage de l'individu est une tâche fondamentale et importante lors de l'application d'un algorithme évolutionnaire. Un schéma de codage approprié de la solution peut aider l'algorithme à obtenir facilement des solutions de haute qualité, le cas échéant, il peut dégrader considérablement

la performance de l'algorithme.

Un chromosome (ou un individu ou une solution) contient un nombre déterminé de gènes, et est divisé en un ou plusieurs segments. Dans notre cas : un individu est représenté par une affectation des employés aux tâches ; un gène correspond à une tâche ; un segment correspond à l'ensemble des tâches affectées à un employé ; et la longueur du segment correspond au nombre de tâches dans ce segment. Ainsi, on utilise dans le schéma de codage d'un individu un séparateur (0) pour indiquer le début et la fin de chaque segment.

Afin de comprendre le schéma de codage, considérons le cas où chaque agent est chargé de l'exécution d'un seul projet. Trois employés (e_1, e_2, e_3) peuvent être affectés aux différentes tâches des projets. Les projets sont constitués de 8 tâches (n_1–n_8) à planifier sur un horizon de deux semaines. Chaque agent veut minimiser la somme pondérée des retards de ses tâches. Le Tableau 7.3 indique pour chaque tâche, l'employée ayant la compétence requise pour la réaliser. Les colonnes grises représentent les tâches de l'agent B, tandis que les autres colonnes représentent les tâches de l'agent A.

	n_1	n_2	n_3	n_4	n_5	n_6	n_7	n_8
c_j	3	5	2	6	4	4	5	3
$m_{j,1}$	1	0	0	1	1	0	0	1
$m_{j,2}$	1	1	1	1	0	0	1	1
$m_{j,3}$	0	0	1	0	1	1	1	1

Tab. 7.3 : Affectations possibles des employés aux tâches.

La Figure 7.2 montre un exemple de codage d'un individu qui correspond à une affectation compatible des employés aux tâches. Cet exemple indique que les tâches $\{n_4, n_5\}$ sont affectées à l'employé e_1, $\{n_1, n_2, n_7\}$ sont affectées à l'employé e_2 et $\{n_3, n_6, n_8\}$ sont affectées à l'employé e_3. Le chiffre 0 est un séparateur utilisé pour séparer deux affectations.

n_4	n_5	0	n_1	n_2	n_7	0	n_3	n_6	n_8

Fig. 7.2 : Représentation d'individu.

Population initiale

La population initiale est construite en appliquant l'algorithme glouton décrit dans la Section 7.3.1.1. On note que la taille des solutions retournées par cet algorithme peut être inférieure à la taille maximale de la population P_{max} (ou P_{max} est un paramètre de l'algorithme NSGA-II). Dans ce cas, des opérations de mutation sont appliquées sur les individus élitistes pour compléter la population initiale.

Opérateurs génétiques Les opérateurs génétiques se composent de plusieurs éléments clés : sélection, croisement, mutation, tri de la population et fonction d'adaptation.

- Sélection : cet opérateur choisit parmi les individus de la population parent, ceux qui vont se reproduire et génèrent la population enfant (offspring) pour l'itération suivante. Pour ce faire, nous utilisons la méthode du tournoi binaire qui est l'un des opérateurs de sélection les plus courants. Cette méthode choisit au hasard un couple de parents parmi la population et compare leurs fonctions d'adaptation. Si les deux parents sont dans le même front, alors on compare leurs crowding distances, et celui avec la plus grande crowding distance est conservé. Sinon, si les deux parents sont dans des fronts différents, la solution ayant le meilleur front de Pareto (c'est-à-dire celle avec la meilleure valeur de la fonction objectif) est conservée.
- Croisement (ou crossover) : une fois la section est effectuée, tous les parents choisis, passent à l'étape de la reproduction. C'est là que tous les parents se recombinent d'une manière

ou d'une autre pour créer une nouvelle population qui sera utilisée dans la prochaine étape génétique (la mutation). Le processus de combinaison de deux parents est ce qu'on appelle souvent le croisement. Le croisement consiste à générer un ou plusieurs nouveaux individus en effectuant un croisement entre deux parents. L'individu résultant hérite partiellement des caractéristiques de ses parents. Nous utilisons les deux opérateurs de croisement PBX (position based crossover) et OBX (order based crossover) proposés dans Syswerda (1991). Le choix de ces opérateurs de croisement est effectué sur la base de l'étude réalisée par Kellegöz et al. (2008) qui compare les performances de 11 opérateurs de croisement. Leur travail considère un problème de minimisation des retards pondérés des tâches sur une seule machine. Leurs résultats expérimentaux montrent l'efficacité des opérateurs de croisement OBX et PBX par rapport aux autres opérateurs pour résoudre ce type de problèmes d'ordonnancement. Nous détaillons ci-dessous les différentes étapes de ces deux opérateurs. De plus, deux exemples correspondants sont présentés dans les Figures 7.3 et 7.4.

- Opérateur de croisement OBX
  - Sélectionner aléatoirement plusieurs gènes (tâches) à partir d'un parent ($P1$ par exemple).
  - Placer les tâches sélectionnées dans le nouvel individu, en respectant leurs positions exactes qu'ils occupent dans l'autre parent ($P2$).
  - Supprimer les tâches qui sont déjà choisies dans l'autre parent ($P2$) afin d'éviter de répéter ces tâches dans le nouvel individu ($O1$).
  - Insérer les tâches restantes dans le nouvel enfant ($O1$), de gauche à droite, dans l'ordre de leur apparition dans le parent (P2).
  - Placer les tâches restantes dans le nouvel enfant ($O1$), de gauche à droite, dans l'ordre de leur apparition dans le parent (P2).
  - En changeant les rôles des parents, la même procédure peut être appliquée pour générer le deuxième enfant ($O2$).
- Opérateur de croisement PBX
  - Sélectionner aléatoirement un ensemble de tâches à partir d'un parent ($P1$ par exemple).
  - Placer les tâches sélectionnées dans le nouvel enfant $O1$, en respectant leurs positions d'origines dans $P1$.
  - Supprimer les tâches qui sont déjà sélectionnées dans le deuxième parent $P2$. La séquence des tâches restantes dans $P2$ contient uniquement les tâches dont le nouvel enfant $O1$ a besoin.
  - Placer les tâches restantes dans $O1$, de gauche à droite, dans l'ordre dans lequel ils apparaissent dans le parent $P2$.
  - En changeant les rôles des parents, la même procédure peut être appliquée pour générer le deuxième enfant ($O2$).

La similarité entre la population enfant et la population parent dépend du nombre de points de croisement. Ce nombre P_c représente le nombre des tâches sélectionnées pour les injecter dans l'individu enfant. Une grande valeur de P_c donne une plus grande probabilité de créer des individus similaires à leurs parents et une petite valeur de P_c permet de sélectionner des solutions éloignées.

- Mutation : cet opérateur est appliqué lors de la dernière étape de la génération de la population pour l'itération suivante. Il est utilisé afin de maintenir une diversité entre les individus, et par conséquent, aider à éviter une convergence prématurée vers un optimum local. Elle est appliquée à chaque individu enfant après qu'elle ait été créée lors de l'étape de croisement. La mutation consiste à choisir aléatoirement deux employés (e_1 et e_2, par exemple)

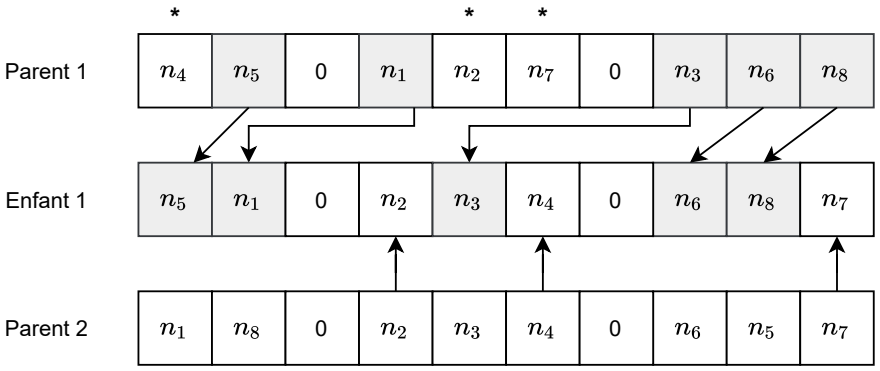


Fig. 7.3 : Illustration de l'opérateur de croisement OBX.

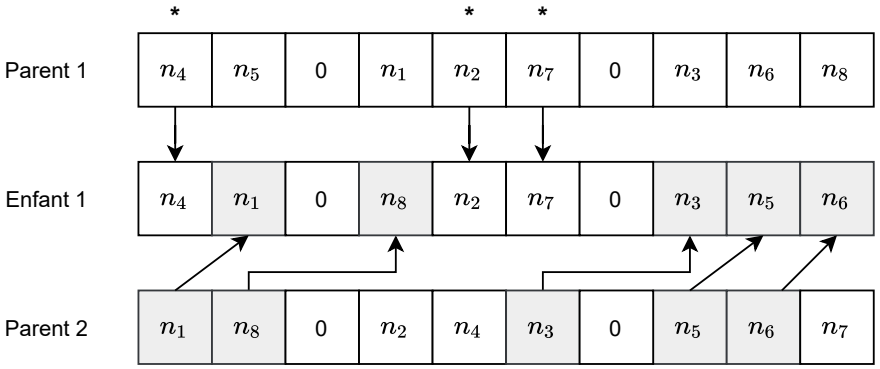


Fig. 7.4 : Illustration de l'opérateur de croisement PBX.

qui ont au moins une seule compétence en commun. Ensuite, toutes les tâches initialement attribuées à e_1 qui demandent cette compétence seront attribuées à e_2. L'inverse s'applique pour les tâches attribuées à e_2. Cette opération est effectuée P_m fois, avec P_m un nombre positif compris entre 2 et 10.

– Tri de la population : cette étape suit la même procédure de tri selon la dominance proposée par Deb et al. (2002). La crowding distance est également appliquée.

– Fitness d'adaptation : la valeur de la fitness de chaque individu repose sur la valeur de la fonction objectif du problème considéré $Pb1$ ou $Pb2$. Les fonctions objectifs considérées sont comme suit.

  – Problème $Pb1$: la fonction objectif minimise simultanément la somme des retards pondérés des deux agents : $f^A = \sum_{j=1}^{J_A} w_j T_j$ et $f^B = \sum_{j=J_A+1}^{J} w_j T_j$.
  – Problème $Pb2$: la fonction objectif minimise à la fois les déviations indésirables des contraintes souples par rapport à leurs objectifs et les deux objectifs des agents : $f^G = \sum_{j=1}^{J} \sum_{s=r_j}^{H} \alpha u_{j,s}^{-} + \sum_{i=1}^{I} \sum_{s=1}^{H} \beta o_{i,s}^{+}$, $f^A = \sum_{j=1}^{J_A} w_j T_j$ et $f^B = \sum_{j=J_A+1}^{J} w_j T_j$.

Une affectation est ainsi évaluée en calculant le flot maximum à coût minimum, telle introduite dans la Section 4.3.2 du Chapitre 4.

7.3.2 Approche itérative

Cette approche (désignée par AIT) correspond à un modèle qui résout d'une manière itérative les problèmes étudiés. Comme mentionné précédemment, cette approche opère en deux phases. Dans la première phase, on cherche à déterminer les quotités initiales des employés aux projets. Pour ce faire, nous appliquons une procédure heuristique de génération des quotités (PGQ_{heur}). Dans la deuxième phase, à partir de ces quotités, on cherche à résoudre l'affectation et l'ordonnancement des tâches par l'algorithme glouton et l'algorithme génétique précédemment détaillés. Au cours du processus de l'algorithme génétique et après un nombre donnée d'itérations PAQ_{max}, une autre procédure désignée par PAQ_{heur}, est appliquée pour ajuster les quotités afin de réduire les retards.

Figure 7.5 montre le schéma général du processus de AIT. Nous récapitulons ci-dessous les principales étapes de ce processus.

Étape 1 : Calculer les quotités initiales des employés aux projets en utilisant la procédure PGQ_{heur}.

Étape 2 : Déterminer une population initiale en utilisant l'algorithme glouton AG.

Étape 3 : Appliquer l'algorithme de recherche tabou pour améliorer la population initiale.

Étape 4 : À partir de cette population, appliquer l'algorithme NSGA-II pour calculer un front de Pareto approché.

Étape 5 : Après chaque PAQ_{max} itérations de l'algorithme NSGA-II, et s'il existe des projets en retard dans la population, appliquer la procédure PAQ_{heur} pour ajuster ces quotités.

Dans les sections suivantes, nous décrivons la procédure de génération et d'ajustement des quotités.

7.3.2.1 Procédures de génération et d'ajustement des quotités

(a) Procédure de génération des quotités (PGQ_{heur})

L'objectif de cette procédure est de déterminer efficacement une allocation initiale des employés aux charges nominales des projets. Sachant qu'un employé peut intervenir dans plusieurs projets, cette procédure doit fixer pour chaque employé un taux d'intervention maximum dans chacun de ces projets. L'ensemble du processus de cette procédure est décrit plus formellement dans l'Algorithme 6. Les besoins en compétences pour chaque projet k_l sont exprimées en charge nominale WL_l^g et données dans la liste Z_g^l. Pour chaque compétence $z_g \in Z_g^l$, on sélectionne la liste des employés E_g maîtrisant cette compétence. Ensuite, la charge nominale WL_l^g de la compétence z_g est repartie uniformément entre des employés sélectionnés aléatoirement de la liste E_g.

(a) Procédure d'ajustement des quotités (PAQ_{heur})

Cette procédure est utilisée lorsque des solutions avec retards ont été trouvées pour certains individus de la population. L'idée est de mieux répartir les quotités non utilisées "le surplus" en les affectant aux projets avec des retards important et ainsi pour les diminuer.

Afin d'obtenir le résultat décrit ci-dessus, nous avons construit deux tableaux. Le premier contient les quotités prévues, c'est-à-dire celles utilisées pour la méthode NSGA-II au début, et le second contient les quotités réelles de la solution qui présente le plus de retard. Ce que nous avons appelé quotités réelles sont les quotités déterminées par la résolution du problème de flot maximum à coût minimum permettant d'évaluer la solution. En effet, après avoir exécuté l'algorithme permettant de calculer le flot maximum à coût minimum sur le graphe, le flot qui passe sur les arcs situés entre les nœuds "Projet, semaine" et "Semaine" donne la quotité réelle utilisée par chaque employé sur chaque projet. Ainsi, en regardant ces deux tableaux, on peut voir où la quotité prévue est strictement supérieure à la quotité réelle utilisée. Le processus de cette procédure est décrit dans Algorithme 7.

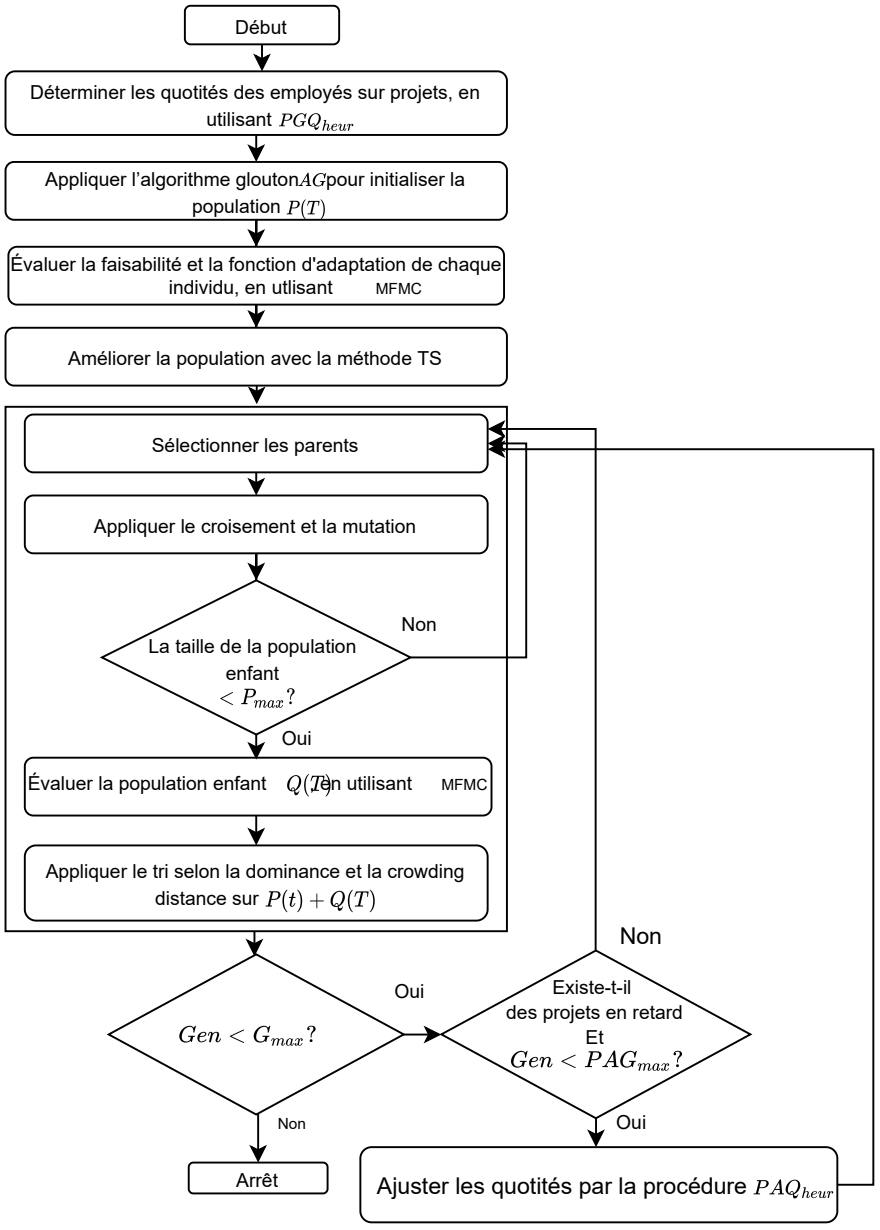


Fig. 7.5 : Schéma de l'approche itérative AIT.

Algorithm 6 : Pseudo-code de la procédure de génération des quotités initiales (PGQ_{heur})

Entrée : K : Liste des projets ; D^{total} : Les disponibilités totales des employés durant l'horizon de planification
Sortie : Q^{prev} : Quotités prévues des employés sur les projets
pour chaque Projet $k_l \in K$ **faire**
    pour chaque Compétence $z_g \in Z_g^l$ **faire**
        tant que $WL_l^g > 0$ **faire**
            Sélectionner un employé aléatoirement de E_g.
            si $WL_l^g < D_i^{total}$ **alors**
                Affecter e_i à WL_l^g
            sinon
                Choisir aléatoirement α_a entre 40% et 60%
                Affecter e_i à α_a de WL_l^g
                Mettre à jour WL_l^g
            fin si
            Mettre à jour D_i^{total}
            Mettre à jour Q^{prev}
        fin tant que
    fin pour
fin pour
retourne Q

Algorithm 7 : Pseudo-code de la procédure d'ajustement des quotités (PAQ_{heur})

Entrée : Kr : Liste des projets en retard ; Q^{prev} : Quotités prévues des employés sur les projets ; Q^{rel} : Quotités réelles des employés sur les projets
Sortie : Q^{prev} : Quotités prévues modifiées
$ListTrinome = \phi$
pour chaque Employé $e_i \in E$ **faire**
    pour chaque Projet $k_l \in K$ **faire**
        si $Q^{rel} < Q^{prev}$ **alors**
            Ajouter le trinôme à $ListTrinome$
        fin si
    fin pour
fin pour
pour chaque trinôme dans $ListTrinome$ **faire**
    Alimenter $ListProEnRetard$ de l'employé
    pour chaque Projet en retard dans $ListProEnRetard$ **faire**
        Constituer le trinôme correspondant
        Ajouter la quotité inutilisée au trinôme constitué
        Retirer la quotité inutilisée du trinôme en cours
        Mettre à jour Q^{prev}
    fin pour
fin pour
retourne Q^{prev}

7.4 Résultats expérimentaux

Dans cette section, nous présentons les caractéristiques des instances et les résultats expérimentaux de différentes approches de résolution proposées. Les expérimentations menées consistent en trois parties. La première partie est consacrée à la résolution du problème $Pb1$ (deux agents en compétition) et ainsi au calcul du front de Pareto exact et approché. La deuxième partie quant à elle, est dédiée à l'analyse des performances des méthodes de résolution dédiées aux problème $Pb2$. Et enfin, la troisième partie met l'accent sur l'apport de l'introduction de l'agent global sur la réduction des pénalités de retard de chaque agent.

Les expérimentations ont été menées sur un Intel® core™ i7-1.9 GHz avec 16 Go de RAM sous Windows 10. Tous les algorithmes ont été écrits en Python. Le solveur CPLEX 12.8.0 associé à l'API Python a été utilisé pour résoudre les modèles GPNE et PLNE, en utilisant les paramètres du solveur par défaut à l'exception du paramètre temps.

7.4.1 Jeux de données pour les problèmes $Pb1$ et $Pb2$

Les méthodes proposées dans cette section sont validées expérimentalement sur des instances dérivées des situations réelles (voir la Section 4.5.3.1). Pour réaliser nos expérimentations, nous avons généré artificiellement 8 ensembles d'instances (T1–T8) avec 40 instances par ensemble. Chaque ensemble d'instances a les mêmes caractéristiques que les instances de taille réelle avec un nombre différent d'employés et de tâches. Toutes les compétences requises par les projets peuvent être exercées par les employés de l'entreprise.

Le Tableau 7.4 présente les caractéristiques de certains paramètres généraux par ensemble d'instances. De gauche à droite, les colonnes indiquent le type d'instance, l'horizon de planification des projets, le nombre de projets, le nombre d'employés et la dernière colonne correspond au nombre de tâches. Pour chaque groupe d'instances, le nombre de tâches par agent est choisi entre $\{0,4 \times J\,;\, 0,5 \times J\,;\, 0,6 \times J\}$, avec J est le nombre de tâches. Les plages des paramètres restants sont sélectionnées de la même manière que dans les instances réelles T_{real} (voir Tableau 4.15).

Ensemble d'instances	H	L	I	J
T1	4	2	3	15
T2	4	2	3	20
T3	6	2	5	25
T4	8	2	10	50
T5	10	4	15	100
T6	14	4	20	150
T7	18	4	25	200
T8	24	4	30	250

Tab. 7.4 : Valeurs des paramètres généraux par ensemble d'instances

7.4.2 Cas de deux agents en compétition ($Pb1$)

Dans cette section, nous allons d'abord présenter les performances de l'approche exacte pour calculer le front de Pareto optimal, puis analyser les performances des heuristiques après avoir fixé les paramètres et opérateurs des approches proposées.

7.4.2.1 Performances expérimentales de PLNE-$Pb1$

Cette section présente les résultats expérimentaux permettant de trouver l'intégralité du front optimal de Pareto exact pour le problème $Qm \mid CO, m-skill, r_j, pmtn \mid f^A, f^B$. Cela permet aussi de déterminer la taille des instances du $Pb1$ qui peuvent être résolues par le modèle PLNE-$Pb1$ en temps raisonnable, lorsque l'approche ε-contrainte est utilisée (cf Section 7.2.1).

D'après les premières expériences, l'utilisation des solutions de départ (warm start) pour résoudre le programme linéaire en nombres entiers semble améliorer considérablement les résultats obtenus. Pour cela, nous avons utilisé l'algorithme glouton pour déterminer des solutions à partir desquelles le PLNE-$Pb1$ est initialisé.

L'un des inconvénients de la méthode ε-contrainte est qu'il y a autant de calculs à faire que de valeurs de Q_B. En outre, il peut arriver que plusieurs valeurs différentes de Q_B donnent des solutions redondantes ou des solutions faiblement dominées. Par exemple, le cas d'un front de Pareto composé des points suivants : $\{(\alpha, \beta_0), (\alpha, \beta_1), (\alpha, \beta_2,...,(\alpha, \beta_n)\}$, avec $\beta_1 < \beta_2,..., \beta_n$. Parmi toutes les solutions de ce front, la procédure va garder uniquement la solution (α, β_n), car elle domine faiblement les autres. Il est claire alors que la performance de la méthode ε-contrainte dépend fortement des valeurs du paramètre Q_B. Dans notre cas, la procédure commence par une valeur de Q_B générée en résolvant le problème par l'approche lexicographique. D'abord, on résout le problème avec seulement les tâches de l'agent A considérées. Par la suite, selon l'approche ε-contrainte, nous générons le premier point extrême en tenant compte de la valeur optimale de l'objectif de l'agent A avec objectif la minimisation du critère de l'agent B. Puis, à chaque itération, la valeur de Q_B est diminuée de manière itérative afin d'obtenir tous les points constituant le front de Pareto exact (cf. Section 7.2.1). Dans ce cas, la procédure utilise le paramètre ϵ dont la valeur est égale à la valeur de la pénalité minimale des poids des tâches de l'agent B : $\epsilon = min_{i \in J_B}(w_i)$.

Le Tableau 7.5 présente les résultats obtenus sur les différents ensembles d'instances. Dans ce tableau, de gauche à droite, les colonnes font référence au type d'instances, au nombre moyen de solutions de Pareto faible, nombre moyen de solutions de Pareto strict et au temps de calcul moyen du PLNE en secondes.

Ensemble d'instances	Nombre moyen de solutions de Pareto faible	Nombre moyen de solutions de Pareto strict	Temps de calcul moyen du PLNE (s)
T1	6.54	5.87	23.47
T2	6.24	5.56	33.91
T3	5.98	5.32	41.42
T4	5.52	4.12	292.43
T5	5.14	4.83	342.43
T6	-	-	-
T7	-	-	-
T8	-	-	-

Tab. 7.5 : Front de Pareto exact : problème $Pb1$

On constate que PLNE-$Pb1$ est performant pour résoudre les instances de moyenne taille (de moins de 100 tâches). En effet, pour ces instances, le modèle arrive à trouver chaque solution au bout de 6 minutes. De même, le temps moyen pour calculer chaque front de Pareto optimal est inférieur à 30 minutes. Par contre, le modèle semble avoir des difficultés à trouver les fronts de Pareto exacts pour les instances de plus de 100 tâches dans la limite du temps fixé, c'est le cas des

ensembles $T6, T7$ et $T8$. Pour l'ensemble des instances de ces trois derniers ensembles, nos fronts de Pareto de référence seront ceux trouvés par le solveur dans les temps impartis, nommés "MFP" pour Meilleurs Fronts de Pareto.

7.4.2.2 Paramétrage des méthodes approchées et choix des opérateurs : cas $Pb1$

Ayant recours aux algorithmes de type NSGA-II cela nécessite une phase de paramétrage et choix des opérateurs importants. Ainsi, une première campagne de tests est menée pour décider expérimentalement du choix des différents paramètres. Ensuite, une deuxième campane de tests est réalisée pour analyser l'efficacité de NSGA-II selon l'opérateur de croisement choisi (OBX ou PBX, cf. Figures 7.3 et 7.4). Une troisième campagne de tests est effectuée pour savoir l'impact de l'application de la recherche tabou (TS) introduite dans la Section 4.4.3 sur les solutions obtenues par la méthode NSGA-II.

Pour évaluer la qualité des solutions retournées, nous avons utilisé deux indicateurs de performance décrits dans la Section 5.5. Ces indicateurs sont la distance générationnelle (DG) et la taille du front de Pareto (TFP). Dans cette partie de nos expérimentations, nous avons utilisé 10 instances de chaque ensemble de nos jeux de données. Notons que, pour obtenir les fronts de Pareto exacts pour les instance $T1$–$T5$, nous avons exécuté le modèle PLNE-$Pb1$ sans limite de temps (jusqu'à ce que le front de Pareto soit obtenu). Cependant, la méthode exacte n'a pas pu retourner les fronts de Pareto exacts pour les instances sur les ensembles $T6$–$T8$. De ce fait, nous avons récupéré les meilleurs front de Pareto "MFP" après 2h/instance sur de temps de calcul de PLNE-$Pb1$ (le solveur a obtenu une solution faisable pour chacune des 10 instances de chaque groupe).

Les quotités des employés sur les projets sont initialement calculées par le PLNE PGQ_{exact}. Ces valeurs n'ont donc pas été modifiées afin de conserver la cohérence des tests, mais il a été estimé que leur modification n'aurait pas conduit à des conclusions différentes quant aux performances des méthodes.

Paramétrage de NSGA-II

Les paramètres considérés sont la taille de la population, le nombre d'itérations, le nombre de points de croisement et le nombre de points de mutation. Sur chaque instance, l'algorithme NSGA-II a été exécuté 5 fois avec chaque combinaison des paramètres. Les meilleurs valeurs obtenues sont présentées dans le tableau 7.6. Pour chaque exécution, l'algorithme génétique a été lancé à partir des mêmes solutions initiales générées par l'heuristique glouton HG afin de garantir la cohérence des tests. Pour fixer le nombre de points de croisement, nous avons effectué les tests avec les deux opérateurs de croisement. On a constaté d'une manière rassurante que les meilleures valeurs obtenues sont souvent les mêmes avec les deux opérateurs.

Paramètre	Plage	Valeur
Taille de la population, P_{max}	[100,300]	200
Nombre d'itérations, G_{max}	[200,3000]	1000
Nombre de points de croisement, P_c	[1,10]	8
Nombre de points de mutation, P_m	[1,10]	7

Tab. 7.6 : Paramétrage expérimentale de NSGA-II

Choix d'opérateur de croisement

En utilisant les valeurs de paramètres présentées dans le Tableau 7.6, nous avons exécuté l'algorithme NSGA-II 5 fois en utilisant les opérateur de croisement OBX et PBX. Pour chaque exécution, NSGA-II avec chaque opérateur de croisement a été lancé à partir des mêmes solutions

initiales générées par l'heuristique gloutonne HG. Figure 7.6 résume les résultats obtenus avec chacun des opérateurs. Pour les ensembles d'instances $T1$–$T5$, les distances générationnelles moyennes sont calculées entre les fronts de Pareto exacts et les fronts de Pareto approchés. Pour les restes des ensembles ($T6$–$T8$), les distances générationnelles moyennes sont calculées entre les fronts de Pareto MFP et les fronts de Pareto MFP. .

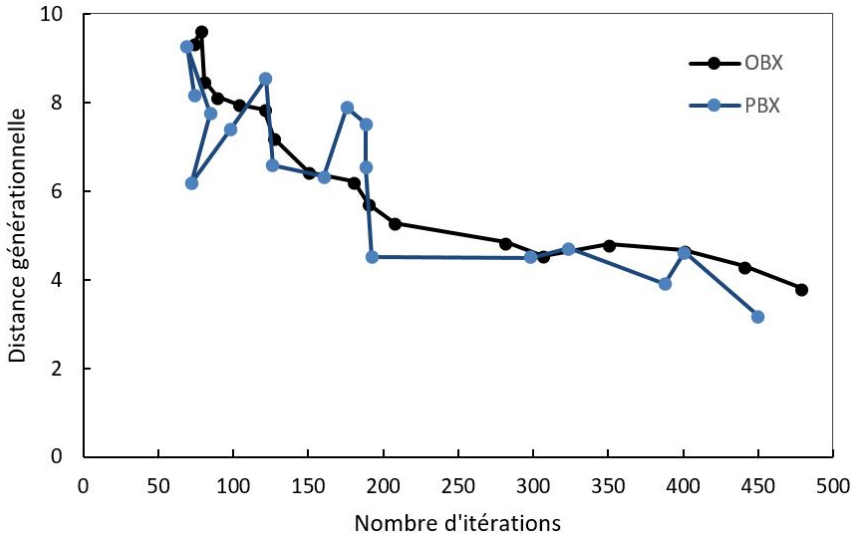


Fig. 7.6 : Comparaison des opérateurs de croisement

Les résultats illustrés par la Figure 7.6 montrent que pour résoudre le problème d'ordonnancement multi-agent, l'opérateur OBX a une meilleure performance par rapport à l'autre opérateur PBX. En fait, NSGA-II a une convergence plus stable et plus rapide avec l'opérateur génétique OBX. De plus, on a constaté d'une manière rassurante que les temps de calcul pour les deux opérations de croisement sont presque les mêmes pour chaque taille de problème. Nous retenons donc l'opérateur de croisement OBX.

Impact de la méthode recherche tabou sur la qualité du front approché

Comme mentionné précédemment, la méthode recherche tabou peut être appliquée sur les populations pour améliorer la qualité des solution retournées.

Des expériences ont été menées pour identifier la meilleure fréquence (nombre de fois) d'appels à cette méthode afin d'améliorer les solutions calculées par NSGA-II. L'objectif est de garder un bon ratio entre la qualité du front de Pareto et le temps de calcul. NSGA-II a été exécuté avec les meilleures valeurs des paramètres obtenues et le meilleur opérateur de croisement déduit.

Figure 7.7 montre les distances générationnelles calculées entre les fronts de Pareto exacts et MFP et les fronts de Pareto approchés, respectivement. Dans cette figure, nous avons reporté 4 expériences réalisées avec différentes fréquences d'appels de la méthode TS (voir tableau 7.7). Nous avons également utilisé les mêmes 10 instances précédent de chaque ensemble. La courbe noire montre la distance générationnelle pour le front de Pareto obtenu en appliquant uniquement le NSGA-II. La courbe rouge montre la distance générationnelle pour le front de Pareto obtenu en appliquant la méthode TS uniquement sur la population initiale. La courbe bleu (resp. verte) montre la distance générationnelle pour le front de Pareto obtenu en appliquant la méthode TS chaque 10 (resp. 5) générations. Nous avons utilisé les meilleures valeurs présentées dans le tableau 7.6 pour paramétrer notre méthode NSGA-II.

Expérience	Fréquence d'appel de TS
Exp 1	Sans le TS
Exp 2	1 (sur la population initiale)
Exp 3	10
Exp 4	5

Tab. 7.7 : Paramétrage de la fréquence d'appels de la méthode TS depuis NSGA-II

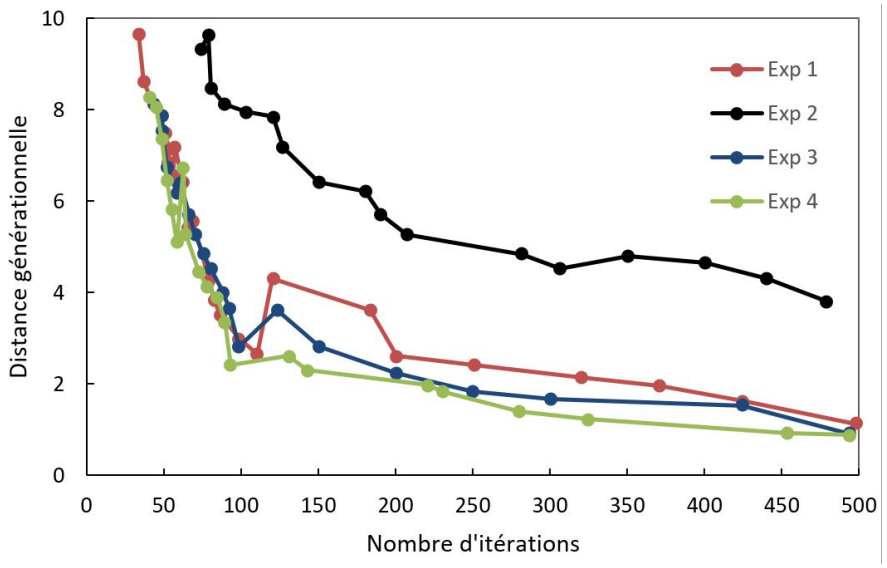


Fig. 7.7 : Impact de la méthode TS sur le front de Pareto optimal obtenu par NSGA-II

À partir des résultats montrés dans la Figure 7.7, on peut observer clairement que la méthode TS a un effet très intéressant sur la performance du NSGA-II. En appliquant la méthode TS à la population initiale, la distance générationnelle diminue d'une manière significative. De plus, un plus grande nombre de solutions non-dominées ont été trouvées et beaucoup de solutions retournées se trouvent sur le front de Pareto optimal avec une distribution uniforme. D'autre part, nous pouvons observer que le fait d'appeler la méthode TS toutes les 5 ou les 10 itérations, apporte peu d'amélioration sur la qualité du front approché, si on le compare au temps de calcul qui augmente. Sur la base de ces résultats, nous avons décidé d'appliquer la méthode TS une seule fois et sur la population initiale.

7.4.2.3 Analyse des performances de NSGA-II vs heuristique gloutonne

Un des objectifs maintenant est d'évaluer à quel point NSGA-II améliore les solutions obtenues par l'heuristique gloutonne. Tableaux 7.8 et 7.9 résument les résultats expérimentaux des méthodes HG, NSGA-II et la méthode exacte (en limitant le temps de calcul de PLNE-$Pb1$ à 30 minutes par instance). Le premier tableau, nous donne les déviations du front de Pareto obtenu par chacune des ces méthodes approchées par rapport aux fronts de Pareto exacts, tandis que le deuxième tableau résume les déviations des fronts de Pareto obtenus pas ces méthodes par rapport aux meilleurs fronts de Pareto trouvés (c'est-a-dire par rapport au MFP). Rappelons qu'un temps de calcul supérieur à 10 min ne répond pas aux exigences de l'entreprise. Nous avons utilisé les meilleures

valeurs obtenues pour le paramétrage de l'algorithme génétique (cf Tableau 7.6). Les solutions retournées par l'algorithme glouton sont récupérées après un délai HG_{max} limité à 5 min. Nous avons exécuté les deux algorithmes 10 fois et les valeurs moyennes sont calculées et rapportées. Pour chaque exécution de NSGA-II, nous avons utilisé la même population initiale générée par HG afin d'assurer la cohérence des tests. De même, les quotités des employés sur les projets sont identiques est fixées pour les deux approches de résolution et correspondent aux valeurs retournées par le PLNE PGQ_{exact}.

Pour comparer entre les résultats de ces trois méthodes, nous avons utilisé quatre indicateurs de performances différents. Ces indicateurs sont la taille moyenne du front de Pareto (TFP), l'hypervolume (HV) moyenne, la distances générationnelle (DG) moyenne, et le temps de calcul moyen en seconds (dans la colonne CPU).

Ensemble d'instances	PLNE-$Pb1$		NSGA-II				HG		
	TFP^*	$CPU*(s)$	TFP	HV	DG	CPU(s)	TFP	HV	DG
T1	5.42	746.22	4.82	0.82	3.51	177.12	2.81	0.47	8.91
T2	5.91	961.86	4.13	0.76	4.43	182.72	3.54	0.43	9.32
T3	5.52	1212.98	3.89	0.73	3.87	202.98	2.95	0.38	8.12
T4	4.23	1663.78	3.63	0.68	4.43	256.10	2.42	0.35	7.25
T5	4.85	17892.21	3.51	0.64	5.14	301.45	2.26	0.28	7.89

Tab. 7.8 : Évaluation de performance de NSGA-II vs Front de Pareto optimal

Ensemble d'instances	PLNE-$Pb1$			NSGA-II				HG		
	HV	TFP	$CPU(s)$	TFP	HV	DG	CPU(s)	TFP	HV	DG
T6	0.58	2.25	69762.81	4.51	0.62	5.14	189.95	3.42	0.18	6.91
T7	0.52	2.19	88190.11	3.41	0.68	5.14	212.34	3.11	0.11	6.89
T8	0	-	-	1.12	0.59	5.14	271.73	0	0.18	7.12

Tab. 7.9 : Évaluation de performance de NSGA-II vs MFP retourné par PLNE-$Pb1$

Les points qui peuvent être attirés de l'analyse des résultats présentés dans le Tableau 7.8 (par rapport aux front exacts) et le Tableau 7.9 (par rapport aux front MFP) sont les suivants. On peut constater que la taille des fronts de Pareto (TFP) générés par le NSGA-II sont très proches de la taille des fronts de Pareto stricts générés par la méthode exacte. Prenons par exemple les instances $T5$. La taille moyenne du front de Pareto optimal est égale à 4.85 et celle du front de Pareto généré par le NSGA-II est égale à 3.51. En comparant les deux distances générationnelles obtenues par les méthodes NSGA-II et HG, respectivement, nous pouvons remarquer que l'algorithme NSGA-II a considérablement réduit la DG obtenue par la méthode HG. Pour les instances $T5$, par exemple, la distance générationnelle du front générés par le HG est égale à 7.89 contre une valeur égale à 5.15 obtenue par le NSGA-II. Les hypervolumes dominés par les fronts de Pareto approchés est plus grand que ceux dominés par les fronts de Pareto obtenus par la méthode exacte (après 30 minutes de calcul fixé sur le PLNE-$Pb1$) et le HG. À partir du tableau 7.9, on peut constater que les résultats de l'algorithme NSGA-II surclassent les résultats de la méthode exacte (après 30/instance minutes de calcul du PLNE-$Pb1$) ainsi que ceux de la méthode HG. Pour les instance $T7$, par exemple, les hypervolumes dominés sont de valeur moyenne égale à 0.52 pour la méthode exacte, 0.11 pour le HG, et 0.68 en moyenne. Pour les instances $T8$ la méthode exacte n'arrive pas à retourner une solution faisable dans le temps de calcul fixé, les distances générationnelles sont de valeur moyenne égale à 7.12 pour le HG, et 5.14 pour le NSGA-II. Pour tous les ensembles d'instances, le temps de calcul de NSGA-II est très acceptable pour une heuristique (inférieur à 5 minutes). À travers ces résultats, nous pouvons conclure clairement que l'algorithme NSGA-II

améliore significativement les solutions obtenues par l'heuristique gloutonne.

Dans la suite et pour la résolution du problème $Pb1$, nous allons mettre l'accent sur les performances des approches de résolutions proposées lorsque les quotités ne sont pas connues par avance. Rappelons que l'approche de résolution globale notée AGL s'appuie sur le modèle mathématique PGQ_{exact} pour déterminer les quotités d'intervention de chaque employé sur les projets, quant à la méthode itérative, notée AIT, les quotités sont ajustées au fur et à mesure de l'exécution.

7.4.2.4 Analyse des performances de AGL vs AIT pour le problème $Pb1$

Cette section rapporte les résultats expérimentaux de l'approche de résolution globale AGL et de l'approche de résolution itérative AIT pour résoudre le problème $Qm \mid CO, m-skill, r_j, pmtn \mid f^A, f^B$. Trois différentes campagnes de tests sont réalisées. La première campagne vise à analyser la performance de AGL, en évaluant la qualité du modèle PGQ_{exact} développé. La deuxième campagne vise à équilibrer la fréquence d'appel de la procédure PAQ_{heur} lors du processus AIT. Enfin, dans la troisième campagne, nous comparons la qualité des solutions retournées par les deux approches.

Résultats PGQ_{exact}

Cette section présente les résultats expérimentaux du modèle mathématique PGQ_{exact} (cf Section 7.2.2). Comme indiqué précédemment, ce modèle est utilisé par le AGL dans le but de déterminer des affectations (quotités) initiales des employés aux projets. Cette phase de résolution est primordiale car en partant de quotients efficaces et bien pensées, on augmente considérablement les chances de trouver une bonne solution.

PGQ_{exact} a été testé sur tous les ensembles d'instances de nos jeux de données. Le Tableau 7.10 montre les résultats obtenus après un temps de calcul limité à 5 min par instance. Dans ce tableau, de gauche à droite, les colonnes font référence au type d'instance, au nombre d'instances réalisables, au nombre de solutions qui sont prouvées comme étant les solutions optimales, au temps moyen nécessaire pour prouver l'optimalité, et aux déviations moyennes par rapport à l'optimalité pour les instances qui ne sont pas résolues de manière optimale.

Ensemble d'instances	Nombre d'instances réalisables	Nombre d'instances résolues jusqu'à l'optimalité	Temps moyen pour atteindre l'optimalité (s)	Déviation moyen par rapport a l'optimal
T1	40	40	72.8	0%
T2	40	40	122.2	0%
T3	40	40	150.7	0
T4	40	37	194.6	3.3%
T5	40	35	206.5	4.1%
T6	40	31	223.2	5.3%
T7	40	25	243.8	6.4%
T8	40	22	296.7	7.2%

Tab. 7.10 : Résultats du PGQ_{exact} après un temps de calcul limité à 5 min

Les résultats présentés dans le Tableau 7.10 montre la performance du modèle PGQ_{exact} en termes de faisabilité et d'optimalité. Le modèle arrive à trouver une solution réalisable pour chaque instance. De plus, toutes les solutions obtenues pour les instances de type $T1$–$T3$ sont des solutions optimales. Le modèle n'arrive pas à retourner une solution optimale pour 3/40, 5/40 et 9/40 instances des ensembles $T4$, $T5$ et $T6$, respectivement. Pour les instances $T7$ et $T8$, un grand nombre d'instances (15, et 18) ne sont pas résolues de manière optimale. Cependant, l'écart relatif en moyenne par rapport à l'optimum est très acceptable ($6,4\%$ pour l'ensemble $T7$ et $7,2\%$ pour $T8$).

Résultats de la procédure d'ajustement des quotités

Comme mentionné ci-dessus, le but de cette nouvelle campagne de tests est de fixer une fréquence efficace pour appeler la procédure d'ajustement des quotités. Dans nos expériences, nous avons exécuté AIT 10 fois sur les mêmes 5 instances représentatives de chaque ensemble d'instances. L'algorithme génétique a été lancé avec les meilleures valeurs des paramètres (voir Tableau 7.6), ainsi qu'avec l'opérateur de croisement OBX.

En faisant varier le paramètre PAQ_{max}, différents résultats expérimentaux sont obtenus. Par chaque exécution de la procédure AIT, nous avons exécuté le AIT avec les mêmes solutions initiales déterminées par PGQ_{heur} et une nouvelle valeur de PAQ_{max}.

Dans le Tableau 7.11, nous présentons les résultats obtenus. Les colonnes de ce tableau font référence à la fréquence d'appel de cette procédure dans le AIT, à l'hypervolume moyen (HV), à la distance générationnelle (DG) et au temps de calcul moyen (CPU).

Fréquence d'appel de PAQ_{heur}	HV	DG	CPU(s)
5	0.73	5.22	308.56
10	0.71	5.86	268.61
20	0.63	6.12	245.31
30	0.58	6.78	217.53
40	0.51	7.2	196.56
Sans le PAQ_{heur}	0.47	8.94	270.31

Tab. 7.11 : Paramétrage de la fréquence d'appel de PAQ_{heur}

D'après les résultats du tableau 7.11, nous pouvons conclure qu'en augmentant la fréquence d'appel de PAQ_{max} les solutions sont considérablement améliorées, et ceci se confirme quelque soit la métrique de performance utilisée. Ainsi, les tests effectués montrent aussi que le temps d'exécution est modérément augmenté. Afin de garder un bon ratio qualité/temps, nous avons décidé de fixer le paramètre PAQ_{max} à 10 itérations.

Résultats expérimentaux des deux approches de résolution

Cette partie des expériences conclut le travail effectué pour résoudre le problème $Qm \mid CO, m-skill, r_j, pmtn \mid f^A, f^B$. Son objectif est de comparer les différentes approches de résolution du problème, en utilisant les trois métriques de performance (c'est-à-dire HV, DG, et TFP). Nous avons exécuté chacune des deux approches 10 fois et les valeurs moyennes sont calculées et présentées dans le Tableau 7.12. Les tests sont menés sur toutes les instances de nos ensembles de jeux de données.

Ensemble d'instances	AGL				AIT			
	TFP	HV	DG	CPU(s)	TFP	HV	DG	CPU(s)
T1	5.1	0.88	3.81	161.98	4.85	0.67	4.91	98.91
T2	5.39	0.86	4.38	172.21	4.54	0.73	6.32	123.32
T3	5.09	0.79	3.75	212.89	4.95	0.61	4.12	188.12
T4	4.93	0.71	3.98	252.42	4.42	0.62	4.91	207.25
T5	4.81	0.68	4.47	311.54	4.06	0.51	5.11	267.89
T6	3.73	0.61	4.12	354.24	3.22	0.45	5.52	287.25
T7	3.21	0.58	4.97	381.41	2.91	0.41	6.23	321.34
T8	3.03	0.52	5.13	452.42	2.12	0.38	6.92	397.65

Tab. 7.12 : Résultats de la comparaison des approches de résolution AGL et AIT : cas problème $Pb1$

Dans le Tableau 7.12, les résultats conduisent aux points suivants. Les deux métriques HV et DG montrent que l'approche AGL domine légèrement l'approche AIT. Cependant, la taille du front de Pareto est très souvent légèrement grande avec l'approche AIT. Nous observons également que le temps de calcul de l'approche AIL est inférieur à celui de l'approche AGL, et ce pour tous les types d'instances. Sur la base de ces résultats, nous concluons que l'approche AGL surclasse légèrement l'approche AIT pour la résolution du problème $Pb1$.

7.4.3 Cas de deux agents en compétition avec un agent global ($Pb2$)

Cette section est consacrée à la résolution exacte et approchée du problème $Qm, m - skill \mid CO - GA, r_j, pmtn, f^A \leq Q_A, f^B \leq Q_B \mid f^G$. Tous d'abord, nous présentons les performances de l'approche exacte pour calculer le front de Pareto optimal, puis nous analysons les performances des heuristiques.

L'ensemble des tests conduits dans la suite sont effectués sur la base des conclusions du travail mené pour le cas $Pb1$ en terme de paramétrage des algorithmes proposés. Nous avons utilisé les mêmes jeux de données du cas $Pb1$.

7.4.3.1 Performances de la méthode exacte pour la résolution du $Pb2$

Cette section présente les résultats expérimentaux du PLNE-$Pb2$ où l'approche ε-contrainte est utilisée pour déterminer le front optimal.

Ensemble d'instances	Nombre moyen de solutions de Pareto faible	Nombre moyen de solutions de Pareto strict	Temps de calcul moyen du PLNE (s)
T1	7.76	6.86	36.56
T2	7.24	6.95	41.56
T3	6.98	6.21	56.78
T4	6.78	5.97	193.65
T5	5.16	5.65	392.21
T6	-	-	-
T7	-	-	-
T8	-	-	-

Tab. 7.13 : Front de Pareto exact : problème $Pb2$

Similaire aux résultats exacts du problème $Pb1$, on constate également que le modèle mathématique PLNE-$Pb2$ s'en sort bien pour la résolution des instances de moins de 100 ($T1$–$T5$). Par contre, il ne parvient pas à prouver l'optimalité des solutions retournées pour le reste des instances ($T6$–$T8$). Pour les instances $T1$–$T5$, le PLNE-$Pb2$ arrive à résoudre chaque solution au bout de 7 minutes. De même, nous remarquons que le nombre de solutions de Pareto faibles et strictes est plus important que dans le cas du problème $Pb1$. Ceci est dû au fait que certaines contraintes ont été relâchées. Pour les instances $T5$, par exemple, le PLNE-$Pb1$ génère en moyen 4.83 solutions strictes (resp. 5.14 solutions faibles), lorsque le PLNE-$Pb2$ en propose 5.65 solutions strictes (resp. 5.16 solutions faibles). D'autre part, nous observons qu'on général le temps de calcul est augmenté dans le cas du $Pb2$, ce qui s'explique par le fait de l'introduction d'une nouvelle contrainte et un nouveau objectif. Toujours pour les instances $T5$, le temps de calcul moyen du PLNE-$Pb1$ est égal à 342.43 seconds et celui du PLNE-$Pb2$ est égal à 392.21 seconds.

7.4.3.2 Performances des méthodes approchées AGL et AIT pour le problème $Pb2$

Dans cette partie de nos expérimentations, nous évaluions les résultats des approches de résolution AGL et AIT pour le cas du problème $Qm, m - skill \mid CO - GA, r_j, pmtn, f^A \leq Q_A, f^B \leq$

$Q_B \mid f^G$. Pour la comparaison des résultats, nous utilisons les trois métriques de performance (c'est-à-dire HV, DG, et TFP). Nous avons exécuté chacune des deux approches 10 fois et les valeurs moyennes sont calculées et présentées dans le Tableau 7.12. De même, les tests sont menés sur toutes les instances de nos ensembles de jeux de données.

Ensemble d'instances	*AGL*				*AIT*			
	TFP	HV	DG	CPU(s)	TFP	HV	DG	CPU(s)
T1	5.8	0.90	4.21	131.71	4.54	0.96	4.31	108.67
T2	5.18	0.92	4.58	197.44	5.89	0.95	4.92	184.67
T3	5.59	0.89	4.71	242.79	4.05	0.91	5.12	208.24
T4	3.12	0.88	4.68	292.28	4.22	0.87	5.75	236.71
T5	3.87	0.84	4.87	356.81	4.61	0.88	6.11	297.65
T6	2.35	0.68	4.92	394.33	3.92	0.75	6.24	331.23
T7	2.41	0.71	5.19	411.81	3.11	0.72	7.65	401.07
T8	2.37	0.57	5.98	472.92	2.97	0.68	7.13	421.16

Tab. 7.14 : Résultats de la comparaison des approches de résolution *AGL* et *AIT* cas problème $Pb2$

Les résultats numériques dans le Tableau 7.14 conduisent aux remarques suivantes. La première remarque concerne la métrique hypervolume. Nous pouvons constaté que les l'hypervolumes moyens dominés par les fronts de Pareto approchés sont proches pour les deux méthodes de résolution. La comparaison des fronts de Pareto approchés obtenus par les deux approches révèle que l'approche *AGL* est plus performante puisque les distances générationnele calculées sont plus petites comparées par les valeurs obtenues avec l'approche *AIT*. Cependant, le temps de calcul moyen de l'approche *AIL* est légèrement inférieur à celui de l'approche *AGL* et sont tous inférieurs à 8 minutes, et ce pour tous les types d'instances. Ces résultats confirment que l'approche *AGL* surclasse légèrement l'approche *AIT* pour la résolution des deux problèmes.

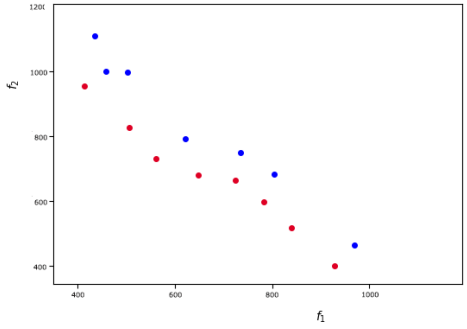
7.4.4 Analyse de l'apport de l'agent global sur les résultats des agents locaux

Cette section à pour objectif d'examiner l'effet d'intégrer un agent global pour la gestion des projets. A cette fin, nous comparons les retards des deux agents locaux pour les deux différents cas : celui où les violations des contraintes sont interdites (cas $Pb1$) et celui où les violations des contraintes souples sont autorisées (cas $Pb2$). Pour diversifier les tests, nous avons utilisé une instances de chaque ensemble de jeux de données. Les fronts de Pareto sont générés par l'approche *AGL* car celui qui donne les meilleurs résultats. Les résultats sont illustrés dans la Figure 7.8, où les points bleus sont les solutions du problème $Pb1$ et ceux en rouge sont les solutions du problème $Pb2$. Le Tableau 7.15 donne, pour chacune de ces 8 instances, les déviations moyennes des contraintes par rapport à leurs objectifs.

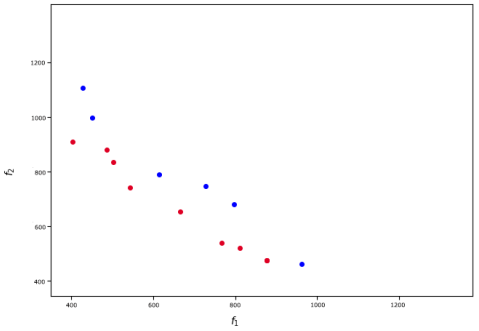
Ces résultats montrent que le fait de considérer un agent global a considérablement contribué à réduire les résultats des ensembles de tâches des agents locaux, c'est un fait escompté et attendu. De même, nous constatons que le degré de violation des contraintes souples est très raisonnable (de 0.58% et 0.72% pour l'instance de la plus petite taille ($T1$) jusqu'à 5.26% et 6.43% pour l'instance de la plus grande taille ($T8$)).

7.5 Conclusion

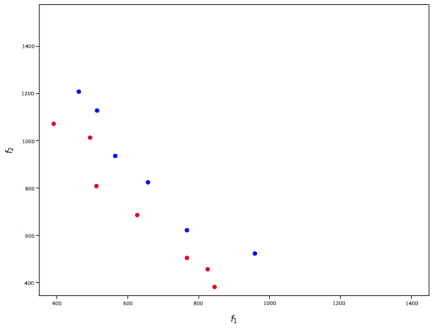
Dans ce chapitre, nous avons étudié des problèmes d'ordonnancement de projets multi-agent pour l'affectation de ressources multi-compétences aux charges des projets. Dans chacun de ces problèmes, plusieurs chefs de projets sont en concurrence pour l'utilisation de ressources (employés)



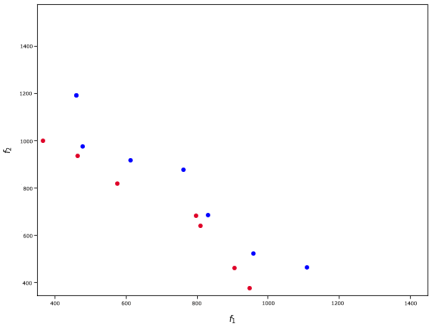
(a) Instance $T1$



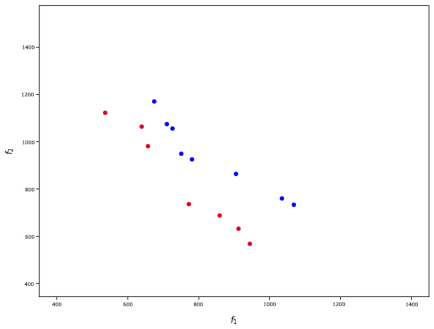
(b) Instance $T2$



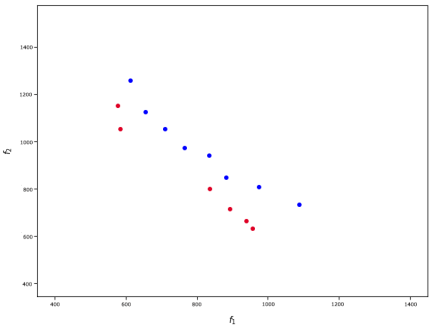
(c) Instance $T3$



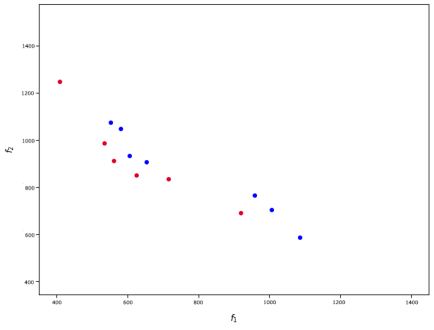
(d) Instance $T4$



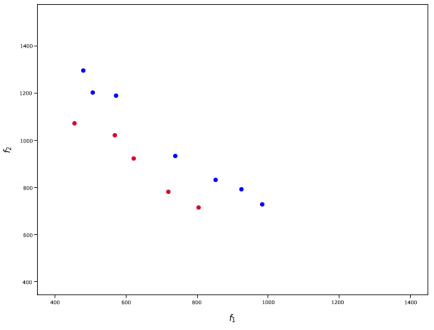
(e) Instance $T5$



(f) Instance $T6$



(g) Instance $T7$



(h) Instance $T8$

Fig. 7.8 : Exemples de fronts de Pareto

Tab. 7.15 : Écart moyen de chaque instance par rapport aux objectifs des contraintes

Instance	Objectif 1	Objectif 2
T1	0.58	0.72%
T2	0.62%	0.82%
T3	1.02%	2.14%
T4	2.82%	3.22%
T5	3.21%	3.84%
T6	4.33%	5.65%
T7	5.16%	5.94%
T8	5.26%	6.43%

dont la disponibilité est limitée et les compétences variées. Chaque chef de projet (ou agent) est responsable de l'exécution de son/ses projet(s), et doit donc négocier l'utilisation des ressources avec les autres chefs de projets. Tous les projets sont disjoints. Chaque chef de projet cherche donc à constituer son équipe d'employés en définissant une quotité allouée à chacun, c'est-à-dire à définir le taux de participation par semaine d'un employé aux projets en question. Un employé peut intervenir dans divers projets. Le but de chaque chef de projet est de minimiser la somme des retards pondérés de son sous-ensemble de tâches.

Deux problèmes d'ordonnancement font l'objet du travail présenté dans cette partie. Le premier problème (noté $Pb1$) est de type optimisation sous contrainte avec deux agents en compétition. Le deuxième problème (noté $Pb2$) s'agit d'un problème d'ordonnancement à deux agents en compétition et un objectif global. Il ajoute au premier problème le faite que certaines contraintes (dites contraintes souples, désignées par le décideur) peuvent être violées lorsqu'il n'est pas possible de trouver une solution réalisable pour le problème $Pb1$ ou pour diminuer le coût du retard des tâches de chaque agent. Ainsi, un objectif global consistant à minimiser la violation de ces contraintes est ajouté.

Afin de résoudre les problèmes étudiés, nous présentons deux modèles mathématiques basés sur la programmation linéaire en nombres entiers (PLNE). Ces modèles sont basés chacun sur l'approche ε-contrainte pour déterminer les fronts de Pareto exacts. Cependant, cette méthode n'est efficace que pour résoudre les instances de moins de 100 tâches. Nous développons donc par la suite deux approches heuristiques. Tout d'abord, une approche globale qui détermine une seule fois les quotités des employés aux projets via un modèle mathématique simplifié. Puis, à partir de ces quotités, calcule un ordonnancement pour minimiser le retard des tâches de chaque agent. La deuxième approche, qui est une approche itérative, commence la cherche avec des quotités initiales déterminées par une procédure heuristique. Puis, en ajustant ces quotités au cours des itérations, elle cherche à diminuer les retards. Les deux approches de résolution utilise un algorithme génétique NSGA-II efficace pour calculer les fronts de Pareto approchés. Nous avons également développé une recherche tabou qui est appelée pour générer la population initiale, appliquée sur des solutions initiales générées par les heuristiques gloutonnes. Les résultats de ces approches ont été comparés en termes de qualité des solutions sur des instances de taille réelle. Tous les algorithmes développés trouvent rapidement de bonnes solutions. Les expériences montrent une légère dominance de l'approche globale dans ce contexte.

Conclusion générale et perspectives

Dans cette thèse nous avons étudié un ensemble de problèmes d'ordonnancement multi-projet à contraintes de ressources multi-compétences partagées. Dans chacun de ces problèmes, de multiples projets doivent être exécutés simultanément et achevés dans un horizon de planification fixe (semaines consécutives). Chaque projet est décomposé en un ensemble de tâches indépendantes et préemptives, avec des dates de début au plus tôt et de fin souhaitées associées aux différentes tâches et sans des contraintes de précédence explicites. Chaque tâche doit être exécutée par un seul employé possédant plusieurs compétences et un niveau d'efficacité par compétence. De plus, ce niveau d'efficacité est prise en compte dans le temps d'exécution de la tâche. Les ressources considérées sont constituées par des équipes d'employés dont la disponibilité est limitée et les compétences sont variées. Un employé peut intervenir dans plusieurs projets. De plus, pour chaque employé nous avons une disponibilité hebdomadaire connue au début de l'ordonnancement et n'est pas sujette au changement. Dans chacun des problèmes étudiés, l'objectif est de déterminer un ordonnancement des tâches dans le temps, tout en minimisant une ou plusieurs fonctions objectifs. Un ordonnancement est défini par une affectation des employés aux tâches ainsi que la charge que chaque employé doit accomplir de chacune de ses tâches au cours de chaque semaine de l'horizon de planification connu.

Les problèmes qui nous intéressaient ont été examinés selon les deux cas de figures ci-dessous.

- Cas de quotités fixées : simule le cas idéal ou la décision d'affectation des employés aux projets est centralisée. Dans ce cas, les employées sont alloués aux différents projets selon des quotités maximales fixées par le décideur. Ainsi, les disponibilités par semaine de chaque employé sont réparties entre les différents projets au prorata de ses quotités sur ces projets. Ce cas fait l'objet du travail réalisé dans la première partie de la thèse (Chapitres 2, 3 et 4). Au Chapitre 2, nous introduisons des notions introductives sur l'ordonnancement de projets, et les différentes méthodes de résolutions, exactes et approchées de la littérature. Le Chapitre 3 établit un état de l'art approfondi des travaux en lien avec notre sujet. À savoir, problèmes d'ordonnancement de projets à contraintes de ressources limitées, problèmes d'ordonnancement de projet à contraintes de ressources multi-compétences et problèmes d'ordonnancement multi-projet. Le Chapitre 4 présente le modèle générale du problème résolu dans la première partie de la thèse. Comme susmentionné, dans ce modèle, les quotités ou les taux maximaux d'allocation des employés aux projets sont prédéterminés. Une seule fonction objectif est à minimiser : la somme des retards pondérés plus la somme pondérée des violations des contraintes molles. Il s'agit donc d'un travail sur l'ordonnancement de projet monocritère.

- Cas de quotités non-fixées : simule le cas où la décision d'affectation des employés aux projets est le fruit d'un processus non centralisé faisant intervenir plusieurs agents (chefs de projets). Chaque agent est responsable de son propre sous-ensemble de tâches et cherche à optimiser une fonction objectif qui dépend uniquement des dates de fin d'exécution de ses tâches, il est

en compétition avec les autres agents sur l'utilisation des ressources. Deux types de problèmes d'ordonnancement à deux agents ont été abordés. Le premier est de type optimisation sous contrainte, où l'objectif est de minimiser le critère d'un agent, tout en bornant supérieurement la valeur du critère du second agent. Nous nous intéressons à générer le front de Pareto. L'approche ε-contrainte est utilisée principalement pour déterminer le front exact. Le second problème d'ordonnancement traite le cas de deux agents avec une fonction objectif globale. Il ajoute au deuxième problème le faite que certaines contraintes peuvent être violées. Ainsi, un objectif global est ajouté consistant à minimiser la violation de ces contraintes en réduisant les déviations indésirables par rapport aux objectifs de ces contraintes. Ces problèmes sont donc abordés dans la deuxième partie du manuscrit (Chapitres 5, 6 et 7). Le Chapitre 5 introduit des notions générales sur l'ordonnancement multi-agent. Le Chapitre 6 dresse un état de l'art des problèmes d'ordonnancement multi-agent. Le Chapitre 7 présente les problèmes traités. Nous présentons également dans le même chapitre différentes approches de résolution, exactes et approchées, appliquées sur ces problèmes.

Notons que l'ensemble des approches de résolutions proposées peuvent être généralisées aux cas de plusieurs agents, moyennant un temps de calcul plus important. Toutes nos réflexions pour développer ces méthodes de résolution ont été menées avec l'idée d'adapter "facilement" ses approches de résolution aux cas de plusieurs projets en considérant le cas monocritère ou multicritère, quelque soit le nombre d'agents à considérer. Et ainsi, aider le décideur de la société *Alfa Conseil* dans la répartition des charges de travail sur l'ensemble des employés, d'honorer ses engagements envers ses clients et de réduire les coûts dûs aux retards des réalisation.

Les futures recherches peuvent prendre plusieurs directions. En premier lieu, finalisé le déploiement des solutions proposées au sein de la société pour une intégration totale dans leur outil de gestion de projet.

En second lieu, il nous semble intéressent de proposer des scénarios de tests pour valider la robustesse des solutions proposées en intégrant différents types d'incertitude (temps de réalisation, absence des personnes non prévues). De même, il n'a pas échappé au lecteur que la méthode ϵ-contrainte proposée pour résoudre le problème du Chapitre 7 ne peut pas résoudre les instances comportant plus de 100 tâches. On a donc proposé des approches hybrides basées sur le développement des heuristiques efficaces lorsque la taille des instances deviennent importante et donc lorsque le solveur échoue. Une autre approche de type math-heuristique serait intéressante de développer. En effet, vue l'efficacité des $PLNE - PbX$ $(X = 1, 2)$ pour résoudre des instances de taille moyenne, ils peuvent être hybridés avec la recherche tabou ou NSGA-II proposés.

Une autre perspective concerne la modélisation des contraintes molles sur le graphe de flot maximum à coût minimum utilisé pour évaluer une affectation et/ou développer une procédure permettant à chaque itération du calcul du flot à coût minimum de bien choisir la chaîne augmentante du flot en privilégiant des chemins violant certaines contraintes désignées par le décideur. Ce travail peut débuter par le choix et la détermination des coûts des arcs appropriés.

Les problèmes étudiés sont $\mathcal{NP}$-difficiles au sens fort. Il serait intéressant de développer des bornes inférieures efficaces permettant déjà d'avoir des fronts de références de qualité lorsque le nombre de tâches à traiter devient important. Dans un second lieu, il serait tout à fait intéressant d'élaborer des procédures par séparation et évaluation.

Les problèmes abordés ici admettent des agents locaux disjoints, il est intéressant d'élargir l'étude pour le cas où les agents locaux ne sont pas forcément disjoints. En effet, certains développements informatiques pour un projet peut-être bénéfique pour un autre projet se déroulant en même temps, i.e. les agents locaux ont leur propre tâches et peuvent aussi partager d'autres.

Dans cette étude, seul le critère principal considéré est la moyenne pondérée des retards. Un autre critère peut être étudié dans un futur proche est le retard maximum (pondéré ou non). Prendre en compte ce critère permet ainsi d'éviter des déviations plus importantes pour certains

projets. Une analyse plus fine de nos résultats expérimentaux devrait être entreprise pour confirmer nos premières constations et d'affirmer dans quelles situations cela se produit. Ce travail débutera par l'analyse des instances générées.

Enfin, nous nous sommes intéressés à la gestion des projets issue du domaine opérationnelle. Nous souhaitons dans un futur intégrer en même temps l'aspect décisionnel de la gestion de projet, en s'appuyant sur les outils de la Recherche Opérationnelle (RO). Cette future étude permet aussi d'aider le décideur à *"faire les bons projets"*. Notre idée est de prendre en compte un autre critère qui est la minimisation des coûts de rejets des projets. Ce critère vient en plus en aide au décideur dans l'atteinte des objectifs prescrits dans le contrat de service et respecter ainsi ses engagements sur les délais de livraison.

Les travaux réalisés ont fait l'objet de publications dans deux revues internationales ainsi que dans des conférences nationales et internationales. Nous récapitulons ci-dessous l'ensemble des publications.

1. Haroune, M., Dhib, C., Néron, E., Soukhal, A., Babou, H., and Nanne, M. F. (2022b). Multi-project scheduling problem under shared multi-skill resource constraints. *TOP*
2. Haroune, M., Dhib, C., Néron, E., Soukhal, A., Babou, H., and Nanne, M. F. (2022a). A hybrid heuristic for a two-agent multi-skill resource-constrained scheduling problem. *International Journal of Advanced Computer Science and Applications (IJACSA)*
3. (PMS'21) Haroune, M., Dhib, C., Néron, E., Soukhal, A., Babou, H., and Nanne, M. F. (2021a). Multi-project scheduling problems with shared multi-skill resource constraints
4. (ROADEF'21) Haroune, M., Dhib, C., Néron, E., Soukhal, A., Babou, H., and Nanne, M. F. (2021b). Problème d'ordonnancement multi-projets et d'allocation des ressources humaines multi-compétences aux projets
5. (ROADEF'20) Haroune, M., Dhib, C., Néron, E., Soukhal, A., Babou, H., and Nanne, M. F. (2020). Ordonnancement multi-projet à contraintes de ressources partagées par plusieurs agents
6. (ROADEF'19) Haroune, M., Dhib, C., Néron, E., Soukhal, A., Babou, H., and Nanne, M. F. (2019). Ordonnancement multi-projet à contraintes de ressources partagées multi-compétences

Bibliographie

Adhau, S., Mittal, M., and Mittal, A. (2012). A multi-agent system for distributed multi-project scheduling : An auction-based negotiation approach. *Engineering Applications of Artificial Intelligence*, 25(8) :1738–1751.

Afshar-Nadjafi, B. and Majlesi, M. (2014). Resource constrained project scheduling problem with setup times after preemptive processes. *Computers & Chemical Engineering*, 69 :16–25.

Agnetis, A., Billaut, J.-C., Gawiejnowicz, S., Pacciarelli, D., and Soukhal, A. (2014). *Multiagent Scheduling : Models and Algorithms*. Springer, Berlin, Heidelberg.

Agnetis, A., De Pascale, G., and Pacciarelli, D. (2009). A lagrangian approach to single-machine scheduling problems with two competing agents. *Journal of Scheduling*, 12(4) :401–415.

Agnetis, A., Mirchandani, P. B., Pacciarelli, D., and Pacifici, A. (2004). Scheduling problems with two competing agents. *Operations Research*, 52(2) :229–242.

Ahuja, R. K., Magnanti, T. L., and Orlin, J. B. (1993). *Network Flows : Theory, Algorithms, and Applications*. Pearson Education.

Almeida, B. F., Correia, I., and da Gama, F. S. (2016). Priority-based heuristics for the multi-skill resource constrained project scheduling problem. *Expert Systems with Applications*, 57 :91–103.

Alonso-Pecina, F., Pecero, J., and D.Romero (2013). A three-phases based algorithm for the multi-mode resource-constrained multi-project scheduling problem. In *6th Multidisciplinary International Scheduling Conference (MISTA)*, page 812–814.

Araujo, J. A., Santos, H. G., Gendron, B., Jena, D. S., Brito, S. S., and Souza, D. S. (2020). Strong bounds for resource constrained project scheduling : Preprocessing and cutting planes. *Computers & Operations Research*, 113 :104782.

Asta, S., Karapetyan, D., Kheiri, A., Özcan, E., and Parkes, A. J. (2016). Combining monte-carlo and hyper-heuristic methods for the multi-mode resource-constrained multi-project scheduling problem. *Information Sciences*, 373 :476–498.

Baker, K. and Smith, J.-C. (2003). A multiple-criterion model for machine scheduling. *Journal of Scheduling*, 6 :7–16.

Balasubramanian, H., Fowler, J., Keha, A., and Pfund, M. (2009). Scheduling interfering job sets on parallel machines. *European Journal of Operational Research*, 199(1) :55–67.

Ballestín, F., Valls, V., and Quintanilla, S. (2008). Pre-emption in resource-constrained project scheduling. *European Journal of Operational Research*, 189(3) :1136–1152.

Bellenguez, O. and Néron, E. (2005). Lower bounds for the multi-skill project scheduling problem with hierarchical levels of skills. In *Practice and Theory of Automated Timetabling V*, pages 229–243. Springer Berlin Heidelberg.

Bellenguez, O. and Néron, E. (2007). A branch-and-bound method for solving multi-skill project scheduling problem. *RAIRO - Operations Research*, 41(2) :155–170.

Bellenguez, O. and Néron, E. (2008). *Multi-Mode and Multi-Skill Project Scheduling Problem*, chapter 9, pages 149–160. John Wiley & Sons, Ltd.

Ben Issa, S., Patterson, R. A., and Tu, Y. (2021). Solving resource-constrained multi-project environment under different activity assumptions. *International Journal of Production Economics*, 232 :107936.

Bernhard, K. and Jens, V. (2006). *Bin-Packing*, volume 21, pages 426–441. Springer.

Beşikci, U., Ümit Bilge, and Ulusoy, G. (2015). Multi-mode resource constrained multi-project scheduling and resource portfolio problem. *European Journal of Operational Research*, 240(1) :22–31.

Bianco, L., Caramia, M., and Dell'Olmo, P. (1999). *Solving a Preemptive Project Scheduling Problem with Coloring Techniques*, volume 1, pages 135–145. Springer US.

Blazewicz, J., Lenstra, J., and Kan, A. (1983a). Scheduling subject to resource constraints : classification and complexity. *Discrete Applied Mathematics*, 5(1) :11 – 24.

Blazewicz, J., Lenstra, J. K., and Kan, A. H. G. R. (1983b). Scheduling subject to resource constraints : classification and complexity. *Discrete Applied Mathematics*, 5(1) :11–24.

Browning, T. and A.Yassine (2010). Resource-constrained multi-project scheduling : Priority rule performance revisited. *International Journal of Production Economics*, 126(2) :212–228.

Brucker, P., Bernd, B. J., and Krämer, A. (1997). Complexity of scheduling problems with multi-purpose machines. *Annals of Operations Research*, 70(0) :57–73.

C.E.Cheng, T., Cheng, S.-R., Wu, W.-H., Hsu, P.-H., and Wu, C.-C. (2011). A two-agent single-machine scheduling problem with truncated sum-of-processing-times-based learning considerations. *Computers and Industrial Engineering*, 60(4) :534–541.

Chen, L. and Zhang, Z. (2016). Preemption resource-constrained project scheduling problems with fuzzy random duration and resource availabilities. *Journal of Industrial and Production Engineering*, 33(6) :373–382.

Chen, R., Liang, C., Gu, D., and Leung, J. (2017). A multi-objective model for multi-project scheduling and multi-skilled staff assignment for it product development considering competency evolution. *International Journal of Production Research*, 55(21) :6207–6234.

Chen, R. X. and Li, S. S. (2019). Two-agent single-machine scheduling with cumulative deterioration. *4or*, 17(2) :201–219.

Cheng, J., Fowler, J., Kempf, K., and Mason, S. (2015). Multi-mode resource-constrained project scheduling problems with non-preemptive activity splitting. *Computers & Operations Research*, 53 :275–287.

Cheng, T., Chung, Y.-H., Liao, S.-C., and Lee, W.-C. (2013). Two-agent singe-machine scheduling with release times to minimize the total weighted completion time. *Computers & Operations Research*, 40(1) :353–361.

Cheng, T., Ng, C., and Yuan, J. (2006). Multi-agent scheduling on a single machine to minimize total weighted number of tardy jobs. *Theoretical Computer Science*, 362(1) :273–281.

Choi, B.-C. and Park, M.-J. (2017). Two-agent parallel machine scheduling with a restricted number of overlapped reserved tasks. *European Journal of Operational Research*, 260(2) :514–519.

Choi, B.-C., Park, M.-J., and Du, J. (2020). Scheduling two projects with controllable processing times in a single-machine environment. *Journal of Scheduling*, 23(1) :619–628.

Choi, I. and Choi, D. (2002). A local search algorithm for jobshop scheduling problems with alternative operations and sequence-dependent setups. *Computers & Industrial Engineering*, 42 :43–58.

Correia, I. and da Gama, F. S. (2014). The impact of fixed and variable costs in a multi-skill project scheduling problem : An empirical study. *Computers & Industrial Engineering*, 72 :230–238.

Correia, I., Lourenço, L. L., and da Gama, F. S. (2012). Project scheduling with flexible resources : formulation and inequalities. *OR Spectrum*, 34 :635–663.

Creemers, S. (2019). The preemptive stochastic resource-constrained project scheduling problem. *European Journal of Operational Research*, 277(1) :238–247.

Cui, L., Liu, X., Lu, S., and Jia, Z. (2021). A variable neighborhood search approach for the resource-constrained multi-project collaborative scheduling problem. *Applied Soft Computing*, 107 :107480.

Dai, H. and Cheng, W. (2019). A memetic algorithm for multiskill resource-constrained project scheduling problem under linear deterioration. *Mathematical Problems in Engineering*, 2019.

Damay, J., Quilliot, A., and Sanlaville, E. (2007). Linear programming based algorithms for preemptive and non-preemptive rcpsp. *European Journal of Operational Research*, 182(3) :1012–1022.

De, K., Miettine, K., and Chaudhuri, S. (2011). Toward an estimation of nadir objective vector using a hybrid of evolutionary and local search approaches. *Evolutionary Computation, IEEE Transactions on*, 14 :821 – 841.

Deb, K., Pratap, A., Agarwal, S., and Meyarivan, T. (2002). A fast and elitist multiobjective genetic algorithm : NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6(2) :182–197.

Demeulemeester, E. and Herroelen, W. S. (1992). A branch-and-bound procedure for the multiple resource-constrained project scheduling problem. *Management Science*, 38(2) :1687–1826.

Demeulemeester, E. and Herroelen, W. S. (1996). An efficient optimal solution procedure for the preemptive resource-constrained project scheduling problem. *European Journal of Operational Research*, 90(2) :334–348.

Dhib, C., Soukhal, A., and Néron, E. (2015). *Mixed-Integer Linear Programming Formulation and Priority-Rule Methods for a Preemptive Project Staffing and Scheduling Problem*, volume 1, pages 603–617. Springer International Publishing, Cham.

Drezet, L.-E. and Billaut, J.-C. (2008). A project scheduling problem with labour constraints and time-dependent activities requirements. *International Journal of Production Economics*, 112(1) :217–225. Special Section on Recent Developments in the Design, Control, Planning and Scheduling of Productive Systems.

Edmonds, J. and Karp, R. M. (1972). *Theoretical Improvements in Algorithmic Efficiency for Network Flow Problems*, volume 19, page 248–264.

Elvikis, D., Hamacher, H. W., Vincent, and T'kindt (2011). Scheduling two agents on uniform parallel machines with makespan and cost functions. *Journal of Scheduling*, 14(5) :471–481.

Elvikis, D. and T'kindt, V. (2014). Two-agent scheduling on uniform parallel machines with min-max criteria. *Annals of Operations Research*, 213(1) :79–94.

Eynde, R. V. and Vanhoucke, M. (2020). Resource-constrained multi-project scheduling : benchmark datasets and decoupled scheduling. *Journal of Scheduling*, 23 :301–325.

F., L. and Z., X. (2018). A multi-agent system for distributed multi-project scheduling with two-stage decomposition. *PLoS ONE*, 10.

Fan, B. Q., Cheng, T. C., Li, S. S., and Feng, Q. (2013). Bounded parallel-batching scheduling with two competing agents. *Journal of Scheduling*, 16(3) :261–271.

Felberbauer, T., Gutjahr, W. J., and Doerner, K. F. (2019). Stochastic project management : multiple projects with multi-skilled human resources. *Journal of Scheduling*, 22 :271–288.

Fırat, M. and Hurkens, C. A. J. (2012). An improved mip-based approach for a multi-skill workforce scheduling problem. *Journal of Scheduling*, 15.

Galvagnon, V. (2000). *Aide à la décision en gestion multi-projet distribuée : approche locale pour la planification à moyen terme*. PhD thesis.

Gawiejnowicz, S. and Suwalski, C. (2014). Scheduling linearly deteriorating jobs by two agents to minimize the weighted sum of two criteria. *Computers & Operations Research*, 52 :135–146.

Geiger, M. J. (2017). A multi-threaded local search algorithm and computer implementation for the multi-mode, resource-constrained multi-project scheduling problem. *European Journal of Operational Research*, 256(3) :729–741.

Geoffrion, A. M. (1968). Proper efficiency and the theory of vector maximization. *Journal of Mathematical Analysis and Applications*, 22(3) :618–630.

Gerstl, E. and Mosheiov, G. (2013). Scheduling problems with two competing agents to minimized weighted earliness–tardiness. *Computers & Operations Research*, 40(1) :109–116.

Glover, F. (1989). Tabu search—part i. *ORSA Journal on Computing*, 1(3) :190–206.

Glover, F. (1990). Tabu search—part ii. *ORSA Journal on Computing*, 2(1) :4–32.

Gonçalves, J. F., de Magalhães Mendes, J. J., and Resende, M. G. C. (2015). *The Basic Multi-Project Scheduling Problem*, pages 667–683. Springer International Publishing.

Gonçalves, J. F., Mendes, J. J. M., and Resende, M. G. C. (2008). A genetic algorithm for the resource constrained multi-project scheduling problem. *European Journal of Operational Research*, 189(3) :1171–1190.

Graham, R., Lawler, E., Lenstra, J., and Kan, A. (1979). Optimization and approximation in deterministic sequencing and scheduling : a survey. In Hammer, P., Johnson, E., and Korte, B., editors, *Discrete Optimization II*, volume 5 of *Annals of Discrete Mathematics*, pages 287–326. Elsevier.

Gutjahr, W. J., Katzensteiner, S., Reiter, P., Stummer, C., and Denk, M. (2010). Multi-objective decision analysis for competence-oriented project portfolio selection. *European Journal of Operational Research*, 205(3) :670–679.

HAIMES, Y. (1971). On a bicriterion formulation of the problems of integrated system identification and system optimization. *IEEE Transactions on Systems, Man, and Cybernetics*, 1(3) :296–297.

Haitao, L. and Keith, W. (2008). Scheduling projects with multi-skilled personnel by a hybrid milp/cp benders decomposition algorithm. *Journal of Scheduling*, 12 :281.

Haroune, M., Dhib, C., Néron, E., Soukhal, A., Babou, H., and Nanne, M. F. (2019). Ordonnancement multi-projet à contraintes de ressources partagées multi-compétences.

Haroune, M., Dhib, C., Néron, E., Soukhal, A., Babou, H., and Nanne, M. F. (2020). Ordonnancement multi-projet à contraintes de ressources partagées par plusieurs agents.

Haroune, M., Dhib, C., Néron, E., Soukhal, A., Babou, H., and Nanne, M. F. (2021a). Multi-project scheduling problems with shared multi-skill resource constraints.

Haroune, M., Dhib, C., Néron, E., Soukhal, A., Babou, H., and Nanne, M. F. (2021b). Problème d'ordonnancement multi-projets et d'allocation des ressources humaines multi-compétences aux projets.

Haroune, M., Dhib, C., Néron, E., Soukhal, A., Babou, H., and Nanne, M. F. (2022a). A hybrid heuristic for a two-agent multi-skill resource-constrained scheduling problem. *International Journal of Advanced Computer Science and Applications (IJACSA)*.

Haroune, M., Dhib, C., Néron, E., Soukhal, A., Babou, H., and Nanne, M. F. (2022b). Multi-project scheduling problem under shared multi-skill resource constraints. *TOP*.

Hartmann, S. and Briskorn, D. (2010). A survey of variants and extensions of the resource-constrained project scheduling problem. *European Journal of Operational Research*, 207(1) :1 – 14.

Hartmann, S. and Briskorn, D. (2021). An updated survey of variants and extensions of the resource-constrained project scheduling problem. *European Journal of Operational Research*.

He, Y., He, Z., and Wang, N. (2020). Tabu search and simulated annealing for resource-constrained multi-project scheduling to minimize maximal cash flow gap. *Journal of Industrial & Management Optimization*, 0 :1547–5816.

Heimerl, C. and Kolisch, R. (2010a). Scheduling and staffing multiple projects with a multi-skilled workforce. *OR Spectrum*, 32(2) :19–25.

Heimerl, C. and Kolisch, R. (2010b). Work assignment to and qualification of multi-skilled human resources under knowledge depreciation and company skill level targets. *International Journal of Production Research*, 48(13) :3759–3781.

Hematian, M., Esfahani, M., Mahdavi, I., Mahdavi-Amiri, N., and Rezaeian, J. (2020). A multiobjective integrated multiproject scheduling and multiskilled workforce assignment model considering learning effect under uncertainty. *Computational Intelligence*, 36(1) :276–296.

Herroelen, W., Reyck, B. D., and Demeulemeester, E. (1998). Resource-constrained project scheduling : A survey of recent developments. *Computers & Operations Research*, 25(4) :279 – 302.

Hossein, H. A. and Baradaran, V. (2020). P-gwo and mofa : two new algorithms for the msrcpsp with the deterioration effect and financial constraints (case study of a gas treating company). *Applied Intelligence*, 50 :2151–2176.

Ishibuchi, H. and Murata, T. (1998). A multi-objective genetic local search algorithm and its application to flowshop scheduling. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 28(3) :392–403.

Kaplan, L. (1988). *Resource-constrained Project Scheduling with Preemption of Jobs*. PhD thesis.

Kellegöz, T., Toklu, B., and Wilson, J. (2008). Comparing efficiencies of genetic crossover operators for one machine total weighted tardiness problem. *Applied Mathematics and Computation*, 199(2) :590–598.

Kolisch, R. and Hartmann, S. (1999). *Heuristic Algorithms for the Resource-Constrained Project Scheduling Problem : Classification and Computational Analysis*, pages 147–178. Springer US, Boston, MA.

Kolisch, R. and Heimerl, C. (2012). An efficient metaheuristic for integrated scheduling and staffing it projects based on a generalized minimum cost flow network. *Naval Research Logistics (NRL)*, 59(2) :111–127.

Kolisch, R. and Sprecher, A. (1997). Psplib - a project scheduling problem library : Or software - orsep operations research software exchange program. *European Journal of Operational Research*, 96(1) :205–216.

Koné, O., Artigues, C., Lopez, P., and Mongeau, M. (2011). Event-based milp models for resource-constrained project scheduling problems. *Computers & Operations Research*, 38(1) :3–13. Project Management and Scheduling.

Kovalyov, Y. M., Oulamara, A., and Soukhal, A. (2015). Two-agent scheduling with agent specific batches on an unbounded serial batching machine. *Journal of Scheduling*, 18(2) :423–434.

Kumar, M., Mittal, M., Soni, G., and Joshi, D. (2018). A hybrid tlbo-ts algorithm for integrated selection and scheduling of projects. *Computers & Industrial Engineering*, 119 :121–130.

Labetoulle, J., Lawler, E., Lenstra, J., and Kan, A. R. (1984). Preemptive scheduling of uniform machines subject to release dates. In *Progress in Combinatorial Optimization*, pages 245 – 261. Academic Press.

Laszczyk, M. and Myszkowski, P. B. (2019). Improved selection in evolutionary multi–objective optimization of multi–skill resource–constrained project scheduling problem. *Information Sciences*, 481 :412–431.

Lee, W.-C., Chung, Y.-H., and Hu, M.-C. (2012). Genetic algorithms for a two-agent single-machine problem with release time. *Applied Soft Computing*, 12(11) :3580–3589.

Lee, W.-C., Wang, J.-Y., and Lin, M.-C. (2016). A branch-and-bound algorithm for minimizing the total weighted completion time on parallel identical machines with two competing agents. *Knowledge-Based Systems*, 105 :68–82.

Lee, W. C., Wang, J. Y., and Su, H. W. (2015). Algorithms for single-machine scheduling to minimize the total tardiness with learning effects and two competing agents. *Concurrent Engineering Research and Applications*, 23(1) :13–26.

Leo, L. (2009). Reformulations in mathematical programming : Definitions and systematics. *RAIRO - Operations Research*, 43(1) :55–85.

Leung, J. Y.-T., Pinedo, M., and Wan, G. (2010). Competitive two-agent scheduling and its applications. *Operations Research*, 58(2) :458–469.

Li, D. and Lu, X. (2017). Two-agent parallel-machine scheduling with rejection. *Theoretical Computer Science*, 703 :66–75.

Li, J., Gajpal, Y., and Appadoo, S. S. (2021a). Algorithms for a two-agent single machine scheduling problem to minimize weighted number of tardy jobs. *Journal of Information and Optimization Sciences*, 42(4) :785–811.

Li, J., Gajpal, Y., Bhardwaj, A. K., Chen, H., and Liu, Y. (2021b). Two-Agent Single Machine Order Acceptance Scheduling Problem to Maximize Net Revenue. *Complexity*, 2021.

Li, M. and Yao, X. (2019). *Quality evaluation of solution sets in multiobjective optimisation : A survey*, volume 52.

Li, S. and Yuan, J. (2012). Unbounded parallel-batching scheduling with two competitive agents. *Journal of Scheduling*, 15(5) :629–640.

Li, S. S., Chen, R. X., and Feng, Q. (2016). Scheduling two job families on a single machine with two competitive agents. *Journal of Combinatorial Optimization*, 32(3) :784–799.

Lin, J., Zhu, L., and Gao, K. (2020). A genetic programming hyper-heuristic approach for the multi-skill resource constrained project scheduling problem. *Expert Systems with Applications*, 140 :112915.

Liu, P., Gu, M., and Li, G. (2019). Two-agent scheduling on a single machine with release dates. *Computers & Operations Research*, 111 :35–42.

Liu, P., Yi, N., and Zhou, X. (2011). Two-agent single-machine scheduling problems under increasing linear deterioration. *Applied Mathematical Modelling*, 35(5) :2290–2296.

Liu, P., Yi, N., Zhou, X., and Gong, H. (2013). Scheduling two agents with sum-of-processing-times-based deterioration on a single machine. *Applied Mathematics and Computation*, 219(17) :8848–8855.

Liu, S.-S. and Wang, C.-J. (2011). Optimizing project selection and scheduling problems with time-dependent resource constraints. *Automation in Construction*, 20(8) :1110–1119.

Maghsoudlou, H., Afshar-Nadjafi, B., and Akhavan Niaki, S. T. (2017). Multi-skilled project scheduling with level-dependent rework risk ; three multi-objective mechanisms based on cuckoo search. *Applied Soft Computing*, 54 :46–61.

Maghsoudlou, H., Afshar-Nadjafi, B., and Niaki, S. (2019). Preemptive multi-skilled resource constrained project scheduling problem with hard/soft interval due dates. *RAIRO - Operations Research*, 53 :1877–1898.

Mei, B., Lambrechts, A., Verkest, D., Mignolet, J. Y., and Lauwereins, R. (2005). Architecture exploration for a reconfigurable architecture template. *IEEE Design and Test of Computers*, 22(2) :90–101.

Mika, M., Waligóra, G., and Weglarz, J. (2008). Tabu search for multi-mode resource-constrained project scheduling with schedule-dependent setup times. *European Journal of Operational Research*, 187(3) :1238 – 1250.

Montoya, C., Bellenguez, O., Pinson, E., and Rivreau, D. (2014). Branch-and-price approach for the multi-skill project scheduling problem. *Optimization Lettersh*, 8 :1721–1734.

Mor, B. and Mosheiov, G. (2011). Single machine batch scheduling with two competing agents to minimize total flowtime. *European Journal of Operational Research*, 215(3) :524–531.

Moukrim, A., Quilliot, A., and Toussaint, H. (2015). An effective branch-and-price algorithm for the preemptive resource constrained project scheduling problem based on minimal interval order enumeration. *European Journal of Operational Research*, 244(2) :360–368.

Myszkowski, P. B., Laszczyk, M., and Kalinowski, D. (2017). Co-evolutionary algorithm solving multi-skill resource-constrained project scheduling problem. In *2017 Federated Conference on Computer Science and Information Systems (FedCSIS)*, pages 75–82.

Myszkowski, P. B., Skowroński, M. E., and Sikora, K. (2015). A new benchmark dataset for multi-skill resource-constrained project scheduling problem. In *2015 Federated Conference on Computer Science and Information Systems (FedCSIS)*, pages 129–138.

Myszkowski, P. B., Łukasz P. Olech, Laszczyk, M., and Skowroński, M. E. (2018). Hybrid differential evolution and greedy algorithm (degr) for solving multi-skill resource-constrained project scheduling problem. *Applied Soft Computing*, 62 :1–14.

Néron, E. and Baptista, D. (2002). Lower bounds for the multi-skill project scheduling problem. In *the Eighth International Workshop on Project Management and Scheduling*, pages 274–277.

Ng, C. T., Cheng, T. C., and Yuan, J. J. (2006). A note on the complexity of the problem of two-agent scheduling on a single machine. *Journal of Combinatorial Optimization*, 12(4) :386–393.

Pan, N., Hsaio, P., and Chen, K. (2008). A study of project scheduling optimization using tabu search algorithm. *Engineering Applications of Artificial Intelligence*, 21(7) :1101 – 1112.

Pei, J., Wei, J., Liao, B., Liu, X., and Pardalos, P. M. (2020). Two-agent scheduling on bounded parallel-batching machines with an aging effect of job-position-dependent. *Annals of Operations Research*, 294(1-2) :191–223.

Polo-Mejía, O., Artigues, C., Lopez, P., and Basini, V. (2019). Mixed-integer/linear and constraint programming approaches for activity scheduling in a nuclear research facility. *International Journal of Production Research*, 58(23) :7149–7166.

Pérez, E., Posada, M., and Lorenzana, A. (2016). Taking advantage of solving the resource constrained multi-project scheduling problems using multi-modal genetic algorithms. *Soft Computing*, 20 :1879–1896.

Quintanilla, S., Lino, P., Pérez, Á., Ballestín, F., and Valls, V. (2015). *Integer Preemption Problems*, pages 231–250. Springer International Publishing.

Reisi-Nafchi, M. and Moslehi, G. (2015). A hybrid genetic and linear programming algorithm for two-agent order acceptance and scheduling problem. *Applied Soft Computing*, 33 :37–47.

Rosenkrantz, D. J., Stearns, R. E., and Lewis, P. M. (1974). Approximate algorithms for the traveling salesperson problem. In *15th Annual Symposium on Switching and Automata Theory (swat 1974)*, pages 33–42.

Sabouni, M. Y. and Jolai, F. (2010). Optimal methods for batch processing problem with makespan and maximum lateness objectives. *Applied Mathematical Modelling*, 34(2) :314–324.

Sadi, F. and Soukhal, A. (2017). Complexity analyses for multi-agent scheduling problems with a global agent and equal length jobs. *Discrete Optimization*, 23 :93–104.

Sadi, F., Soukhal, A., and Billaut, J. (2014). Solving multi-agent scheduling problems on parallel machines with a global objective function. *RAIRO - Operations Research*, 23 :255–269.

Sahu, S. N., Gajpal, Y., and Debbarma, S. (2018). Two-agent-based single-machine scheduling with switchover time to minimize total weighted completion time and makespan objectives. *Annals of Operations Research*, 269(1-2) :623–640.

Schwindt, C. and Paetz, T. (2015). *Continuous Preemption Problems*, pages 251–295. Springer International Publishing.

Schwindt, C. and Zimmermann, J. (2015). *Handbook on Project Management and Scheduling*, volume 2. Springer.

Shariatmadari, M., Nahavandi, N., Zegordi, S. H., and Sobhiyah, M. H. (2017). Integrated resource management for simultaneous project selection and scheduling. *Computers & Industrial Engineering*, 109 :39–47.

Shou, Y., Li, Y., and Lai, C. (2015). Hybrid particle swarm optimization for preemptive resource-constrained project scheduling. *Neurocomputing*, 148 :122–128.

Singh, A. (2014). Resource constrained multi-project scheduling with priority rules & analytic hierarchy process. *Procedia engineering*, 69 :725–734.

Snauwaert, J. and Vanhoucke, M. (2021). A new algorithm for resource-constrained project scheduling with breadth and depth of skills. *European Journal of Operational Research*, 292(1) :43–59.

Srinivas, N. and Deb, K. (1994). Muiltiobjective Optimization Using Nondominated Sorting in Genetic Algorithms. *Evolutionary Computation*, 2(3) :221–248.

Syswerda, G. (1991). *Scheduling optimization using genetic algorithms*. New York, NY.

SłowínskiR, R. (1980). Two approaches to problems of resource allocation among project activities : a comparative study. *Journal of Operationnal Research*, 31 :711–723.

T'kindt, V. and Billaut, J.-C. (2001). Multicriteria scheduling problems : a survey. *RAIRO-Oper. Res.*, 35(2) :143–163.

Toffolo, T. A. M., Santos, H. G., Carvalho, M. A. M., and Soares, J. A. (2016). An integer programming approach to the multimode resource-constrained multiproject scheduling problem. *Journal of Scheduling*, 19 :295–307.

Tofighian, A. A. and Naderi, B. (2015). Modeling and solving the project selection and scheduling. *Computers & Industrial Engineering*, 83 :30–38.

Tseng, L. and Lin, Y. (2009). A hybrid genetic local search algorithm for the permutation flowshop scheduling problem. *European Journal of Operational Research*, 198 :84–92.

Valls, V., Ángeles Pérez, and Quintanilla, S. (2009). Skilled workforce scheduling in service centres. *European Journal of Operational Research*, 193(3) :791–804.

Vanhoucke, M. and Coelho, J. (2019). Resource-constrained project scheduling with activity splitting and setup times. *Computers & Operations Research*, 109 :230–249.

Vanhoucke, M. and Debels, D. (2008). The impact of various activity assumptions on the lead time and resource utilization of resource-constrained projects. *Computers & Industrial Engineering*, 54(1) :140–154.

Veldhuizen, D. A. V. and Lamont, G. B. (2000). Multiobjective evolutionary algorithms : Analyzing the state-of-the-art. *Evolutionary Computation*, 8(2) :125–147.

Walter, M. (2015). *Multi-Project Management with a Multi-Skilled Workforce*. Springer Gabler, Wiesbaden.

Walter, M. and Zimmermann, J. (2016). Minimizing average project team size given multi-skilled workers with heterogeneous skill levels. *Computers & Operations Research*, 70 :163–179.

Wan, G., Vakati, S. R., Leung, J. Y.-T., and Pinedo, M. (2010). Scheduling two agents with controllable processing times. *European Journal of Operational Research*, 205(3) :528–539.

Wang, D. J., Kang, C. C., Shiau, Y. R., Wu, C. C., and Hsu, P. H. (2017a). A two-agent single-machine scheduling problem with late work criteria. *Soft Computing*, 21(8) :2015–2033.

Wang, D. J., Yin, Y., Wu, W. H., Wu, W. H., Wu, C. C., and Hsu, P. H. (2017b). A two-agent single-machine scheduling problem to minimize the total cost with release dates. *Soft Computing*, 21(3) :805–816.

Wang, D.-J., Yin, Y., Xu, J., Wu, W.-H., Cheng, S.-R., and Wu, C.-C. (2015). Some due date determination scheduling problems with two agents on a single machine. *International Journal of Production Economics*, 168 :81–90.

Wang, J.-Q., Fan, G.-Q., Zhang, Y., Zhang, C.-W., and Leung, J. Y.-T. (2017c). Two-agent scheduling on a single parallel-batching machine with equal processing time and non-identical job sizes. *European Journal of Operational Research*, 258(2) :478–490.

Wang, L. and long Zheng, X. (2018). A knowledge-guided multi-objective fruit fly optimization algorithm for the multi-skill resource constrained project scheduling problem. *Swarm and Evolutionary Computation*, 38 :54–63.

Wang, Y., He, Z., Kerkhove, L., and Vanhoucke, M. (2017d). On the performance of priority rules for the stochastic resource constrained multi-project scheduling problem. *Computers & Industrial Engineering*, 114 :223–234.

Wauters, T., Kinable, J., Smet, P., Vancroonenburg, W., Vanden, G., and Verstichel, J. (2016). The multi-mode resource-constrained multi-project scheduling problem. *Journal of Scheduling*, 19 :271–283.

Wu, C.-C., Wu, W.-H., Chen, J.-C., Yin, Y., and Wu, W.-H. (2013a). A study of the single-machine two-agent scheduling problem with release times. *Applied Soft Computing*, 13(2) :998–1006.

Wu, M. C. and Sun, S. H. (2006). A project scheduling and staff assignment model considering learning effect. *The International Journal of Advanced Manufacturing Technology*, 28 :1190–1195.

Wu, W.-H., Xu, J., Wu, W.-H., Yin, Y., Cheng, I.-F., and Wu, C.-C. (2013b). A tabu method for a two-agent single-machine scheduling with deterioration jobs. *Computers & Operations Research*, 40(8) :2116–2127.

Xiao, J., Ao, X.-T., and Tang, Y. (2013). Solving software project scheduling problems with ant colony optimization. *Computers & Operations Research*, 40(1) :33–46.

Yannibelli, V. and Amandi, A. (2013). Hybridizing a multi-objective simulated annealing algorithm with a multi-objective evolutionary algorithm to solve a multi-objective project scheduling problem. *Expert Systems with Applications*, 40(7) :2421–2434.

Yin, Y., Chen, Y., Qin, K., and Wang, D. (2019). Two-agent scheduling on unrelated parallel machines with total completion time and weighted number of tardy jobs criteria. *Journal of Scheduling*, 22(3) :315–333.

Yin, Y., Cheng, S.-R., Cheng, T., Wang, D.-J., and Wu, C.-C. (2016a). Just-in-time scheduling with two competing agents on unrelated parallel machines. *Omega*, 63 :41–47.

Yin, Y., Cheng, S.-R., Cheng, T., Wu, C.-C., and Wu, W.-H. (2012a). Two-agent single-machine scheduling with assignable due dates. *Applied Mathematics and Computation*, 219(4) :1674–1685.

Yin, Y., Cheng, S.-R., Cheng, T., Wu, W.-H., and Wu, C.-C. (2013a). Two-agent single-machine scheduling with release times and deadlines. *International Journal of Shipping and Transport Logistics*, 5(1) :75–94.

Yin, Y., Cheng, S.-R., and Wu, C.-C. (2012b). Scheduling problems with two agents and a linear non-increasing deterioration to minimize earliness penalties. *Information Sciences*, 189 :282–292.

Yin, Y., Cheng, T., Hsu, C.-J., and Wu, C.-C. (2013b). Single-machine batch delivery scheduling with an assignable common due window. *Omega*, 41(2) :216–225. Management science and environmental issues.

Yin, Y., Cheng, T., Wan, L., Wu, C.-C., and Liu, J. (2015). Two-agent single-machine scheduling with deteriorating jobs. *Computers & Industrial Engineering*, 81 :177–185.

Yin, Y., Li, D., Wang, D., and Cheng, T. C. (2021). Single-machine serial-batch delivery scheduling with two competing agents and due date assignment. *Annals of Operations Research*, 298(1-2) :497–523.

Yin, Y., Wang, Y., Cheng, T., Wang, D.-J., and Wu, C.-C. (2016b). Two-agent single-machine scheduling to minimize the batch delivery cost. *Computers & Industrial Engineering*, 92 :16–30.

Yin, Y., Wu, W.-H., Cheng, S.-R., and Wu, C.-C. (2012c). An investigation on a two-agent single-machine scheduling problem with unequal release dates. *Computers & Operations Research*, 39(12) :3062–3073.

Yu, F., Wen, P., and Yi, S. (2018). A multi-agent scheduling problem for two identical parallel machines to minimize total tardiness time and makespan. *Advances in Mechanical Engineering*, 10(2) :1687814018756103.

yu Zheng, H., Wang, L., and long Zheng, X. (2017). Teaching–learning-based optimization algorithm for multi-skillresource constrained project scheduling problem. *Soft Computing*, 21 :1537–1548.

Zhang, X. (2021). Two competitive agents to minimize the weighted total late work and the total completion time. *Applied Mathematics and Computation*, 406 :126286.

Zhang, X. and Wang, Y. (2017). Two-agent scheduling problems on a single-machine to minimize the total weighted late work. *Journal of Combinatorial Optimization*, 33(3) :945–955.

Zhao, K. and Lu, X. (2013). Approximation schemes for two-agent scheduling on parallel machines. *Theoretical Computer Science*, 468 :114–121.

Zhao, K. and Lu, X. (2016). Two approximation algorithms for two-agent scheduling on parallel machines to minimize makespan. *Journal of Combinatorial Optimization*, 31(1) :260–278.

Zhu, J., Li, X., and hen, S. (2011). Effective genetic algorithm for resource-constrained project scheduling with limited preemptions. *International Journal of Machine Learning and Cybernetics*, 2 :55–65.

Zhu, L., Lin, J., Li, Y.-Y., and Wang, Z.-J. (2021). A decomposition-based multi-objective genetic programming hyper-heuristic approach for the multi-skill resource constrained project scheduling problem. *Knowledge-Based Systems*, 225 :107099.

Zhu, L., Lin, J., and Wang, Z.-J. (2019). A discrete oppositional multi-verse optimization algorithm for multi-skill resource constrained project scheduling problem. *Applied Soft Computing*, 85 :105805.

Zitzler, E., Deb, K., and Thiele, L. (2000). Comparison of multiobjective evolutionary algorithms : empirical results. *Evolutionary computation*, 8(2) :173–195.

Zitzler, E., Knowles, J., and Thiele, L. (2008). Quality assessment of pareto set approximations. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 5252 LNCS :373–404.

Zitzler, E. and Thiele, L. (1998). Multiobjective optimization using evolutionary algorithms — a comparative case study. In Eiben, A. E., Bäck, T., Schoenauer, M., and Schwefel, H.-P., editors, *Parallel Problem Solving from Nature — PPSN V*, pages 292–301, Berlin, Heidelberg. Springer Berlin Heidelberg.

Résumé

Dans cette thèse, nous considérons un ensemble de problèmes d'ordonnancement multi-projet à contraintes de ressources multi-compétences limitées. Les problèmes étudiés sont issus d'un cas industriel rencontré par une entreprise de développement de services informatiques. Dans chacun de ces problèmes, de multiples projets doivent être exécutés simultanément et achevés dans un horizon de planification fixe. Chaque projet est décomposé en un ensemble de tâches préemptives avec des dates de début au plus tôt et de fin souhaitées associées aux tâches. Chaque tâche doit être exécutée par une seule ressources possédant plusieurs compétences et un niveau d'efficacité par compétence. En outre, la durée de la tâche peut être réduite en fonction du niveau d'efficacité de la ressource affectée à cette tâche. Les ressources sont un ensemble d'employés multi-compétences de disponibilité limitée. Pour chaque employé, nous avons une disponibilité hebdomadaire connue au début de l'ordonnancement et qui n'est pas sujette au changement. De plus, un employé peut intervenir dans plusieurs projets avec un taux de participation maximum (quotité) par projet.

Deux variantes de ce modèle sont abordées : une variante à un seul agent avec un seul critère à optimiser ; une variante avec plusieurs agents où chacun cherche à optimiser son critère. Dans la première variante, les quotités des employés sur les projets sont fixées par le décideur. Ainsi, les disponibilités hebdomadaires des employés, sont réparties entre les différents projets au prorata de leurs quotités sur les projets. L'objectif est d'affecter les employés aux tâches des projets de manière à minimiser la somme des retards pondérés des tâches plus la somme pondérée des violations de certaines contraintes. Pour une résolution à l'optimum, nous présentons un modèle mathématique basé sur la programmation par objectif mixte en nombres entiers. Pour résoudre des instances de grandes tailles, nous proposons une heuristique gloutonne, une recherche locale et une recherche tabou. Nous comparons les performances de ces méthodes par rapport au modèle mathématique en utilisant des instances dérivées de données réelles données par l'entreprise partenaire.

Dans la seconde variante, nous étendons le modèle précédent pour inclure le cas où les quotités des employés sur les projets ne sont pas fixées. Nous considérons que ces quotités doivent être déterminées par le calcul d'ordonnancement, en faisant intervenir le critère de chaque agent. Deux problèmes d'ordonnancement à deux ou trois agents ont été abordés. Le premier problème est de type optimisation sous contrainte avec deux agents en compétition. Le deuxième problème s'agit d'un problème d'ordonnancement à deux agents avec une fonction objectif globale (agent global). Il ajoute au premier problème le faite que certaines contraintes peuvent être violées lorsqu'il n'est pas possible de trouver une solution réalisable pour le premier problème ou pour diminuer le coût du retard. Ainsi, un objectif global consistant à minimiser la violation de ces contraintes est ajouté. Afin de résoudre les problèmes étudies, nous présentons deux modèles mathématiques basés sur la programmation linéaire en nombres entiers (PLNE). Ces modèles sont utilisés chacun par l'approche ε-contrainte pour déterminer les fronts de Pareto exacts. Cependant, cette méthode n'est efficace que pour résoudre les instances de petites tailles. Pour résoudre les instances de grandes tailles, des approches heuristiques utilisant une méthode génétique de type NSGA-II sont développées. Ces différentes approches ont été comparées en termes de qualité de solution sur des instances de taille réelle.

Mots-clefs : ordonnancement multi-projet, ressources multi-compétences, ordonnancement à deux agents, algorithme génétique ; recherche tabou, recherche locale, GPEN, PLNE

Abstract

In this thesis, we consider a set of multi-project scheduling and multi-skilled employees assignment problem with hard and soft constraints. The studied problems stem from an industrial case in an IT company. In each of those problems, multiple projects must be executed simultaneously and completed within a fixed planning horizon. Each project is broken down into a set of preemptive tasks with release and due dates associated with the tasks. We consider that every task needs exactly one skill and must be performed by one employee who has the corresponding skill with an efficiency level per skill. Thus, the processing time of the task may be reduced according to the efficiency level of the employee assigned to this task. Every employee has an availability per week (a working time) known in advance. Also, an employee may be involved in more than one project at the same time, with a maximum quota, i.e. percentage of his/her time, allotted to each project.

Two variants of this model are tackled : a variant with one agent and one criterion to optimize ; a variant with several agents where each agent desires to optimize his own criterion. In the first variant, the decision-maker fixes the quotas of the employees on the projects. Thus, the weekly availabilities of the employees are shared between the different projects in proportion to their quota on the projects. The objective is to assign employees to project tasks such that the total weighted tardiness of tasks and the undesirable goal deviations are minimized. For this problem, we present a mixed-integer goal programming (MIGP) formulation to produce an optimal schedule. Furthermore, a local search algorithm and a tabu search algorithm are proposed to tackle large-scale instances. We compare the performance of the heuristic algorithms against the corresponding MIGP with simulated instances derived from real-world instances got from the partner company.

In the second variant, we extend the previous model to include the case where the quotas of employees on projects are not known. We consider that these quotas must be determined by the scheduling computation, taking into account the agents' critaria. Each agent manages one or several projects and competes with another agent for the use of common multi-skilled employees. We address two scheduling problems with two or three agents and multi-skill constrained-resources. In the first problem, the objective is to minimize the criterion of each compiting agent. The second problem is a two compiting agent scheduling problem with a global objective function (global agent). It also considers that some constraints can be violated when there is no feasible schedule for the first problem or to reduce the cost of delay of each agent. Thus, we further consider a global objective function that minimizes the soft constraint violations. The objectives of each agent is to minimize the total weighted tardiness of its tasks. The overall objective is to find a schedule that minimizes both agents objective functions and the global objective function. In each of these problems. For these problems, we provide two mathematical models based on mixed integer linear programming (MILP). These models are each used by the ε-constraint method to determine the pareto-optimal frontier. However, this method is only effective for solving small instances. To solve large instances, we provide two hybrid heuristics based on a genetic method of type NSGA-II algorithm. We compare the performance of the hybrid heuristics with simulated instances derived from real-world instances.