Etude et résolution de problèmes d'ordonnancement de projets multi-compétences : intégration à un ERP libre

Cheikh Dhib

UNIVERSITÉ FRANÇOIS RABELAIS DE TOURS

École Doctorale MIPTIS

Laboratoire d'Informatique (EA 6300)
Équipe Ordonnancement et Conduite (ERL CNRS 6305)

THÈSE présentée par :

Cheikh Mohamed DHIB

soutenue le : 8 avril 2013

pour obtenir le grade de : Docteur de l'université François Rabelais de Tours

Discipline/ Spécialité : Informatique

Etude et résolution de problèmes d'ordonnancement de projets multi-compétences : intégration à un ERP libre

THÈSE DIRIGÉE PAR :

Néron Emmanuel	Professeur des Universités, Université François Rabelais de Tours
Soukhal Ameur	Maître de Conférences, HdR, Université François Rabelais de Tours

RAPPORTEURS :

Oulamara Ammar	Professeur des Universités, Université de Lorraine
Siarry Patrick	Professeur des Universités, Université Paris-Est Créteil

JURY :

Bellenguez-Morineau Odile	Chargée de Recherche, École de Mines de Nantes
Kovalyov Mikhail	Professeur des Universités, Belarusian State University of Minsk, Bélarus
Néron Emmanuel	Professeur des Universités , Université François Rabelais de Tours
Oulamara Ammar	Professeur des Universités , Université de Lorraine
Péridy Laurent	Professeur des Universités, Université Catholique de l'Ouest
Siarry Patrick	Professeur des Universités, Université Paris-Est Créteil
Soukhal Ameur	Maître de Conférences, HdR, Université François Rabelais de Tours
Heintz Olivier	Directeur des projets à Néréide, membre invité

Remerciements

Mes remerciements vont, en premier lieu, à M. Emmanuel Néron qui, par ses qualités scientifiques et humaines uniques, m'a encouragé à mener cette étude jusqu'au bout. Je profite de ces lignes pour lui exprimer ma gratitude éternelle.

Mes remerciements vont également à M. Ameur Soukhal à qui je dois beaucoup, grâce à son encadrement, ses idées et ses conseils qui m'ont été très utiles.

Je souhaite également remercier vivement M. Patrick Siarry et M. Ammar Oulama pour avoir accepté de rapporter ma thèse et surtout pour leurs remarques et conseils très pertinents.

Mes remerciements, vont également à M. Laurent Péridy qui m'a honoré d'avoir présidé le jury de ma soutenance.

Je remercie chaleureusement Mme Odile Bellenguez d'avoir accepté d'être dans le comité de suivi de ma thèse et ensuite membre du jury. Ses remarques et ses questions et explications pertinentes m'ont beaucoup aidé dans mes recherches. Je lui exprime toute ma gratitude.

J'adresse également mes remerciements et ma gratitude au professeur Mikael Kovalyov qui a accepté de participer au jury de cette thèse.

Je remercie sincèrement M. Olivier Hientz pour son investissement dans ce travail et ses remarques pendant ces 3 ans.

Mes remerciements vont bien évidement à la société Néréide qui m'a offert la chance de faire cette étude et de profiter des compétences de ses équipes pour découvrir plusieurs domaines aussi bien techniques que fonctionnels. J'en profite pour leur exprimer ma profonde gratitude.

Je remercie profondément, M. Jean-Charles Billaut, non seulement parce qu'il m'a accueilli dans son laboratoire, mais aussi pour ses conseils et sa disponibilité.

Je ne peux que remercier Mme Christelle Grange et Mme Béatrice Pawlik du secrétariat du Laboratoire d'Informatique (LI) pour leur sympathie et leur bonne humeur, et surtout pour le travail qu'elles mènent dans l'ombre avec efficacité.

Enfin, je tiens à remercier ma famille ainsi que tous mes amis pour leur soutien et leurs encouragements.

Résumé

Les travaux de cette thèse réalisée sous contrat CIFRE portent sur des problématiques d'ordonnancement de projets multi-compétences. Définis en collaboration avec des experts de gestion de projet au sein de la société Néréide, deux modèles d'ordonnancement de projet font l'objet de cette étude.

Dans le premier modèle, une tâche est définie par l'ensemble des compétences dont elle a besoin, la charge nécessaire de chaque compétence ainsi que la possibilité d'être interrompue ou non. Pour l'élaboration d'un planning prédictif respectant toutes les contraintes et minimisant la date de fin du projet, nous proposons des heuristiques de liste et métaheuristiques. Un modèle mathématique linéaire en nombres entiers ainsi que des bornes inférieures sont également développés. Dans un second temps, nous proposons, à partir d'un planning prédéfini, des méthodes pour ajuster le planning et répondre aux aléas survenus lors du déroulement du projet. Pour résoudre ce problème réactif, nous proposons une approche exacte itérative basée sur une formulation linéaire en nombres entiers ainsi qu'un algorithme génétique de type NSGA-II. Il s'agit donc d'une approche réactive bicritère où les solutions calculées doivent minimiser à la fois la date d'achèvement du projet et le nombre maximum de changements d'affectation de tâches aux employés.

Dans le deuxième modèle, un cas particulier du modèle préemptif précédent est étudié. Nous nous intéressons au cas où une tâche nécessite une seule compétence avec possibilité de préemption seulement si les ressources ne sont pas disponibles (absence, congés, etc.). Dans ce modèle, une tâche est définie également par sa date de disponibilité et une date de fin souhaitée. Un coût d'utilisation personne/compétence est introduit. Pour ce dernier modèle, il s'agit d'un problème d'ordonnancement de projet bicritère, pour lequel les solutions calculées doivent minimiser le retard maximum et le coût global d'affectation des personnes aux tâches. Des heuristiques et métaheuristiques sont proposées pour ce modèle.

Certaines méthodes de résolution proposées ont été implémentées sous forme d'*add-ons* intégrables au *framework OFBiz*.

Mots clés : Recherche Opérationnelle, Optimisation, Ordonnancement de projets, Contraintes de ressources humaines, Compétences, Prédictif et Réactif, Bornes inférieures, Modèles mathématiques, Heuristiques, Métaheuristiques.

Abstract

The work presented in this thesis deals with multi-skill project scheduling problems. We have studied two models of project scheduling which are defined in collaboration with project management experts in Néréide company.

In the first model, a task is defined by a set of required skills, the load needed for each skill as well as the possibility of preemption. To build a predictive planning which respects all problem constraints and minimize the project completion time (makespan), we propose heuristics and meta-heuristics methods. A mixed integer mathematical linear programming model and lower bounds are also proposed. From a predefined planning, we propose an exact method based on a mathematical program as well as a genetic algorithm of type NSGA-II allowing to deal with disruptions occurred during the project realization. It is, therefore, a reactive approach in which we look for feasible solutions minimizing both the project completion date and the maximum number of resources assignment changes.

In the second studied model, we focus on a case where a task exactly requires one skill with preemption possibility only in case of resources unavailability. In this model, a task is also characterized by its release and due date. A cost per person/skill is given. It is, therefore, a bi-objective problem in which the computed solutions must minimize both the maximum tardiness and the project global cost. Heuristics and meta-heuristics are proposed for solving this problem.

Some proposed methods are integrated in the framework OFBiz as add-ons.

Keywords : Operations research, optimization, project scheduling, human resource constraints, Skills, Predictive and Reactive, lower bounds, Mathematical Models, Heuristics, Metaheuristics.

Table des matières

Introduction

La préoccupation primordiale d'un gestionnaire industriel ou fournisseur de services est de tirer le maximum de son système de production de biens ou de services. Un chef d'atelier, comme un chef de projet, cherche la meilleure organisation et donc la meilleure efficacité relativement à l'utilisation des ressources dont il dispose. Face à la complexité des systèmes et la concurrence accrue du marché, le besoin d'outils et méthodes d'optimisation et d'aide à la décision est de plus en plus considérable. Les méthodes d'optimisation et outils de la Recherche Opérationnelle ont montré leur efficacité pour accompagner le décideur dans sa quête de réponses aux exigences de ses clients. En effet, la Recherche Opérationnelle est constituée d'un ensemble de méthodes et d'outils scientifiques pour modéliser les problèmes de décision et fournir une ou plusieurs solutions afin d'aider le décideur.
L'ordonnancement de projets est une classe des problèmes d'optimisation combinatoire traitée par la Recherche Opérationnelle. Cette classe de problèmes est largement étudiée dans la littérature et représente un challenge permanent pour tout gestionnaire industriel et chercheur quant au développement de méthodes de résolution permettant de gérer au mieux ses ressources et atteindre ses objectifs (satisfaire ses clients, respecter au mieux le choix de ses employés, faire face à la concurrence du marché, etc.).

La société *Néréide*, dont l'activité principale est le développement, l'intégration et le consulting centrée sur un *ERP*[1] libre (*OFBiz*) n'échappe pas à cette règle, et ce à double titre : pour la gestion de ses propres projets, mais aussi et surtout pour la mise en place d'un outil intégré à *OFBiz*, devant à terme permettre à ses clients de bénéficier d'un outil de planification de projet. Dans les deux cas, disposant d'une équipe de développeurs et consultants avec des compétences diverses et travaillant principalement en mode projet, le gestionnaire doit gérer de façon efficace les ressources humaines pour garantir le bon déroulement de ses projets. Trois critères stratégiques doivent être parallèlement pris en compte : le temps, la qualité et le coût. En outre, la réalisation d'un projet est soumise à de nombreuses contraintes (les compétences, la disponibilité, les délais de livraison-finaux ou intermédiaires, etc.). C'est dans ce cadre et sous convention *CIFRE* avec la société Néréide que les travaux présentés dans cette thèse ont été réalisés. Il s'agit de problèmes d'ordonnancement de projets connexes à ceux de l'ordonnancement de projet à moyens limités, dit *RCPSP* (*Resource Constrained Project Scheduling Problem*) [Artigues *et al.*, 2008], et plus généraux que le problème de gestion de projet multi-compétences, dit *MSPSP* (*Multi-Skill Project Scheduling Problem*) [Bellenguez-Morineau, 2006].

1. *Enterprise Resource Planning* ; Progiciel de Gestion Intégré.

Comme dans le *RCPSP*, le projet est découpé en tâches (activités). Les tâches sont liées entre elles par des relations de précédence classiques de type fin-début, i.e. une tâche ne peut commencer que si celles qui la précèdent sont finies. Chaque tâche possède une durée opératoire et nécessite pour son exécution une quantité constante d'une ou de plusieurs ressources tout au long de sa réalisation. Les ressources sont disponibles en quantité constante pendant l'horizon de planification du projet. Contrairement au cas classique du RCPSP, pour le problème identifié dans la société Néréide, les tâches sont caractérisées par leurs besoins en compétences et chaque ressource (personne) ne peut traiter qu'une seule tâche à la fois, c'est-à-dire une personne ne peut exercer qu'une seule compétence à une date donnée. Les besoins en compétences de chacune des tâches sont exprimés en charge nécessaire tout au long de sa réalisation. Enfin la disponibilité de chaque personne (ressource) est connue tout au long de l'horizon de planification. Il s'agit donc d'un problème de gestion de projets multi-compétences avec la particularité que certaines tâches peuvent être préemptées après avoir été commencées. Nous traitons un problème de **gestion de projet multi-compétences** avec **préemption** des tâches et **synchronisation** de compétences.

L'objectif considéré dans la version standard du RCPSP comme pour le MSPSP est la minimisation de la durée totale du projet, *makespan*. Il est à noter que ces problèmes sont $\mathcal{NP}$-difficiles au sens fort. Même si la séquence d'ordonnancement des tâches est donnée, vérifier la faisabilité de la solution a été également prouvé $\mathcal{NP}$-difficile. La résolution de ces problèmes suscite donc un intérêt réel dans la communauté de la Recherche Opérationnelle, où de très nombreux travaux leur ont été consacrés [Hartmann et Briskorn, 2010].

Malgré l'existence de plusieurs modèles de gestion de projet multi-compétences [Montoya *et al.*, 2010], [Correia *et al.*, 2010], [Valls *et al.*, 2009], [Bellenguez-Morineau, 2006], [Xiao *et al.*, 2013], ces modèles peuvent ne pas être suffisants dans un contexte industriel. Très généraux, ces modèles nécessitent d'être complétés par quelques contraintes spécifiques, telles que la synchronisation des tâches, la possibilité de préempter une tâche et la prise en considération des dates d'arrivée et dates de fin souhaitée des tâches. Certes, ces contraintes spécifiques répondent à un besoin particulier de la société Néréide, mais peuvent être rencontrées dans toutes sociétés de type SSII et plus généralement de production de services. Il s'agit donc bien de développer des outils et méthodes pour la résolution de problèmes de gestion de projets multi-compétences généraux. L'apport principal de notre étude est de proposer des modèles de gestion de projet multi-compétences rencontrés dans un contexte industriel (Société Néréide). Notre étude concerne à la fois la proposition de solutions de base efficaces, mais aussi l'élaboration de nouveaux plannings, après que des aléas soient survenus lors de la mise en œuvre d'un planning en cours.

Définis en collaboration avec des experts de gestion de projet au sein de la société Néréide, deux modèles d'ordonnancement de projet font l'objet de cette étude.

- Dans le premier modèle, une tâche est définie par l'ensemble des compétences dont elle a besoin, la charge nécessaire de chaque compétence, ainsi que la possibilité d'être interrompue ou non. Dans un premier temps, il s'agit de proposer des **méthodes de**

résolution exactes ou approchées pour l'élaboration d'un planning **prédictif** respectant toutes les contraintes et minimisant la date de fin du projet. Dans un second temps, des **méthodes réactives** de résolution exactes ou approchées sont développées pour ajuster le planning prédictif répondant aux aléas survenus lors du déroulement du projet. Il s'agit donc d'une approche **réactive bicritère** où les solutions calculées doivent minimiser à la fois la date d'achèvement du projet et le nombre maximum de changements d'affectation de tâches aux employés.

– Dans le deuxième modèle étudié, une tâche nécessite une seule compétence. Une tâche ne peut être préemptée que si la ressource allouée n'est pas disponible. Dans ce modèle, une tâche est définie également par sa date de disponibilité et une date de fin souhaitée. Un **coût d'utilisation** personne/compétence est introduit. Il s'agit de proposer des méthodes de résolution exactes ou approchées pour l'élaboration d'un planning prédictif respectant toutes les contraintes pour résoudre ce **problème d'ordonnancement de projet bicritère**, où les deux objectifs à minimiser sont le **retard maximum** et le **coût global** d'affectation des personnes aux tâches.

Les travaux effectués pendant cette thèse seront présentés à travers cinq chapitres structurés comme suit. Dans le chapitre I, nous présentons le contexte et la motivation de cette thèse. Ainsi, nous décrivons en détail les deux modèles de gestion de projets sur lesquels notre étude a été menée. Nous insistons sur les contraintes spécifiques aux modèles traités, à savoir la préemption des tâches, la synchronisation des compétences au début et l'expression des besoins des tâches par compétence. Une définition formelle de chacun de ces modèles sera donnée. Les liens avec les problèmes d'ordonnancement de projets connus de la littérature seront exposés.

Nous présentons au chapitre II un état de l'art centré sur les problèmes connexes à ceux décrits et étudiés dans cette thèse. Ainsi, nous concentrons cette étude sur quelques méthodes et outils correspondant aux objectifs de notre travail. Notamment, nous évoquons les problèmes d'ordonnancement de projet multi-compétences et la prise en compte de la préemption dans la littérature pour le *RCPSP*. Nous nous intéressons aussi à la prise en compte de l'incertitude dans les problèmes d'ordonnancement, notamment les problèmes d'ordonnancement de projet réactifs.

Le chapitre III est consacré à l'étude du premier modèle que nous avons traité. La spécificité de ce modèle est la prise en compte de **préemption de certaines tâches**, la définition de la **charge nécessaire par compétence** et la **synchronisation** des compétences de chaque tâche, i.e. les compétences de chaque tâche doivent commencer à la même date. Un programme linéaire à variables mixtes est proposé. Des résultats expérimentaux de ce *PL* sont présentés. Dans un deuxième temps, nous nous sommes intéressés au développement de bornes inférieures. Les résultats expérimentaux mesurant les performances de ces bornes inférieures par rapport à la méthode exacte sont également présentés dans ce chapitre. Comme méthodes de résolution approchées, des heuristiques de listes basées sur des règles de priorité et des métaheuristiques (recherche tabou et algorithmes génétiques) sont développées. Les résultats expérimentaux mettant en évidence les per-

formances de ces méthodes de résolution sont par la suite présentés. La dernière partie de ce chapitre est dédiée à l'étude d'un modèle **réactif bicritère** du problème MSPSP préemptif. Il s'agit de prendre en considération les aléas survenus en cours d'exécution du projet. En plus du critère d'optimisation initial qui a permis de calculer le planning prédictif (C_{max}), un deuxième critère visant à minimiser le nombre maximum de changements d'affectations par personne est introduit. Un algorithme génétique de type $NSGA-II$ est proposé pour trouver une approximation de l'ensemble des solutions non dominées. A la fin de ce chapitre, une étude expérimentale de performance de cet algorithme est réalisée en utilisant la métrique d'hypervolume.

Les différents travaux menés sur ce problème ont fait l'objet des communications suivantes : [Dhib *et al.*, 2010], [Dhib *et al.*, 2011c], [Dhib *et al.*, 2011b], [Dhib *et al.*, 2011a].

Dans le chapitre IV, nous étudions le deuxième problème d'ordonnancement de projet multi-compétences identifié. Dans ce modèle, la **préemption** est considérée uniquement lorsque les ressources sont indisponibles (absence, congés, etc.). Les tâches sont également définies par des dates de disponibilité ainsi que des dates de fin souhaitée. Un aspect financier est considéré afin de minimiser le coût de réalisation d'un projet. Ce coût d'affectation des personnes aux compétences peut également représenter la qualité du planning en tenant compte des préférences des personnes. Deux critères sont à minimiser à la fois : le **retard maximum** des tâches et le **coût global du projet**. Un modèle mathématique est présenté. Par la suite, des méthodes de résolution approchées de types heuristiques de liste basées sur des règles de priorité sont proposées pour résoudre ce problème en optimisant le coût d'affectation global avec un retard maximum (T_{max}) inférieur ou égal à une valeur Q. Un algorithme génétique de type $NSGA-II$, ainsi qu'une recherche tabou énumérant le front de Pareto itérativement en utilisant l'approche $\epsilon-$contrainte, sont développés pour approcher le front de Pareto. Un générateur de données est développé pour étudier expérimentalement la performance des méthodes de résolution proposées. Les résultats expérimentaux ainsi que les tests de calibrage des heuristiques proposées clôturent ce chapitre.

Les différents travaux menés sur ce problème ont fait l'objet des communications suivantes : [Dhib *et al.*, 2012a], [Dhib *et al.*, 2012b].

Dans le chapitre V, nous présentons la partie logicielle intégrée à *OFBiz*. L'objectif est une mise à disposition des outils présentés dans le cadre industriel d'un *ERP* libre. Nous présentons les techniques et outils utilisés, les *add-ons*[2] qui ont été développés.

Enfin, nous clôturons ce manuscrit par une conclusion générale, dans laquelle nous rappelons les résultats obtenus lors de ces travaux ainsi que les perspectives de recherches envisageables pour la continuité de ces travaux.

2. *add-ons* : greffons ou *Plug-ins*.

Chapitre 1

Présentation du problème et contexte

Dans ce chapitre, nous définissons d'abord le contexte et les motivations de ce travail (section 1.1). Une fois le contexte défini, nous décrivons en détail, à l'aide d'exemples, les problèmes sur lesquels cette étude a été menée (section 1.3). Nous montrons aussi les similarités et les différences par rapport aux problèmes connexes de gestion de projet que l'on trouve dans la littérature. Les notations utilisées tout au long de ce manuscrit seront définies dans les tableaux 1.1 et 1.2

1.1 Contexte du travail

Les travaux de cette thèse font l'objet d'une convention *CIFRE*[1] qui s'inscrit dans le cadre d'une collaboration entre la société *Néréide* et le Laboratoire d'Informatique de l'université de Tours (LI, EA 6300, Équipe Ordonnancement et Conduite (ERL CNRS 6305)).

Néréide (Veretz, 37, http ://www.nereide.biz) est une SSII, intégrateur de Progiciel de Gestion Intégré en logiciel libre. Comme pour toute société proposant des solutions du monde libre, le coût de la licence de l'outil est nul. L'activité de la société consiste principalement en des prestations de services (création de nouveaux modules, formations, conseil, etc.).

Le logiciel auquel la société Néréide contribue principalement est *OFBiz* (*Apache Open for Business*) et ses extensions, qui couvrent la gestion financière, la collaboration logistique (planification de la production et distribution), le *CRM* (*Customer Resource Management*) comprenant le marketing et la gestion des ventes et du service après-vente, ainsi que le commerce électronique et la gestion des points de vente.

Il ressort des demandes des clients, outre la mise en place d'un *ERP* (*Entreprise Resource Planning*) et des modules spécifiques, eg., *GPAO* (*Gestion de Production Assistée*

1. Conventions Industrielles de Formation par la Recherche

par Ordinateur), un besoin de plus en plus important d'outil de planification et de suivi des activités effectuées en mode projet. Cette problématique se retrouve chez des clients travaillant par exemple comme intégrateurs ou installateurs, dont l'activité se prête particulièrement au mode projet.

1.2 Définitions et terminologie

Afin que ce manuscrit soit accessible au lecteur intéressé non issu de la communauté de l'ordonnancement, nous allons donner les définitions de quelques termes utilisés à différents endroits de ce document. Bien que ces termes soient utilisés aussi bien en ordonnancement de projet que dans d'autres problèmes d'ordonnancement, nous ferons le plus souvent référence aux problèmes d'ordonnancement de projet. Nous renvoyons à [Blazewicz *et al.*, 1983] et [Artigues *et al.*, 2008], par exemple, pour une description complète de ces notions.

Tâche : La tâche est l'entité élémentaire d'un projet. Elle se caractérise par une durée (temps nécessaire à son exécution), des besoins en ressources et des dates d'exécution (ou de réalisation). On utilise ici sans distinction le terme activité ou tâche.

Ressource : Une ressource est un moyen technique ou humain qui est capable de réaliser ou participer à la réalisation d'une tâche. Elle possède généralement une capacité et un calendrier de disponibilité. Plusieurs catégories de ressources ont été identifiées dans la littérature :

- Ressources renouvelables : si la capacité de la ressource sur chaque période de son calendrier ne dépend pas de la quantité consommée avant cette période, la ressource est dite renouvelable. Les ressources humaines sont des exemples de ressources renouvelables.
- Ressources non renouvelables : contrairement aux ressources renouvelables, si la quantité consommée par le passé induit une baisse de la capacité restante de la ressource, elle est dite non renouvelable. Les produits consommables pendant un projet sont un exemple de cette catégorie de ressource.
- Ressources partiellement renouvelables : si la capacité d'une ressource est limitée sur des périodes données de l'horizon, on parle alors d'une ressource partiellement renouvelable. Par exemple, les personnes avec des indisponibilités sur l'horizon de planification.
- Ressources doublement contraintes : ce sont les ressources où la capacité globale est limitée ainsi que la capacité sur des périodes données. On peut citer comme exemple, les ressources budgétaires d'un projet.

Compétence : une compétence modélise l'aptitude d'une ressource (généralement une personne) à réaliser une tâche ou participer à sa réalisation afin de procurer une connaissance ou un savoir-faire dont la tâche a besoin pendant une durée donnée.

Projet : un projet, dans le cadre de cette étude, est un ensemble de tâches liées entre elles par des contraintes de précédence et dont les caractéristiques sont connues (durée, besoins, ressources). Il se caractérise généralement par un horizon de planification et éventuellement une fenêtre de temps (début et fin impérative du projet).

Relations de précédence : dans un projet, les tâches sont en général soumises à des contraintes de séquencement. Ces contraintes visent à lier l'exécution d'une tâche à celle d'une ou plusieurs autres tâches. On trouve habituellement dans les problèmes de gestion de projets les types de relations de précédence suivants :

- "fin-début" : si la tâche i précède la tâche j avec ce type de précédence, alors j ne peut pas commencer avant la fin de i ;
- "début-début" : si la tâche i précède la tâche j avec ce type de précédence, alors j ne peut pas commencer avant que i ait commencé ;
- "début-fin" : si la tâche i précède la tâche j avec ce type de précédence, alors i doit commencer avant la fin de j ;
- "fin-fin" : si la tâche i précède la tâche j avec ce type de précédence, alors i doit finir avant j ;
- relations de précédence généralisées : dans les quatre types de relations de précédence ci-dessus, on peut ajouter un délai qui peut être une durée constante ou définie en fonction de la durée de l'une des deux tâches. Par exemple, si deux tâches i et j sont liées entre elles par une relation de précédence de type "fin-début" avec un délai de 2 périodes de temps, alors la tâche j ne peut commencer que 2 périodes de temps après la fin de i. On utilise le terme relation de précédence généralisée pour ce type de contraintes.

Date de disponibilité : dans l'ordonnancement en général, on peut trouver des contraintes sur la date de début d'une tâche. La contrainte la plus utilisée consiste à imposer une date avant laquelle la tâche ne peut pas commencer (livraison de matière première, livrable sous traité par exemple). On utilise le terme date de disponibilité (*release date*) pour définir cette contrainte. On utilise également le terme "date de début au plus tôt". Ce dernier terme est également utilisé lorsqu'on parle des dates de disponibilité déduites à partir des relations de précédence entre les tâches.

Ordonnancement : il existe, dans la littérature, plusieurs définitions du terme "ordonnancement". Nous donnons ici celle de [Carlier et Chrétienne, 1988] :

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

1.3 Problématique

La société Néréide étant un prestataire de services en logiciels libres, son activité se fait en totalité en mode projet. Outre le besoin interne, il arrive que des clients de la société

aient besoin d'un outil de gestion de projet intégrable avec les autres modules d'OFBiz. De ce fait, Néréide a voulu concevoir et mettre en œuvre une ou plusieurs solutions de gestion de projets multi-compétence, qui répondent à ses besoins internes mais aussi aux besoins des clients ou prospects pour lesquels le problème se pose.
Pour répondre à ce besoin, nous avons été amenés à étudier deux modèles de gestion de projets multi-compétences que nous détaillons ci-dessous.

1.3.1 Problème de gestion de projet multi-compétences préemptif avec synchronisation au début

Le problème de gestion de projet multi-compétence préemptif avec synchronisation au début, que nous notons par la suite *PMSPSP* (*Preemptive Multi-Skill Project Scheduling Problem*) peut être défini comme suit.
On dispose d'un projet découpé en n tâches réelles et 2 tâches fictives. L'ensemble des tâches est noté $\mathcal{A} = \{A_0, A_1, \ldots, A_{n+1}\}$ avec A_0 et A_{n+1} les deux tâches fictives qui marquent respectivement le début et la fin du projet. Une équipe de personnes $\mathcal{P} = \{P_1, \ldots, P_M\}$ (M : nombre de personnes) est affectée au projet. Chaque personne possède une ou plusieurs compétences parmi un ensemble de compétences $\mathcal{S} = \{S_1, \ldots, S_K\}$ (K : nombre de compétences). Les tâches ont des besoins en compétence exprimés en nombre de périodes. Chaque tâche A_i a besoin de p_{ik} périodes (charge) de temps de la compétence S_k. Une tâche peut donc avoir besoin de plusieurs compétences, chacune ayant une charge différente, chaque compétence étant assurée par une unique personne. Les personnes ont des créneaux de disponibilité qui définissent les périodes de temps où elles peuvent intervenir sur le projet. Un horizon de planification $\mathcal{T} = \{0, \cdots, HZ\}$ est associé au projet dans lequel la planification devra se faire. On utilise la notation $Dispo_{m,t}$ qui est égale à 1 si la personne P_m est disponible pendant $[t, t+1]$, 0 sinon. $Dispo_{m,t}$ est une donnée du problème. Les tâches sont classées en deux sous-ensembles : les tâches préemptives $\mathcal{A}_{\overline{\mathcal{P}}}$ et les tâches non préemptives ($\mathcal{A} \setminus \mathcal{A}_{\overline{\mathcal{P}}}$). Pour les tâches non préemptives, une fois l'exécution d'une tâche commencée, son interruption n'est plus possible. Contrairement aux tâches non préemptives, l'exécution d'une tâche préemptive peut s'étaler sur plusieurs périodes de temps non consécutives.
Une compétence d'une tâche doit être réalisée par une seule personne. Le changement d'affectation n'est donc pas autorisé. Cela exige que la personne réalise la totalité des compétences qu'elle a commencées, y compris dans le cas préemptif.
Une contrainte de synchronisation au début a été introduite. Elle a pour objectif de forcer le démarrage de toutes les compétences d'une tâche à la même période. Les différentes personnes qui vont travailler sur les compétences d'une tâche doivent être toutes présentes à la première période de temps de travail sur la tâche. Cette contrainte a été identifiée lors de l'analyse du problème industriel et correspond à une étape indispensable de concertation des différentes personnes participant à la tâche, lors de son lancement. Enfin, si la tâche est préemptive, chacun peut interrompre la compétence sur laquelle il ou elle travaille indépendamment des autres compétences, après cette période de synchronisation.

Les deux tâches fictives A_0 et A_{n+1} n'ont pas de besoin en compétence et leur durée sont nulles.
Notons que la notion de durée de tâche comme dans le *RCPSP* et *MSPSP* n'a pas le même

Notation	*Définition*
	Ensembles
n	nombre de tâches réelles du projet
$\mathcal{A} = \{A_i \mid i \in \{0, \ldots, n+1\}\}$	ensemble de tâches ou activités du projet avec A_0, A_{n+1} les deux tâches fictives marquant respectivement le début et la fin du projet
$\mathcal{A}_{\overline{\mathcal{P}}}$	l'ensemble des tâches préemptives
K	nombre de compétences
$\mathcal{S} = \{S_k \mid k \in \{1, \ldots, K\}\}$	ensemble de compétences nécessaires pour la réalisation du projet
M	nombre de personnes affectées au projet
$\mathcal{P} = \{P_m \mid m \in \{1, \ldots, M\}\}$	ensemble des personnes affectées au projet
HZ	horizon de planification (le nombre de périodes maximum autorisé entre le début de planification et la fin du projet)
$\mathcal{T} = \{0, 1, \ldots, \ldots HZ - 1\}$	ensemble de périodes de temps sur lesquelles le projet peut s'exécuter
	Données liées aux tâches
$p_{i,k}$	nombre de périodes dont la tâche A_i a besoin en compétence S_k
$p_i = \max_k p_{i,k}$	durée de la tâche A_i
$\mathcal{G}(\mathcal{A}, E, d)$	graphe de précédence valué avec les durées des tâches
$\{A_i, A_j\} \in E$	si A_j est un successeur direct de A_i
	Données liées aux ressources
$MS_{m,k}$	égal à 1 si P_m maîtrise S_k, sinon égal à 0
$Dispo(m,t)$	égal à 1 si P_m est disponible sur $[t, t+1]$, 0 sinon
$MS_{m,k,t}$	égal à 1 si P_m est disponible sur $[t, t+1]$ et maîtrise S_k, 0 sinon

Tableau 1.1 – Notations utilisées pour le problème *PMSPSP*

sens, du fait que les compétences d'une tâche n'ont pas les mêmes durées et les préemptions des compétences d'une tâche ne sont pas nécessairement synchronisées. Malgré ces différences, on emploie la notion de durée p_i (durée de la tâche A_i) que nous définissons comme le besoin en compétences (charge) maximum ($p_i = \max_k p_{ik}\ k \in \{1, \ldots, K\}$).
Les relations de précédence de type "fin-début" sont considérées. Chaque tâche A_i a un ensemble de prédécesseurs $Pred_i$ et un ensemble de successeurs $Succ_i$. Pour deux tâches A_i et A_j telles que A_i précède A_j, l'exécution de A_i doit être terminée afin de pouvoir commencer la tâche A_j.

Un récapitulatif des notations utilisées est donné dans le tableau 1.1.

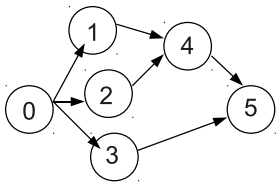
Pour ce problème, nous avons étudié deux cas :

- *Cas prédictif* : il s'agit de trouver un ordonnancement de *base*, à partir des informations dont on dispose avant le commencement du projet. Dans ce cas, l'objectif est de fournir une solution qui respecte toutes les contraintes présentées ci-dessus et dont la durée du projet (C_{max}) est minimale.
- *Cas réactif* : dans la partie 3.5, nous nous intéressons au cas réactif. Pour la société *Néréide*, il est indispensable de pouvoir suivre un projet après son démarrage. Les données sur lesquelles le planning prédictif est constitué peuvent être amenées à changer et une modification du planning peut s'avérer indispensable durant l'exécution du projet. Le détail de ce modèle est présenté dans le chapitre 3 (section 3.5).

Le cas *pro-actif* [Billaut *et al.*, 2010], c'est-à-dire proposant un ordonnancement de base robuste, capable d'absorber certaines modifications des données sans changer sensiblement la structure de la solution, n'a pas été abordé dans cette thèse. Nous estimons que le nombre d'aléas est très important et diversifié dans le cas des projets informatiques. Il est aussi difficile d'obtenir les informations statistiques nécessaires pour élaborer des méthodes robustes. En conséquence, et selon les nombreux travaux menés sur l'aspect de robustesse et flexibilité des ordonnancements, e.g, [de Vonder *et al.*, 2007], [Herroelen et Leus, 2000] et [Billaut *et al.*, 2010], les approches robustes ne seront pas considérées.
Le modèle présenté a été conçu essentiellement pour répondre au problème de gestion de projet auquel la société *Néréide* est confrontée en interne, mais aussi pour pouvoir répondre aux besoins des clients potentiellement intéressés. Néanmoins, il existe des modèles dans la littérature dont notre problème se rapproche. Citons par exemple le problème d'ordonnancement de projet multi-compétence [Bellenguez-Morineau, 2006], le problème d'ordonnancement de projets d'informatique [Xiao *et al.*, 2013]. Les spécificités du problème par rapport à ces modèles résident essentiellement dans la prise en compte de la préemption et de la synchronisation des compétences simultanément, ainsi que la considération des charges par tâche/compétence. A notre connaissance, il n'existe pas de modèles de gestion de projet tenant compte de toutes ces contraintes simultanément. Un état de l'art des problèmes connexes est présenté au Chapitre 2.

1.3.1.1 Exemple

La figure 1.1 illustre les données d'un projet de 6 tâches (0 et 5 sont les deux tâches fictives). Les tâches 1, 2 et 3 n'ont pas de prédécesseur tandis que la tâche 4 est précédée par les deux tâches 1 et 2. Nous considérons une équipe de 2 personnes et 2 compétences nécessaires pour réaliser le projet. Dans le tableau supérieur, on trouve les besoins de chaque tâche en compétences. Par exemple, la tâche 1 nécessite 2 périodes de la compétence 1 et 1 période de la compétence 2. Les tâches 1 et 3 sont préemptives, tandis que 2 et 4 ne le sont pas. La personne P_1 maîtrise les deux compétences et est disponible tout au long de l'horizon du projet, alors que la personne P_2 ne maîtrise que la compétence 1 et n'est pas disponible pendant la première période de temps. La figure 1.2 présente une solution réalisable pour l'exemple de la figure 1.1 de $C_{max} = 8$. Nous remarquons que les tâches 2 et 4 n'ont pas été interrompues après leur démarrage ; quant à la compétence 1 de la tâche 1, une préemption s'est produite à la période 2 où elle a été reprise à la période 5 ; même constat pour la compétence 1 de la tâche 3, qui a été interrompue à la période 1



Préemption : <Y, N, Y, N>

Tâche	Charges
1	<2,1>
2	<3,2>
3	<2,0>
4	<0,2>

Personnes	Compétences	Indisponibilité
1	<Y, Y>	-
2	<Y,N>	[0,1]

FIGURE 1.1 – Exemple du PMSPSP

et reprise à la période 6. Pour les compétences/tâches interrompues, celles-ci sont reprises par la même personne que celle qui les a commencées. La contrainte de synchronisation est respectée, i.e, pour toutes les tâches, les compétences sont exécutées simultanément au moins sur la première période de temps. La personne P_2 n'étant pas disponible pendant la première période, aucune tâche ne lui a été affectée. La tâche 4 n'a pas pu démarrer avant la période 6, malgré la disponibilité des ressources, cela est dû aux contraintes de précédence avec les tâches 1 et 2.

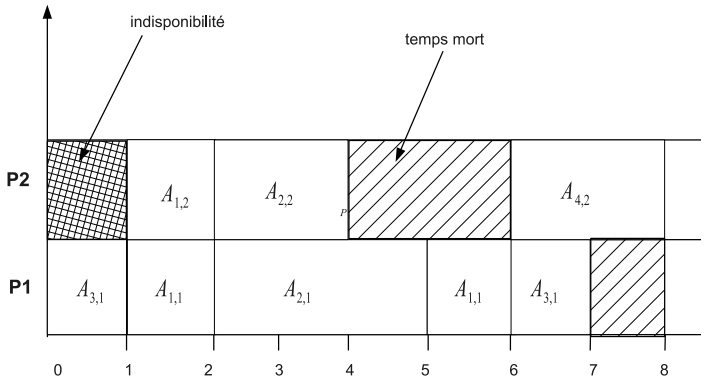


FIGURE 1.2 – Solution réalisable

1.3.1.2 Complexité

Le problème *PMSPSP* est une généralisation du problème d'ordonnancement de projet multi-compétences (*MSPSP*) [Bellenguez-Morineau, 2006]. Toute instance du *MSPSP* peut être réduite en une instance de *PMSPSP* dans laquelle la préemption n'est pas considérée et les compétences de chaque tâche ont la même durée. Nous pouvons alors conclure que

le problème est *NP-Difficile* au sens fort, car le *MSPSP* a été prouvé *NP-Difficile* au sens fort [Bellenguez-Morineau, 2006].

1.3.2 Ordonnancement de projets avec ressources multi-compétences et tâches mono-compétence

Dans un deuxième temps, nous nous sommes intéressés à un second modèle que nous appelons dans le reste de ce document *MSRMST* (*Multi-skilled resource mono-skill task project scheduling problem*). Les différences entre ce modèle et le modèle précédent décrit dans la section 1.3.1 sont les suivantes :

- **Une seule compétence par tâche** : dans le *MSRMST*, une tâche nécessite seulement une seule compétence qui devra être réalisée par une seule personne, alors que dans le *PMSPSP* une tâche peut avoir besoin de plusieurs compétences. Pour chaque tâche A_i, on note $comp_i$ la compétence nécessaire à sa réalisation.
- **Préemption seulement par les périodes d'indisponibilité** : dans le *PMSPSP*, nous avons un ensemble de tâches non préemptives, où chaque tâche doit s'exécuter sans interruption, et un ensemble de tâches pour lesquelles la préemption est autorisée. Cette propriété est remplacée dans ce modèle par l'autorisation de toute tâche à être interrompue si cette interruption est due à une indisponibilité de la personne à qui la tâche est affectée. En revanche, une personne ne peut interrompre une tâche en cours pour en réaliser une autre. D'un point de vue industriel, ce modèle correspond à un modèle de tâches dans un projet de développement informatique, tel que ceux pratiqués dans la société Néréide : les tâches développement, conception, tests, etc. Une tâche peut être interrompue suite à une indisponibilité de la personne qui réalise la tâche lorsque celle-ci est en cours. Dans ce cas, l'exécution de cette tâche sera reprise dès son retour. Par contre, le fait de ne pas autoriser la préemption par d'autres tâches a été introduit pour faciliter la gestion de planning de l'équipe du projet, et éviter ainsi de perturber les personnes par le basculement d'une tâche à une autre.
- **Date de disponibilité** r_i : dans ce modèle, nous considérons des dates de début au plus tôt. Cela permettra de modéliser des dépendances entre les projets. Il arrive qu'une tâche A_i dans un projet P ne puisse pas commencer avant la fin d'une autre tâche $A_j^{'}$ dans un autre projet $P^{'}$. Dans ce cas, la date de fin de $A_j^{'}$ est considérée comme une date de disponibilité de la tâche A_i.
- **Date de fin souhaitée** d_i : des dates de fin souhaitées sont considérées dans ce modèle, contrairement au problème $PMSPSP$. Il s'agit en général des dates de livraison ou autres jalons convenus avec le client. Le respect de ces dates est un facteur important pour mener à bien un projet et donner confiance aux clients.
- **Coût par compétence** : malgré son importance dans la gestion des projets en général et les projets informatiques en particulier, l'aspect financier n'a pas été considéré dans le problème du *PMSPSP*. Nous l'avons intégré dans ce modèle. Le coût d'une personne est lié à la compétence qu'elle exerce. Pour chaque personne P_m et compétence S_k, un coût $w_{m,k}$ par unité de temps est associé. Outre le coût financier, le coût d'affectation d'une personne à une compétence permet de prendre en compte les préférences des personnes à mettre en œuvre leur compétences, gage d'implication

et qualité du projet.

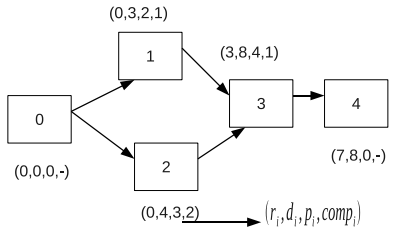
Mises à part ces différences, nous considérons pour ce problème les mêmes relations de précédence que précédemment et pour chaque tâche A_i une durée opératoire p_i est considérée. Les notions de compétence et disponibilité des personnes sont identiques à celles utilisées dans le *PMSPSP*. Les notations liées aux précédence entre les tâches, maîtrise de compétences par les personnes et disponibilités des personnes sont identiques à celles présentées dans la Section 1.3.1.

Pour ce modèle, l'objectif est de trouver des solutions respectant toutes les contraintes du problème en optimisant le retard maximum T_{max} et le coût total du projet. Il s'agit d'un problème bi-critère d'ordonnancement de projet.

Un récapitulatif des notations utilisées est donné dans le tableau 1.2.

1.3.2.1 Exemple

La figure 1.3 représente une instance du *MSRMST* de 3 tâches réelles (1, 2 et 3) avec les deux tâches fictives (0 et 4). Les tâches 1 et 2 n'ont pas de prédécesseurs, alors que la tâche 3 est précédée par 1 et 2. Les libellés figurant en dessous de chaque tâche donnent respectivement la date de disponibilité de la tâche, sa date de fin souhaitée, sa durée opératoire et la compétence dont elle a besoin pour sa réalisation. Par exemple, la tâche 2 a une date de disponibilité de 0, une date de fin souhaitée de 4, sa durée opératoire est égale à 3 et elle nécessite la compétence 2 pour sa réalisation. Le tableau à droite précise les compétences de chaque personne, ainsi que les coûts par personne/compétence. Dans la dernière colonne, les intervalles d'indisponibilité de chacun sont donnés. La personne P_2 maîtrise les deux compétences avec respectivement 1 et 3 comme coût et est indisponible entre les dates 4 et 6. La personne P_1 maîtrise seulement la compétence 1 avec un coût de 1 et n'est pas disponible sur les deux intervalles $[0, 1]$ et $[6, 7]$.



Personnes	Compétence1	Compétence2	Indisponibilité
P1	1	-	[0,1] U[6,7]
P2	1	3	[4,6]

FIGURE 1.3 – Exemple du *MSRMST*

1.3.2.2 Complexité

Ce problème est une généralisation du *Job-shop flexible* [Brucker et Schlie,]. Le problème de *Job-shop flexible* est un problème *NP-Difficile au sens fort* [Brucker et Schlie,

Notation	*Définition*
	Ensembles
n	nombre de tâches réelles du projet
$\mathcal{A} = \{A_i \| i \in \{0, \ldots, n+1\}\}$	l'ensemble des tâches ou activités du projet avec A_0, A_{n+1} les deux tâches fictives marquant respectivement le début et la fin du projet
K	nombre de compétences
$\mathcal{S} = \{S_k \| k \in \{1, \ldots, K\}\}$	ensemble de compétences nécessaires pour la réalisation du projet
M	nombre de personnes affectées au projet
$\mathcal{P} = \{P_m \| m \in \{1, \ldots, M\}\}$	ensemble de personnes affectées au projet
HZ	horizon de planification (le nombre de période maximum autorisé entre le début de planification et la fin du projet)
$\mathcal{T} = \{0, 1, \ldots, \ldots HZ - 1\}$	ensemble de périodes de temps sur lesquelles le projet peut s'exécuter
	Données liées aux tâches
p_i	durée de la tâche A_i
r_i	date de disponibilité de la tâche A_i
d_i	date de fin souhaitée de la tâche A_i
$\mathcal{G}(\mathcal{A}, E, d)$	graphe de précédence évalué avec les durées des tâches
$\{A_i, A_j\} \in E$	Si A_j est un successeur direct de A_i
	Données liées ressources
$MS_{m,k}$	égal à 1 si P_m maîtrise S_k, sinon est égal à 0
$w_{m,k}$	le coût unitaire de P_m lorsqu'il exerce la compétence S_k
$Dispo(m, t)$	égal à 1 si P_m est disponible sur $[t, t+1]$, 0 sinon
$MS_{m,k,t}$	égal à 1 si P_m est disponible sur $[t, t+1]$ et maîtrise S_k, 0 sinon

Tableau 1.2 – Notations utilisées pour le problème *MSRMST*

Le diagramme de Gantt présenté à la figure 1.4, représente une solution réalisable de l'instance dans la figure 1.3. Dans cette solution, les contraintes de précédence sont respectées car la tâche 3 a commencé après la fin de son dernier prédécesseur 1 et 2. Les contraintes de disponibilité sont aussi respectées. Une seule préemption existe dans cette solution, il s'agit de la tâche 3 qui a été interrompue pour une période de temps entre 6 et 7. Cette préemption correspond bien à une période d'indisponibilité de la personne P_1 qui est affectée à cette tâche, ce qui respecte les contraintes du problème. Aucune tâche n'est en retard par rapport à sa date de fin souhaitée ce qui donne un $T_{max} = 0$. Le coût total du projet est égal à $(2 \times 1 + 3 \times 3 + 4 = 15)$.

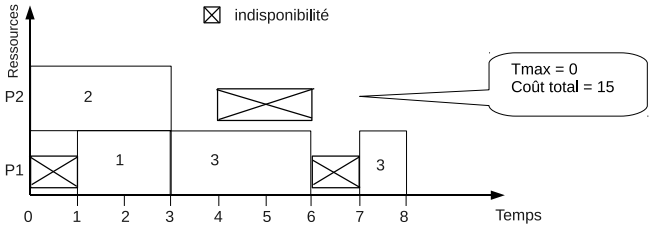


FIGURE 1.4 – Solution réalisable

], [Jurisch, 1992]. Ainsi, si l'on relaxe les contraintes sur les dates de début et de fin et on considère des graphes de précédence où les chemins entre la tâche A_0 et la tâche A_{n+1} sont disjoints, tout en relaxant les contraintes d'indisponibilité des ressources, on a un problème de *Job-shop flexible*, avec un nombre de jobs égal au nombre de chemins et le nombre de machines est le nombre de personnes. Cela nous permet de déduire que le problème *MSRMST* est *NP-Difficile au sens fort.*

1.4 Conclusion

Dans ce chapitre, nous avons défini les deux problématiques traitées à travers cette étude, ainsi que le contexte et les motivations de ces travaux. Nous avons donné la définition de quelques terminologies utilisées dans ce document. Dans la section 1.3.1, une définition formelle et un exemple du premier modèle *PMSPSP* ont été donnés. Dans la section 1.3.2, le deuxième problème a été décrit et un exemple a été présenté. Nous avons aussi établi la liaison entre ces problèmes et les travaux de la littérature proches de ces modèles, tout en montrant les différences et l'originalité des problèmes abordés.

Dans le chapitre suivant, nous présentons un état de l'art qui se focalise essentiellement sur les travaux existants et connexes aux problèmes abordés dans notre étude, et qui sont sources de nos inspirations pour mettre en place des méthodes de résolution.

Chapitre 2

Etat de l'art des problèmes de gestion de projets à contraintes de ressources limitées

Dans cette partie, nous présentons un état de l'art des problèmes de gestion de projets à contraintes de ressources (*Resource Constrained Project Scheduling Problem*) et ses extensions, en rapport avec notre étude. Principalement, cet état de l'art se focalisera sur :

- les problèmes d'ordonnancement de projet multi-compétences ;
- la préemption dans les problèmes de gestion de projet à contraintes de ressources ;
- l'ordonnancement de projet réactif.

2.1 Problème RCPSP

Le *RCPSP* (*Resource Constrained Project Scheduling Problem*) est le premier problème d'ordonnancement modélisant un problème de gestion de projet à contraintes de ressources limitées proposé dans la littérature. Ce problème a fait l'objet de très nombreuses études par les chercheurs de la communauté d'ordonnancement [Blazewicz *et al.*, 1983], [Brucker *et al.*, 1999], [Néron, 1999], [Kolisch et Padman, 2001]. Le *RCPSP* dans sa version standard modélise un projet comme un ensemble de tâches (activités) liées entre elles par des relations de précédence classiques de type fin-début. Chaque tâche a une durée opératoire et nécessite pour son exécution une quantité constante d'une ou plusieurs ressources renouvelables tout au long de son exécution. Les ressources sont disponibles en quantité constante pendant l'horizon de planification du projet, et chaque ressource peut traiter plusieurs tâches à la fois dans la limite de sa capacité. L'objectif est d'ordonnancer les tâches en respectant les contraintes de précédence et de ressources, tout en minimisant la durée du projet, sachant qu'une tâche ne peut pas être interrompue après avoir commencé.

De très nombreuses méthodes de résolution exactes ou approchées ont été abordées dans la littérature. Pour un état de l'art complet nous renvoyons le lecteur à [Hartmann et Briskorn, 2010].

De même, de nombreuses extensions du *RCPSP* ont été proposées et étudiées. Nous soulignons, dans la suite de cette section, celles qui présentent un lien fort avec notre étude.

- **RCPSP avec dates de disponibilité et date de fin impératives (*deadlines*) :** Dans [Drezet et Billaut, 2008], [Kis, 2005], [Kis, 2006], [Klein et Scholl, 2000b], [Klein et Scholl, 2000a] et [Baptiste *et al.*, 1999], les auteurs étudient le RCPSP avec des dates de disponibilité et des dates de fin impératives. [Ballestín *et al.*, 2006], [Brânzei *et al.*, 2002], [Chiu et Tsai, 2002], [Neumann *et al.*, 2002] et [Ozdamar *et al.*, 1998] considèrent des dates de fin au plus tard avec des pénalités de retard.
- **Contraintes de synchronisation :** dans [Brucker et Knust, 2001], les auteurs introduisent la notion de "parallélisme" entre deux tâches i et j qui impose que les deux tâches doivent s'exécuter au moins une unité de temps simultanément. Néanmoins, cette notion n'intervient pas dans la définition du problème traité, mais est introduite pour la mise en place d'un schéma de séparation efficace au sein d'une méthode de type Procédure par séparation et évaluation (Branch-and-Bound).
- **Ressources dédiées :** les ressources dédiées sont des ressources qui ne peuvent être affectées qu'à une seule tâche pendant une période de temps (voir [Bianco *et al.*, 1998,Bianco *et al.*, 1999]). Dans [Dorndorf *et al.*, 1999], les auteurs font référence au RCPSP avec ressources dédiées comme problème d'ordonnancement disjonctif (voir aussi [Dorndorf *et al.*, 2000]).
- **Ressources à capacité dépendante du temps :** les ressources à capacité variable en fonction du temps ont été étudiées, par exemple, dans [Bomsdorf et Derigs, 2008], [Klein et Scholl, 1999], [Klein, 2000a], [Klein et Scholl, 2000b], [Nonobe et Ibaraki, 2002], [Pesch, 1999] et [Schwindt et Trautmann, 2000]. Dans [Hartmann, 1999], l'auteur utilise les ressources dont la capacité varie dans le temps pour déterminer les disponibilités des chercheurs et équipements du laboratoire dans un projet de recherche médical.

2.2 Problème RCPSP multi-modes

Le RCPSP multi-modes noté *MM-RCPSP* (*Multi-Mode Resource Constrained Project Scheduling problem*), est une extension du *RCPSP* standard. Dans le *RCPSP* standard, la durée d'une tâche et les consommations des ressources sont connues et uniques. Tandis que dans le *MM-RCPSP* chaque tâche peut s'exécuter selon plusieurs modes définis à l'avance. Chaque mode d'une tâche correspond à une durée et des consommations des ressources différentes. Cela veut dire qu'une tâche peut être plus longue en utilisant le mode m_1 qu'en utilisant le mode m_2, ou nécessite des types de ressources et/ou des quantités de ces ressources différentes d'un mode à l'autre. Outre ces différences, le *MM-RCPSP* définit un projet exactement comme le *RCPSP* avec les contraintes de précédence, la non-préemption des tâches, et l'objectif est de trouver une solution réalisable, minimisant la durée du projet.

Les travaux de Elmaghraby [Elmaghraby, 1977] ont été parmi les premiers à traiter le *MM-RCPSP*. Durant ces trois dernières décennies, ce problème a reçu l'attention de nombreuses études, e.g., [Brucker *et al.*, 1999], [Kolisch *et al.*, 1996], [Demeulemeester, 1992]. Des méthodes de résolution exactes ont été proposées visant à résoudre le *MM-RCPSP*, elles sont

basées majoritairement sur une énumération explicite des différents modes d'exécution des tâches [Sprecher, 1994].

2.3 Problème d'ordonnancement de projet multi-compétences

Le problème d'ordonnancement de projet multi-compétences MSPSP (*Multi-Skill Project Scheduling Problem*), comme il a été défini par [Bellenguez-Morineau, 2006], modélise un projet découpé en n tâches $\mathcal{A}_i \in \{0, \ldots, n+1\}$, avec $\mathcal{A}_0$ et $\mathcal{A}_{n+1}$ deux activités fictives de durée nulle, qui représentent respectivement le début et la fin du projet.
Ces activités sont liées entre elles par des relations de précédence de type "fin-début", i.e, le projet est représenté sous forme d'un graphe potentiel-tâche E, avec $(i,j) \in E$ si la tâche i doit être réalisée avant de commencer la tâche j. Les tâches sont aussi caractérisées par leur durée d'exécution (p_i) et leurs besoins en compétences. Les besoins en compétences sont exprimés en nombre de personnes $b_{i,k}$ maîtrisant une compétence S_k qui doivent travailler sur la tâche $\mathcal{A}_i$ pendant sa durée d'exécution p_i. Une tâche peut donc avoir besoin de différentes compétences et plusieurs personnes pour chacune de ces compétences.
Les ressources considérées sont des ressources humaines (personnes), chaque personne P_m maîtrise une ou plusieurs compétences parmi les compétences nécessaires pour le projet. La disponibilité de chaque personne est donnée tout au long de l'horizon de planification. Pour chaque personne P_m et chaque période de temps $t \in \{0, \ldots, T-1\}$, $dispo(P_m, t) = 1$ si la personne P_m est disponible sur l'intervalle de temps $[t, t+1]$, 0 sinon. Les compétences sont aussi des données binaires : $S_{m,k} = 1$ si la personne P_m maîtrise la compétence k, 0 sinon. La préemption n'étant pas autorisée, la date de fin de chaque tâche c_i est égale à sa date de début s_i plus sa durée d'exécution p_i. Outre ces contraintes, une personne ne peut exercer qu'une seule compétence d'une seule tâche pendant une période de temps t.

L'objectif est de trouver une solution réalisable qui minimise la date de fin du projet C_{max}.

Un modèle linéaire en nombres entiers a été proposé pour ce problème [Bellenguez-Morineau et Néron, 2004a]. Le modèle se base sur deux types de variables binaires indexées sur le temps : $x_{i,m,t}$ qui vaut 1 si la personne P_m commence l'activité $\mathcal{A}_i$ à la date t, 0 sinon. L'autre variable $\delta_{i,m,k}$ a pour but de préciser la compétence k que la personne P_m doit exercer lorsqu'elle est affectée à la tâche $\mathcal{A}_i$, $\delta_{i,m,k} = 1$ si la personne P_m exerce la compétence S_k lorsqu'elle travaille sur la tâche $\mathcal{A}_i$, 0 sinon .
Les contraintes de précédence sont modélisées comme ci-dessous :

$$\frac{\sum_0^M \sum_0^{T-1} x_{i,m,t} \cdot t}{\sum_0^K b_{i,k}} + p_i \leq \frac{\sum_0^M \sum_0^{T-1} x_{j,m,t} \cdot t}{\sum_0^K b_{i,k}} \tag{2.1}$$

avec $\sum_0^{T-1} x_{i,m,t} \cdot t$ est la date de début de la tâche $\mathcal{A}_i$.
Une personne ne peut être affectée qu'à un seul besoin d'une activité donnée :

$$\sum_0^{K-1} \delta_{i,m,k} \leq 1 \tag{2.2}$$

Une personne ne peut être affectée pendant $[t, t+1]$ qu'à une seule activité :

$$\sum_{0}^{n} \sum_{d=t-p_i+1}^{d=t} x_{i,m,d} \leq 1 \ \forall t, \forall n \tag{2.3}$$

Les personnes affectées à la même tâche doivent la commencer à la même date qui correspond à sa date de début :

$$\sum_{0}^{T-1} x_{i,m,t} \leq \frac{\sum_{0}^{M-1} \sum_{0}^{T-1} x_{i,m,t} \cdot t}{\sum_{0}^{K-1} b_{i,k}} \tag{2.4}$$

Une personne ne peut exercer que les compétences qu'elle maîtrise :

$$\delta_{i,l,k} \leq MS_{m,k} \tag{2.5}$$

Le nombre de personnes affectées à chaque tâche doit être égal à la somme de ses besoins :

$$\sum_{0}^{T-1} \sum_{0}^{M-1} x_{i,m,t} \leq \sum_{0}^{K-1} b_{i,k} \tag{2.6}$$

Le nombre de personnes affectées à chaque compétence d'une tâche doit être égal au nombre de personnes nécessaires :

$$\sum_{0}^{M-1} \delta_{i,m,k} = b_{i,k} \ \forall i, \forall k \tag{2.7}$$

Une personne doit exercer une compétence pour une activité à laquelle elle participe :

$$\sum_{0}^{T-1} x_{i,m,t} = \sum_{0}^{K-1} \delta_{i,m,k} \tag{2.8}$$

Le nombre de variables de ce modèle est de $n \cdot M \cdot (T + K)$. Les auteurs montrent que ce modèle ne peut pas résoudre des instances de taille réelle. Selon leurs expérimentations le modèle n'arrive pas à trouver des solutions en temps raisonnable pour des projets de plus de 10 activités.

Notons que d'autres modèles ont été proposés récemment par [Montoya *et al.*, 2010]. Quatre formulations de programmes linéaires à variables mixtes ont été étudiées. La formulation qui a donné les meilleurs résultats est une adaptation du modèle décrit ci-dessus en ajoutant une variable de décision indiquant si une tâche A_i commence à la période t ou non.

Des bornes inférieures pour le MSPSP ont été proposées dans [Bellenguez-Morineau et Néron, 2004a] :

- **Borne inférieure basée sur un RCPSP déduit :** il s'agit de déduire une instance du *RCPSP* standard à partir du *MSPSP*, puis appliquer le raisonnement énergétique proposé dans [Baptiste *et al.*, 1999]. Selon les auteurs, cette borne est dominée par les autres bornes et les résultats expérimentaux n'ont pas été fournis.
- **Borne inférieure basée sur le graphe de compatibilité :** elle est inspirée des travaux de [Mingozzi *et al.*, 1998] pour le *RCPSP*. L'idée est de vérifier, pour chaque couple de tâches $(\mathcal{A}_i, \mathcal{A}_j)$, si les tâches peuvent s'exécuter sur un même intervalle de temps sans violer les contraintes du problème (contraintes de ressources et de précédence). Contrairement à la borne proposée pour le *RCPSP* classique, la vérification n'est pas triviale, car la vérification du respect des contraintes de compétence et de disponibilité de ressources nécessite la résolution d'un problème d'affectation.
- **Borne inférieure basée sur le raisonnement énergétique :** cette méthode est inspirée des travaux menés sur le *RCPSP* [Baptiste *et al.*, 1999], [Erschler *et al.*, 1991], [Lopez *et al.*, 1992]. Le calcul des parties obligatoires des tâches utilise la même procédure que pour le *RCPSP* classique. Chaque test de faisabilité nécessite la résolution d'un problème d'affectation pour affecter les parties obligatoires aux personnes en fonction de leurs compétences.

Selon les auteurs, ces deux dernières bornes inférieures peuvent être complémentaires car les résultats expérimentaux montrent qu'aucune méthode ne domine l'autre. C'est-à-dire que, souvent, l'une des bornes inférieures donne des résultats moyens pour certaines instances, lorsque la deuxième borne trouve de meilleurs résultats et vice-versa.

Des méthodes de résolution approchées ont été également proposées par les auteurs pour le problème de gestion de projets à contraintes de compétences [Bellenguez-Morineau et Néron, 2004b], [Bellenguez-Morineau, 2006] :

- **Heuristique basée sur des règles de priorité :** il s'agit d'une adaptation des schémas d'ordonnancement (sériel et parallèle) [Kolisch, 1996]. Deux stratégies d'affectation des ressources aux tâches pendant la procédure d'ordonnancement ont été testées. Ces deux procédures présupposent une liste de priorité des activités :
  - **Affectation par tâche :** dans cette stratégie, une fois que la tâche est prête à être ordonnancée (tous ses prédécesseurs sont déjà placés), un problème d'affectation à coût minimum est à résoudre. Les coûts utilisés, comme dans la méthode présentée ci-après, sont des indicateurs de *criticité* des personnes déduits à partir des disponibilités des ressources ainsi que la charge totale des compétences maîtrisées par les personnes nécessaires à la réalisation de la tâche. Le but de ces coûts est d'économiser les ressources qui seront les plus sollicitées pour d'autres tâches. Le calcul de ces indicateurs est détaillé dans [Bellenguez-Morineau et Néron, 2004b].
  - **Affectation par groupe de tâches :** dans cette stratégie, il s'agit de trouver une bonne affectation pour plusieurs tâches simultanément, en respectant l'ordre dans la liste de priorité. Les auteurs proposent d'ajouter des coûts supplémentaires sur les tâches pour forcer l'affectation des ressources aux tâches dans l'ordre de leur priorité. De cette manière, le problème d'affectation peut ne pas avoir de solution lorsque l'on cherche à affecter toutes les tâches disponibles au même instant. Deux solutions ont été proposées : (1) prendre le résultat calculé s'il y a des tâches qui ont pu être satisfaites et reporter les autres pour les étapes suivantes (2) enlever des

tâches l'une après l'autre, selon l'ordre décroissant de priorité, jusqu'à ce qu'une affectation faisable pour toutes les tâches à affecter soit trouvée.

- **Métaheuristiques :** des métaheuristiques ont été également étudiées [Bellenguez-Morineau et Néron, 2004b], [Bellenguez-Morineau, 2006], notamment une recherche tabou et une approche par algorithme génétique. Dans les deux approches, deux stratégies ont été utilisées :
  - codage basé sur une liste de priorité : les mouvements de la recherche tabou et les opérateurs génétiques se basent sur une modification sur l'ordre des tâches dans la liste de priorité. Le décodage utilise l'heuristique présentée ci-dessus, pour placer les tâches dans le temps, ainsi que pour leur affecter les ressources nécessaires.
  - codage basé sur la liste de priorité conjointement avec l'affectation associée à chaque tâche. Les mouvements utilisés dans la recherche tabou, ainsi que les opérateurs génétiques, appliquent des modifications sur la liste de priorité, ainsi que des changements d'affectation. Le codage utilise l'heuristique sans avoir besoin de résoudre le problème d'affectation, car l'affectation est connue à l'avance.

Les expérimentions ont été menées en utilisant des jeux de données variés. Des instances basées sur les graphes de précédence issus d'instances de la PSPLIB [Kolisch *et al.*, 1996], [Baptiste *et al.*, 1999], [Néron, 1999] et [Patterson *et al.*, 1989] ont été générées. Le nombre maximum de tâches a été fixé à 90 afin de pouvoir tester toutes les méthodes. Quatre groupes d'instances ont été construits ([Bellenguez-Morineau, 2006]).

Dans [Bellenguez-Morineau et Néron, 2004a], une autre version du *MSPSP* a été étudiée et des bornes inférieures ont été proposées. Dans ce modèle, chaque personne maîtrise une ou plusieurs compétences avec un certain niveau et les besoins des tâches en compétences exigent un niveau minimum. Une personne peut être affectée à une compétence S_k d'une tâche A_i si elle a au moins le niveau minimum requis pour assurer cette tâche. Le problème a été modélisé pour répondre au problème d'ordonnancement de projet dans une entreprise de service d'informatique.

Dans [Heimerl et Kolisch, 2010], les auteurs traitent un problème de gestion de projet multi-compétences dans un environnement multi-projets. Le problème se posait chez une société de fabrication de semi-conducteurs. Il s'agit d'un ensemble de projets $\mathcal{P}$. Pour chaque projet $p \in \mathcal{P}$, on dispose de sa fenêtre de démarrage $[ES_p, LS_p]$, i.e, le projet doit commencer entre ES_p et LS_p, une durée d'exécution d_p . Chaque ressource k parmi l'ensemble de ressources $\mathcal{R}$, peut réaliser une ou plusieurs compétences parmi l'ensemble de compétences $\mathcal{S}$ nécessaires pour la réalisation de tous les projets. Un niveau d'efficacité n_{sk} est associé à chaque personne/compétence lorsque la personne peut exercer la compétence. Le temps passé par une personne k à réaliser une unité de temps d'une compétence s est égal à $\frac{1}{n_{sk}}$. Chaque projet nécessite un sous-ensemble de compétences parmi les compétences nécessaires pour l'ensemble de projets. Le besoin est exprimé par période : l'horizon de chaque projet est divisé en périodes unitaires ; on a, pour chaque période $q \in \{1 \dots d_p\}$, r_{psq} est la quantité de compétence s nécessaire pour le projet p pendant la période q.
Les ressources (personnes) sont de deux catégories :

- **ressources internes :** les employés internes de l'entreprise ; le coût ici est moins élevé que celui des ressources externes (rares), et pour chaque ressource interne il y a une charge maximum qu'elle peut assurer pendant son temps de travail régulier notée $R^r_{k,t}$, et une capacité maximum correspondant aux heures supplémentaires $R^o_{k,t}$. Le coût par unité de temps régulier est noté c^r_k et pour le temps supplémentaire il est noté c^o_k.
- **ressources externes (rares) :** ce sont des ressources que l'on peut louer, et dont le coût d'utilisation est élevé par rapport aux ressources internes. Ces ressources sont supposées être disponibles en quantité illimitée, le coût d'une ressource externe pendant une unité de temps est c^e_s.

On suppose que $(c^e_s > c^o_k > c^r_k)$, cela évitera d'affecter aux projets seulement des ressources externes quand on minimise le coût total des projets.

Les auteurs montrent que c'est un problème *NP-Difficile* [Heimerl et Kolisch, 2010].

Un modèle linéaire à variables mixtes a été proposé. Le modèle se base sur les variables suivantes :

- $x^r_{ptsq} \geq 0$: quantité de travail régulier qu'une personne interne k réalise durant une période t pour assurer la compétence k du projet p.
- $x^o_{ptsq} \geq 0$: quantité de travail supplémentaire qu'une personne interne k réalise durant une période t pour assurer la compétence k du projet p.
- y_{pts} : quantité de travail réalisée par une ressource externe pendant la période t pour assurer la compétence s du projet p.
- Une variable de décision $z_{pt} = 1$ si le projet p commence à la période t, 0 sinon.

Un ratio e_p entre la quantité de travail réalisée par des ressources internes et la quantité réalisée par des ressources externes au profit du projet p est donné pour assurer l'existence des ressources internes dans chaque projet. L'objectif est de trouver une solution telle que les besoins de chaque projet sont satisfaits pendant chaque période de temps. Chaque projet doit démarrer dans sa fenêtre de début et respecter les contraintes de compétence des personnes, ainsi que leurs capacités régulières et supplémentaires, tout en optimisant le coût total de réalisation de l'ensemble des projets.

Deux versions du modèle linéaire ont été testées, ainsi que leur relaxation linéaire. Les instances utilisées sont inspirées des données du département informatique d'une grande société de fabrication de semi-conducteurs. La durée de chaque projet est fixée à 6, l'horizon de planification est égal à 12. Les résultats de chaque version du modèle ont été comparés avec ceux de sa relaxation linéaire. Des comparaisons avec des heuristiques simples utilisées pratiquement par des sociétés pour répondre au même problème montrent l'intérêt du modèle linéaire, et surtout la relaxation linéaire qui était capable de trouver un bon nombre de solutions réalisables avec des coûts meilleurs et en temps acceptable, pour des instances de taille importante.
Une analyse de l'impact de divers paramètres a été menée dans cette étude. Cette analyse

montre l'intérêt d'augmenter la longueur des fenêtres de démarrage des projets, d'augmenter le nombre de compétences par personne interne et de réduire la taille du projet. Les auteurs montrent l'intérêt de ne pas séparer l'affectation et la planification des tâches, comme on le trouve dans des environnements de production [Ernst *et al.*, 2004].

Skilled workforce project scheduling problem(SWPSP) **:** dans [Valls *et al.*, 2009], les auteurs ont étudié un problème d'ordonnancement de projet multi-compétences rencontré dans un centre de services. En effet, le centre de services a un ensemble de ressources (personnes) ($\mathcal{W}$). Chaque personne a un calendrier par jour définissant ses heures de travail ainsi que ses heures de repos. Les compétences des employés sont classées en domaines ($\mathcal{A}$). Chaque ressource W_k est affectée à un ou plusieurs domaines de compétence. Si la personne W_k appartient au domaine de compétence A_i, un niveau d'efficacité $e_{i,k}$ est donné permettant de connaître la rapidité de cette personne lorsqu'elle réalise une tâche dans ce domaine de compétence. Trois niveaux d'efficacité sont considérés (Senior, Junior, Standard). Le centre de services reçoit des demandes des clients pour réaliser des tâches données. Une tâche j doit appartenir à un domaine de compétence parmi les domaines $\mathcal{A}$. La tâche ne peut donc être réalisée que par une personne appartenant à son domaine de compétence. Une durée standard d_j est associée à chaque tâche, cette durée sert à déterminer la durée réelle que va nécessiter cette tâche selon le niveau de la personne qui va lui être affectée. La durée d_{jk} de la tâche j lorsqu'elle est réalisée par la personne W_k est égale à sa durée standard si le niveau de la personne W_k est standard. Si la personne est junior, la durée $d_{j,k} = 1.25d_j$, et $d_{j,k} = 0.75d_j$ si la personne W_k est senior. Les clients ont des exigences pour la qualité de service. La qualité de service est exprimée par des dates de début et fin maximales ms_j et mf_j. Ces dates maximales peuvent ne pas être respectées sous des pénalités cs_j pour un retard d'une unité par rapport à la date de début maximum, et cf_j pour un retard d'une unité par rapport à la date de fin maximum. La préemption des tâches n'est pas autorisée, sauf par des périodes de repos. Dans ce cas, la personne doit reprendre la tâche immédiatement après le repos. Une autre donnée liée aux tâches est la criticité d'une tâche c_j qui reflète le degré d'urgence d'une tâche pour le client. Cette criticité est indépendante des autres données relatives à la qualité de service.

Les tâches sont liées entre elles par des relations de précédence généralisées. Pour chaque couple de tâches (i, j) liées par des relations de précédence, on définit $\delta_{i,j}$ un vecteur de 3 valeurs. La première valeur indique le type de relation (SS, SF, FS, FF). SS pour start-start, SF pour $start - finish$, FS pour finish-start et FF pour finish-finish. La troisième valeur indique quelle tâche est utilisée pour calculer le délai entre les deux tâches. Si le délai est en fonction d'un pourcentage de durée de l'une des tâches, cette valeur doit correspondre à cette tâche, sinon elle est égale à 0. Pour la deuxième valeur, elle dépend de la troisième valeur. Si la troisième valeur est égale à 0, alors cette valeur représente le délai en périodes de temps, sinon elle représente le pourcentage du temps de la tâche indiquée dans la troisième valeur. Par exemple $\delta_{2,3} = (SS, 10, 2)$ veut dire que la tâche 3 ne peut commencer qu'après la fin de 10% de la durée de la tâche 2. Il est à noter que ces relations de précédence généralisées peuvent être cycliques.

Vue la difficulté du problème (le problème de décision est NP-complet), l'objectif principal du problème, qui a été traité dans [Valls *et al.*, 2009], est de trouver une solution faisable.

Néanmoins, d'autres objectifs secondaires ont été considérés de façon lexicographique. Les objectifs considérés sont respectivement :

- z_1 : la somme des dates de début des tâches pondérées par leur criticité ;
- z_2 : a pour objectif d'affecter les meilleures personnes aux tâches ;
- z_3 : une fonction pour mesurer l'équilibrage de la charge de travail entre les ressources.

Une méthode de résolution par algorithme génétique assisté par une recherche locale a été développée dans cette étude [Valls *et al.*, 2009]. Les tâches sont regroupées dans des composants appelés "composants forts". Ces composants sont construits à partir du graphe de précédence de la manière suivante :

- Les deux tâches fictives sont représentées par deux composants fictifs de début et fin.
- Chaque groupe de tâches qui constitue un graphe cyclique maximum dans le graphe de précédence est représenté par un composant.
- Une relation de précédence entre deux composants est établie s'il existe une relation de précédence entre au moins deux tâches de ces deux composants.

Le codage de l'algorithme génétique se base sur une liste de composants et les affectations de chaque composant. Une procédure de génération de solutions inspirée du schéma de génération d'ordonnancement sériel ($SGSS$) [Kolisch, 1996] est utilisée pour décoder la solution. Après le décodage d'une solution, une recherche locale en 3 phases est appliquée pour réduire le nombre de contraintes violées. La recherche locale commence par une procédure de décalage à droite des dates de début des tâches, puis applique des changements sur les dates de début et/ou les affectations. Enfin, une procédure de décalage à gauche est appliquée pour compresser la solution.

Les opérateurs de croisement et mutation se basent sur ceux utilisés dans [Hartmann, 2001].

Une fois l'algorithme génétique terminé, une autre recherche locale est appliquée sur la solution trouvée afin d'améliorer sa partie liée à la criticité des tâches.

Des expérimentations ont été menées sur plus de 720 instances générées aléatoirement. Les résultats montrent l'efficacité de cette méthode pour améliorer significativement les solutions initiales en peu de temps.

Dans [Drezet, 2005], les auteurs ont étudié un problème de gestion de projets sous contraintes humaines proche des problèmes de planification. Le problème étudié ici introduit, en plus des contraintes de compétences, des contraintes légales, qui se basent toutes sur la notion de temps du travail. Les tâches ont des dates de disponibilité et des dates de fin souhaitée. L'objectif est de minimiser le retard algébrique maximum (L_{max}). D'autres contraintes sur les tâches ont été prises en compte, comme les charges minimales et maximales par tâche. La préemption d'une tâche sur une ressource est autorisée si elle continue sur une autre ressource. Des méthodes de résolution approchées traitant les cas prédictif, proactif et réactif ont été proposées dans [Drezet et Billaut, 2008]. Dans cette étude, un

état de l'art très riche sur les problèmes de planification de personnel a été présenté (voir Chapitre 2, section 2.2). Pour un état de l'art sur des problèmes de planification et de la gestion de personnel, nous renvoyons le lecteur à [Ernst *et al.*, 2004].

[Mónica et Tereso, 2010] ont étudié un problème d'ordonnancement de projet multi-modes et multi-compétences(*MRCPSP-MS*). Les ressources ont des niveaux ou (modes) d'utilisation. La durée d'une tâche dépend de la ressource utilisée et le niveau avec lequel elle est utilisée. Trois niveaux sont considérés : bas (niveau 1), standard (niveau 2) et haut (niveau 3). Chaque ressource a un coût lorsqu'elle est utilisée dans le projet. L'objectif est de minimiser le coût total du projet en respectant une date de fin souhaitée avec une pénalité de retard et un bonus d'avance par rapport à cette date. Un modèle linéaire, ainsi qu'une méthode de recherche globale, sont proposés. Des petites instances de taille très réduite ont pu être résolues de façon optimale avec cette approche.

Dans [Li et Womer, 2009], les auteurs ont étudié un cas particulier du MSPSP. Dans ce problème, chaque tâche a une durée opératoire, une date de fin souhaitée et nécessite une ou plusieurs compétences. Les ressources sont des personnes qui maîtrisent une ou plusieurs compétences. Des relations de précédence généralisées sont considérées : des délais entre les dates de début des tâches sont donnés. Une charge maximum par personne est à respecter. On associe à chaque ressource un coût qui représente son salaire pendant la réalisation du projet. L'objectif est de minimiser le coût global du projet. Un modèle linéaire et une méthode de décomposition hybride utilisant la programmation linéaire et la programmation par contrainte sont proposés. Des résultats expérimentaux menés sur des instances de taille entre 10 et 30 tâches sont présentés.

Dans [Correia *et al.*, 2010], les auteurs ont étudié un cas particulier du *MSPSP*. Il s'agit du *MSPSP* sans prise en compte de la disponibilité des ressources. Un modèle linéaire basé sur le séquencement des tâches et leurs dates de début est proposé. Des coupes sur ce modèle ont été proposées afin d'améliorer le temps de son exécution. Les premiers résultats montrent l'utilité du modèle pour des instances de petites tailles.

Dans [Gürbüz, 2010], l'auteur a étudié le problème du *MSPSP* avec des niveaux hiérarchiques en introduisant un deuxième critère en plus de la minimisation de la date de fin du projet. Le deuxième critère à optimiser concerne la perte des compétences de ressources. On suppose que l'affectation d'une personne à une tâche, qui demande un niveau plus bas que le sein, peut être une source de démotivation pour cette personne, ce qui justifie l'intérêt porté à ce critère.
Un algorithme *NSGA-II* a été proposé, le codage des solutions est basé sur une liste de priorité des tâches respectant les relations de précédence et une liste d'affectation pour chaque tâche. Les opérateurs de croisement et de mutation sont similaires à ceux présentés dans [Hartmann, 1998].
Les résultats de cette méthode ont été comparés avec une approche $\epsilon - contrainte$ sur des instances de moins de 17 tâches et avec une recherche aléatoire sur des instances de

taille plus importante (jusqu'à 92 tâches). Les résultats montrent l'intérêt de la méthode proposée par rapport aux deux autres approches, et notamment sur le deuxième critère.

2.4 Problème RCPSP avec prise en compte de la préemption

Dans [Demeulemeester et Herroelen, 1996], les auteurs proposent une procédure par séparation et évaluation pour résoudre le *RCPSP* en relaxant la contrainte de non préemption. Des résultats expérimentaux sur les 110 instances de Patterson montrent que, sur 30 instances, une augmentation significative du temps de calcul par rapport aux méthodes de résolution du *RCPSP* classique [Demeulemeester, 1992, Demeulemeester et Herroelen, 1992] est observée. Malgré cette augmentation de temps, le gain sur la valeur du C_{max} était seulement de $0,78\%$. Les auteurs concluent que l'introduction de la préemption en RCPSP a peu d'impact, sauf si des niveaux variés de disponibilité des ressources sont considérés.

Dans [Ballestín *et al.*, 2006], les auteurs étudient le cas du *RCPSP* avec préemption, deux cas ont été distingués :

1. $1 - PRCPSP$ (nombre maximum de préemptions par tâche limité à 1) : les auteurs ont proposé deux méthodes de résolution pour ce problème. Pour la première méthode, un algorithme glouton basé sur des règles de priorité est développé. Une solution est codée en utilisant deux listes : une liste de taille $2n$, qui contient chaque tâche 2 fois, et une liste de taille n, qui a pour objectif de stocker les durées des premières parties de chaque tâche. Comme cela a été montré dans [Valls *et al.*, 2005], l'usage des heuristiques à passage double(double justification) est très bénéfique ; de ce fait, ce schéma a été employé dans cette heuristique. Le schéma d'ordonnancement sériel *SGS* a été adapté pour ce codage. Un algorithme génétique à croisement double inspiré de la méthode proposée par [Valls *et al.*, 2005] a aussi été développé.
2. $PRCPSP$: c'est le cas général où le nombre de préemptions est illimité. Les auteurs proposent une généralisation des méthodes utilisées dans le cas du $1 - PRCPSP$.

Les deux méthodes ont été testées avec et sans double passage. Les résultats montrent que l'introduction de la préemption du *RCPSP* peut être bénéfique en termes de qualité du C_{max}. Ils ont aussi montré que l'usage de la "Double justification" est très intéressant dans le cas de la préemption.

[Nudtasomboon et Randhawa, 1997], [Bianco *et al.*, 1999], [Brucker et Knust, 2001] et [Debels et Vanhoucke, 2008] autorisent la préemption aux points de temps discrets, [Debels et Vanhoucke, 2008] étendent le concept de préemption à ce qu'ils appellent "Fast-tracking option" : les parties résultantes de la préemption d'une tâche ne seront pas obligatoirement traitées en séquence (c'est-à-dire elles peuvent être traitées en parallèle). [Franck *et al.*, 2001] proposent un calendrier pour les projets ayant des tâches préemptives. Le calendrier est défini par une fonction binaire qui associe à chaque tâche/période la valeur 0 si la tâche ne peut pas continuer ni commencer à cette période (à noter qu'il y a une analogie avec le cas des ressources à capacité dépendante du temps). Dans ce cas, la préemption

est autorisée seulement dans le cas d'interruption dans le calendrier (un concept similaire se trouve dans [Schwindt et Trautmann, 2000]). Leur approche consiste, de plus, à déterminer une période minimale avant laquelle une préemption ne peut pas se produire. [Buddhakulsomsiria et Kim, 2006, Buddhakulsomsiria et Kim, 2007] autorisent la préemption suite aux indisponibilités des ressources. [Drexl *et al.*, 2000] introduisent le concept des "périodes taboues". Il s'agit d'associer à chaque tâche des périodes pendant lesquelles la tâche ne peut pas être exécutée. Dans [Damay *et al.*, 2007], les auteurs ont proposé une résolution du *P-RCPSP* par la programmation linéaire.

2.5 Différentes fonctions objectifs

Dans cette section, nous présentons un panorama sur les critères d'optimisation qui ont été souvent considérés lors de l'étude des problèmes d'ordonnancement de projet.

- Objectifs liés au temps : la minimisation de la durée du projet reste l'objectif le plus populaire, des références existent dans [Kolisch et Hartmann, 2006]. Il existe d'autres objectifs liés au retard (absolu ou algébrique). Dans [Ballestín *et al.*, 2006], [Kolisch, 2000], [Nudtasomboon et Randhawa, 1997] et [Viana et de Sousa, 2000], les auteurs considèrent la minimisation du retard pondéré.
- Objectifs liés au coût du projet : la prise en compte des aspects financiers dans l'ordonnancement de projet a reçu moins d'attention par rapport aux critères liés au temps [Drezet, 2008]. Dans [Achuthan et Hardjawidjaja, 2001], les auteurs utilisent des coûts par tâche selon qu'elle finit en retard ou en avance par rapport aux dates de fin souhaitées. Ils minimisent le coût total du projet. Ils considèrent également des coûts liés aux durées des tâches, qui peuvent être raccourcies ou prolongées. Dans ce problème, une date de fin impérative est associée au projet. Dans [Dodin et Elimam, 2001], les auteurs minimisent les coûts générés par les modifications sur les tâches (diminution des durées, etc.), les coûts des matériaux et de stockage. Ils ajoutent un bonus ou une pénalité selon la fin du projet (un bonus s'il est en avance, pénalité en cas de retard). D'autres études traitant des critères proches ont été abordées, par exemple dans [Nonobe et Ibaraki, 2006], [Rummel *et al.*, 2005]. Cependant la maximisation de la NPV (*Net Present Value*) [Russell, 1970] reste le critère le plus utilisé. Il s'agit de maximiser le profit du projet ce qui correspond d'un point de vue prestataire à la différence entre le prix payé par le client et le coût total généré par le projet suite à l'utilisation des ressources (hommes, machines, produits, etc.). Plusieurs travaux ont utilisé ce critère comme dans [Kimms, 2001], [Mika *et al.*, 2005], [Padman et Zhu, 2006], [Vanhoucke *et al.*, 2001].
- objectifs liés à la robustesse : [Al-Fawzan et Haouari, 2005] et [Abbasi *et al.*, 2006] utilisent la marge libre (Sl_j) en maximisant $\sum Sl_j$. Dans [Kobylanski et Kuchta, 2007], les auteurs proposent de maximiser la marge libre minimum ($\max \min Sl_j$), et bornent le *makespan* par une *deadline*. [Chtourou et Haouari, 2008], suivent l'approche de [Al-Fawzan et Haouari, 2005] et ajoutent d'autres mesures de robustesse. Ils proposent de pondérer la marge libre d'une activité par le nombre de ses successeurs immédiats et/ou la somme de ses demandes en ressources.
- **objectifs pour le ré-ordonnancement :** suite à un aléa imprévu, un projet subit un ré-ordonnancement. Ainsi, on s'intéresse par exemple, aux perturbations de plan-

ning des tâches déjà planifiées. [Calhoun *et al.*, 2002] proposent de minimiser le nombre de tâches dont la date de début change après le ré-ordonnancement. [de Vonder *et al.*, 2007] proposent de ré-planifier en minimisant la déviation totale des nouvelles dates de fin par rapport à celles d'avant. [Drezet, 2005] s'intéresse à la minimisation du changement des affectations des ressources, en minimisant le nombre maximum de changements d'affectations par ressource.

2.6 Prise en compte de l'incertitude et ordonnancement en ligne des projets

Dans cette partie, nous cherchons à mettre en avant les grands axes d'intérêt de la littérature lorsque les données du projet sont considérées incertaines. Dès lors, des méthodes d'ordonnancement anticipant le changement de données sont considérées (ordonnancement proactif). Lorsqu'un ré-ordonnancement est nécessaire pour modifier un planning en cours, suite aux aléas survenus, on parle alors de méthodes réactives.

Il est à noter que, dans la réalité, l'ordonnancement établi au démarrage d'un projet reste très rarement sans changement jusqu'à la fin du projet. On distingue dans la littérature trois types d'ordonnancement [Billaut *et al.*, 2010] :

1. Ordonnancement *prédictif* : dans ce cas, un ordonnancement initial basé sur des données déterministes est établi sans anticiper des aléas éventuels. Les méthodes de résolution abordées dans 2.3 en sont des exemples.
2. Ordonnancement *pro-actif* : c'est un ordonnancement prévisionnel, qui est établi avec prise en compte d'un changement probable sur les données initiales du problème, comme le changement de durée d'une tâche ou l'absence imprévue d'une ressource. [de Vonder *et al.*, 2007] ont développé des heuristiques visant à trouver des solutions stables du *RCPSP*, c'est-à-dire, les ordonnancements calculés devront avoir la capacité d'absorber des aléas sans perturber (ou marginalement) l'ordonnancement actuel. Dans cette étude, un seul type d'aléa est considéré. Il s'agit du changement des durées des tâches. Pour chaque tâche, les auteurs définissent un poids qui reflète le degré d'importance ou non du changement de la date de début actuelle de cette tâche dans le prochain ordonnancement. Les auteurs s'intéressent donc à minimiser la somme pondérée des déviations des dates de début des tâches. Ils calculent tout d'abord une solution optimale en utilisant un algorithme exact tel que ceux proposés dans [Demeulemeester et Herroelen, 1992] et [Demeulemeester et Herroelen, 1997], puis ils fixent une date de fin impérative (C_{max}), qui est égale à $\alpha \cdot C_{max}^{opt}$ ($\alpha = 1.3$). Ensuite, des tampons sont créés après les tâches critiques. Ces tampons doivent permettre de prolonger une tâche critique, sans avoir besoin de décaler les tâches lui succédant.
   Dans [Drezet, 2005], les auteurs ont étudié un problème réel, où l'objectif était de minimiser le L_{max}. Les auteurs soulignaient que, du point de vue industriel, des solutions de bonne qualité ne seront intéressantes que si elles ne sont pas mises en cause, dès lors qu'une personne s'absente ou qu'une tâche dure plus longtemps que prévu. Les auteurs se sont intéressés à une approche proactive, où l'objectif est de

trouver des solutions robustes, selon des indicateurs de robustesse qui se basent sur la probabilité d'absence des personnes et de prolongement de durées des tâches. Des heuristiques basées sur des règles de priorité ont été étudiées.
Dans [Chtourou et Haouari, 2008], les auteurs proposent une méthode proactive pour le *RCPSP* classique, avec des durées de tâches incertaines. L'idée est d'ajouter un autre critère à optimiser. Ce critère est une fonction mesurant la robustesse des solutions. Plusieurs mesures fondées sur la marge libre ont été testées. L'approche proposée se résume en deux phases. Dans la première phase, une heuristique basée sur des règles de priorité est exécutée un nombre de fois important pour optimiser le C_{max} et la valeur minimum trouvée est utilisée comme référence dans la deuxième phase. Dans la deuxième phase, on fait la même chose en optimisant la fonction de robustesse et en gardant le C_{max} à sa valeur calculée lors de la première phase ou à une valeur inférieure. L'étude montre l'intérêt d'introduire ces mesures dans les solutions, surtout que des solutions plus robustes peuvent avoir un C_{max} meilleur que d'autres moins robustes.

Il existe d'autres méthodes qui ne s'intéressent pas à l'anticipation des aléas. En revanche, elles proposent de mieux gérer les aléas lors de leur survenue. Par exemple, dans [Artigues et Roubellat, 2000], les auteurs proposent un algorithme polynomial pour insérer une tâche sur une instance du *RCPSP* en cours d'exécution.

3. ordonnancement *réactif* : appelé aussi ordonnancement en ligne ou dynamique, il s'agit de recalculer un ordonnancement suite aux changements survenus sur les données initiales sur lesquelles l'ordonnancement actuel est basé. Plusieurs approches ont été abordées dans la littérature. On distingue essentiellement :
   - reconstruction d'un nouvel ordonnancement : cette stratégie consiste à exécuter la même méthode d'ordonnancement utilisée lors de la phase prédictive sur le reste du projet, sans prendre en compte le planning actuel ;
   - réparation du solution actuelle : dans cette stratégie, on essaie de réparer la solution existante en insérant les aléas survenus ;
   - construction d'une solution basée sur la solution actuelle, en ajoutant un ou plusieurs critères à optimiser, pour garantir la stabilité par rapport au planning actuel.

A notre connaissance, la littérature sur le cas réactif, pour des problèmes de gestion de projets multi-compétences, est quasi vide. Nous trouvons seulement les travaux de [Drezet, 2005], où les auteurs proposent une méthode réactive qui se limite à un horizon très réduit. En effet, ils considèrent seulement les aléas dont l'impact se limite à une seule journée (divisée sur 6 périodes). L'objectif est de minimiser le nombre de contraintes violées, puis de minimiser le nombre maximum de changements d'affectations par personne. Une recherche tabou a été développée en optimisant les deux critères lexicographiquement.

2.7 Conclusion

Dans ce chapitre, nous avons étudié l'état de l'art des problèmes d'ordonnancement de projets à contraintes de ressources limitées. Nous nous sommes intéressés particulièrement aux problèmes d'ordonnancement de projet prenant en considération les notions de ressources multi-compétences et la préemption. Ainsi, nous sommes intéressés à l'ordon-

nancement en ligne pour les problèmes de gestion de projet. Il ressort de cette étude plusieurs conclusions. Premièrement, contrairement aux problème d'ordonnancement de projet ($RCPSP$) et ses extensions qui ont été largement étudiées dans la littérature, le $RCPSP$ préemptif n'a pas été suffisamment étudié et nous trouvons peu de modèles de gestion de projet qui prennent en considération cette notion de préemption. Cette dernière décennie, la notion de compétences a attiré l'attention des chercheurs et plusieurs modèles ont été étudiés [Bellenguez-Morineau, 2006], [Valls *et al.*, 2009], [Heimerl et Kolisch, 2010]. Néanmoins, l'état de l'art sur les problèmes de projet multi-compétences préemptifs, l'optimisation de plusieurs critères et les problèmes d'ordonnancement de projet multi-compétences réactifs est quasi vide.

Dans le chapitre suivant, nous étudions un problème de gestion de projet multi-compétences préemptif. Nous étudions ce problème dans un cadre général où une solution efficace est proposée dès le début du projet, puis on traite le cas dynamique où un planning doit être révisé lors de l'arrivé des aléas.

Chapitre 3

Problème de gestion de projet multi-compétences préemptif avec synchronisation des tâches au début

Dans la partie 1, nous avons décrit des modèles de gestion de projet multi-compétences auxquels nous nous sommes intéressés durant cette étude. Cette partie portera sur l'étude du problème *PMSPSP*.

Nous allons tout d'abord décrire les jeux de données que nous avons utilisés pour tester les différentes méthodes de résolution du *PMSPSP* prédictif. Puis nous exposons respectivement en détail un modèle linéaire à variables mixtes proposé pour résoudre le cas prédictif du *PMSPSP*, les bornes inférieures et enfin les méthodes approchées, que nous proposons pour résoudre ce problème. La dernière partie de ce chapitre est consacrée au cas réactif du *PMSPSP*, où nous définissons le problème et donnons les notations utilisées. Un modèle linéaire, ainsi qu'une approche exacte utilisant ce modèle sont présentés. Cette partie s'achèvera par la présentation des méthodes approchées, que nous proposons pour ce problème. Enfin, Les résultats expérimentaux des méthodes réactives sont présentés en section 3.5.6.

3.1 Jeux de données

L'utilisation des jeux de données étant transversale à toutes les sections suivantes, nous avons fait le choix de les présenter préalablement à toutes les méthodes de résolution.

Les instances, avec un nombre de tâches supérieur à 20, ont été générées à partir des instances de la *PSPLIB* [Kolisch *et al.*, 1996]. Pour chaque instance de la *PSPLIB* (single mode), nous utilisons son graphe de précédence et la durée de la tâche comme charge maximum (au moins l'une des compétences de la tâche aura pour charge cette durée). Ainsi, pour chaque instance, la complexité NC du graphe de précédence associé est connue. Nous reprenons donc le nombre de tâches n et l'indicateur de densité des relations de précédence

de la *PSPLB*. En ce qui concerne les autres paramètres caractérisant une instance, nous avons :

1. Nombre de ressources (NR) : ce paramètre représente le nombre de personnes qui seront utilisées dans le projet. Trois valeurs sont considérées : 10, 15 et 20 ;
2. Nombre de compétences (NS) : ce paramètre est le nombre de compétences nécessaires pour le projet. Les valeurs utilisées sont 3, 5 et 8 ;
3. Nombre moyen de compétences par personne (SPP) : ce paramètre représente le nombre moyen des compétences maîtrisées par chaque membre de l'équipe du projet. Les valeurs utilisées sont 25%, 50%, 75%, et 100% ;
4. Nombre moyen de compétences par tâche (SPT) : ce paramètre représente le pourcentage moyen de compétences requises par une tâche. On fixe ces valeurs à 25%, 50%, 75%, et 100%. Ce paramètre est similaire au facteur de ressource (resource factor) utilisé dans les instances de la *PSPLIB* ;
5. Disponibilité moyenne par personne (APA) : ce paramètre permet de générer aléatoirement des périodes de disponibilité pour les différentes ressources de façon à ce que la durée moyenne de ces périodes soit égale à un pourcentage donné de l'horizon de planification. Nous avons utilisé les valeurs de $\{75\%, 85\%, 95\%\}$;
6. Taux de préemption (PA) : le pourcentage de tâches dont la préemption est autorisée. Nous avons utilisé des pourcentages de $\{0\%, 25\%, 50, 75, 100\%\}$ dans nos différents tests.

Nous avons généré pour chaque instance de la *PSPLIB* et chaque configuration (NR, NS, SPP, SPT, APA, ARV, PR) 10 instances aléatoirement. Nous imposons que la charge maximale nécessaire à chaque tâche est égale à la durée de la tâche telle que définie dans l'instance originale de la *PSPLIB*. Au cours du processus de génération, nous assurons que les besoins en compétences de chaque tâche peuvent être assurés par des personnes différentes. Pour ce faire, nous procédons comme suit : une fois que les compétences nécessaires à une tâche donnée sont choisies, nous résolvons un problème d'affectation, qui prend en considération les compétences des personnes. Si le problème n'admet pas de solution, nous procédons au retrait de compétences une par une jusqu'à ce qu'une affectation réalisable soit trouvée ou qu'il ne reste qu'une seule compétence (pour chaque compétence il y a au moins une personne susceptible de la réaliser).

En ce qui concerne les instances de 10, 16 et 20 tâches, nous utilisons deux groupes d'instances. Le nombre de personnes est fixé à 6, le pourcentage de préemption à 50%, le pourcentage de compétences par personne à 100% et le pourcentage de compétences par tâche à 50%. Nous utilisons les 10 premières instances de 10, 16 et 20 tâches de la *PSPLIB* (Multi-Modes). La topologie du graphe de précédence est conservée. Pour chaque tâche, nous utilisons la durée du premier mode comme charge maximum demandée par la tâche. Dans la première catégorie (tableau 3.1), la disponibilité moyenne des personnes est de 100%, tandis que dans la seconde catégorie (tableau 3.2), ce pourcentage est de 75%. Pour chaque configuration nous générons 10 instances avec un nombre de compétences égal à 4 et le même nombre d'instances avec un nombre de compétences égal à 5. Bien que le modèle dépende de l'horizon de planification, nous avons utilisé la durée du projet calculée par l'heuristique 3.4.1 comme horizon maximum du projet.

Un ordinateur personnel fonctionnant sous Linux 2.6.32, 32 bits, avec processeur Intel Core Duo processeur de 2,2 GHz et 3 Go de mémoire a été utilisé pour tous les tests.

3.2 Modèle mathématique linéaire à variables mixtes

Les modèles linéaires proposés pour résoudre le *MSPSP* standard [Bellenguez-Morineau, 2006], [Bellenguez-Morineau et Néron, 2004b], [Montoya *et al.*, 2010], [Correia *et al.*, 2010] ont montré quelques limites pour la résolution exacte du *MSPSP*. Nous avons malgré tout opté pour une approche par programmation linéaire à variables mixtes [Dhib *et al.*, 2011b]. Cela permet de valider un modèle linéaire, qui prend en considération toutes les contraintes du problème *PMSPSP* décrit dans le Chapitre 1 et de voir si les contraintes liées à la préemption et la synchronisation peuvent faciliter la résolution du problème ou non.
L'objectif est, dans un second temps, de pouvoir mesurer la performance d'autres méthodes de résolution et en particulier des bornes inférieures en utilisant les solutions optimales calculées par ce modèle.
Le modèle proposé ici est inspiré de celui du *MSPSP* indexé sur le temps décrit en section 2.3. La spécificité de ce nouveau modèle réside essentiellement dans la prise en compte des relations de précédence conjointement avec les contraintes de préemption et de synchronisation.

Sans perte de de généralité nous notons :
- i : indice de tâche, $i \in \{0, \ldots, n+1\}$
- m : indice de personne, $m \in \{1, \ldots, M\}$
- k : indice de compétence, $k \in \{1, \ldots, K\}$
- t : indice de temps, $t \in \{1, \ldots, HZ\}$

3.2.1 Variables

$$\forall i \in \{0, \ldots, n+1\}, \forall m \in \{1, \ldots, M\}, \forall k \in \{1, \ldots, K\}, \forall t \in \{1, \ldots, T\}$$

$$x_{i,m,k,t} = \begin{cases} 1 & \text{si } P_m \text{ fait } S_k \text{ de } A_i \text{ durant } [t, t+1[\\ 0 & \text{sinon} \end{cases}$$

- $\forall i \in \{0, \ldots, n+1\}, s_i$: date de début de A_i,
- $\forall i \in \{0, \ldots, n+1\}, c_i$: date fin de A_i,
- $\forall i \in \{0, \ldots, n+1\}, \forall k \in \{1, \ldots, K\}, c_{i,k}$: date de fin de la compétence S_k de l'activité A_i,
- Z : date de fin du projet (makespan).

3.2.2 Contraintes

- Contraintes de disponibilité et compétences des personnes :

$$\forall i, m, k, t \quad x_{i,m,k,t} \leq S_{m,k,t} \tag{3.1}$$

Une personne ne peut pas faire une compétence pour une tâche donnée à un instant t, sauf si elle en a la disponibilité et maîtrise la compétence.

- Satisfaction des besoins de chaque tâche en termes de charges requises :

$$\forall i,k, \quad \sum_t \sum_m x_{i,m,k,t} = p_{i,k} \tag{3.2}$$

La somme des quantités réalisées de chaque tâche/compétence doit être égale à la charge requise par la tâche.

- Une personne ne peut pas être affectée à plus d'une tâche pendant une période de temps donnée :

$$\forall m,t \quad \sum_i \sum_k x_{i,m,k,t} \leq 1 \tag{3.3}$$

- Contraintes sur les dates de début des tâches :

$$\forall i,k,t \quad s_i \geq t - T \sum_m \sum_{q=0}^{max(0,t-1)} x_{i,m,k,q} \tag{3.4}$$

$$\forall i,k,t \quad s_i \leq t.x_{i,m,k,t} + (1 - x_{i,m,k,t})T \tag{3.5}$$

Les contraintes 4 et 5 forcent la variable s_i à prendre la valeur de la première date t, où une variable $x_{i,m,k,t} = 1$. Cela modélise aussi la contrainte de synchronisation.

- Contraintes sur les dates de fin des tâches :

$$\forall i,k,m,t \quad x_{i,m,k,t}(t+1) \leq c_{i,k} \tag{3.6}$$

$$\forall i,k \quad c_{i,k} \leq c_i \tag{3.7}$$

La date de fin d'une tâche correspond au dernier instant t tel que $x_{i,m,k,t} = 1$.

- Contraintes de précédence :

$$\forall (i,j) \in \mathcal{G}, \quad c_i \leq s_j \tag{3.8}$$

- Contrainte de non-préemption :

$$\forall i \notin \mathcal{A}_{\overline{p}} \quad s_i + p_{i,k} = c_{i,k} \tag{3.9}$$

La date de début plus la durée doit être égal à la date de fin de la tâche.

- Une personne par compétence :

$$\forall i,k,m,m',t,t' \quad (m \neq m') \quad x_{i,m,k,t} + x_{i,m',k,t'} \leq 1 \tag{3.10}$$

A tout instant t, une personne ne peut exercer qu'une seule compétence à la fois, et si cette personne commence une compétence, elle a l'obligation de la finir.

- Définition de la date de fin du projet (fonction objectif à minimiser) :

$$\forall i,k \quad c_{i,k} \leq Z \tag{3.11}$$

3.2.3 Fonction objectif

Comme annoncé dans la définition du *PMSPSP*, nous nous sommes intéressés à la minimisation de la date de fin du projet. Par conséquence, la fonction objectif est :

$$Min \quad Z \tag{3.12}$$

3.2.4 Taille du programme linéaire développé

Comme dans tout programme linéaire, la taille du programme linéaire se donne par le nombre de variables et le nombre de contraintes. Ces quantités sont mesurées en fonction de la taille des données du problème. Ainsi, nous avons :

1. Nombre de variables binaires

$$NBD = (n+2) \times M \times K \times HZ \tag{3.13}$$

2. Nombre de variables entières

$$NBE = SPT \times K \times n + 2 \times n + 5 \tag{3.14}$$

3. Nombre de contraintes

$$NBC = n \times M \times K \times HZ(\frac{M-1}{2} + HZ + 2) + n(3 + 2 \times HZ) + M \times HZ + n^2 + n(1 - PA) \tag{3.15}$$

3.2.5 Expérimentations

Les expérimentations ont été menées en utilisant *CPLEX 12.2* version académique, et la librairie *Concert technology* pour *Java*. Pour chaque instance, une limite de temps de calcul est fixée à 3600 secondes.

Les colonnes, dans chaque table, représentent respectivement : nombre de tâches (# tâche), nombre de compétences (# Comp), l'horizon de planification moyenne (Hz), le nombre de variables du modèle(# Var), le nombre de contraintes (# Contr), le pourcentage des solutions optimales obtenues dans la limite du temps alloué (% Opt), le temps d'exécution moyen en secondes ($Temps$) et le pourcentage des solutions faisables non optimales (% Fais).

Nous voyons (Tableau 3.1), que sur les 6 jeux de données, presque toutes les instances de 4 compétences ont été résolues $(90\%, 100\%, 90\%)$. En revanche, si le taux de disponibilité baisse de 100% à 75% (Tableau 3.2), seulement 1 jeu de donnée est résolu de façon optimale par le programme linéaire avec une augmentation de temps de calcul considérable. Ceci peut être expliqué par le fait que l'horizon de planification a été calculé sur la base d'une méthode heuristique. Dans le cas où le taux de disponibilité est moins important, la borne supérieure calculée par l'heuristique est susceptible d'être de moins bonne qualité. Le nombre de variables et de contraintes dans le modèle dépendent de l'horizon, il est ainsi

normal que le temps de calcul augmente.
La dernière ligne du tableau 3.2, montre la limite de ce *PLNE*. Plus précisément, pour des instances atteignant les 20 tâches, 6 personnes, 5 compétences, avec un horizon de planification d'environ 40 unités, le programme linéaire est incapable de trouver des solutions optimales dans la limite de temps imposée. Seule une solution faisable a été trouvée.

# tâche	# Comp	Hz	# Var	# Contr	%Opt	Temps	% Fais
10	4	14,1	4 133,8	177 770,2	90%	5,44	10%
	5	15,1	5 221	247 565,1	100%	114,6	0%
16	4	22	9 613	575 487	90%	48,67	0%
	5	23,5	12 817	818 292	60%	209,16	10%
20	4	22	11 749	718 002	90%	55,44	10%
	5	25,9	17 249	1 236 300,4	20%	164	20%

Tableau 3.1 – 10 instances de 6 personnes, 50% de préemption, 100% disponibilité

# tâche	# Comp	Hz	# Var	# Contr	%Opt	Temps	% Fais
10	4	19	5 545	317 536,8	100%	119,7	0%
	5	24,2	8 797	647 114,2	80%	754,25	0%
16	4	30,9	13 457,8	1 114 164,4	50%	915,2	20%
	5	34,7	18 865	1 798 141,8	10%	554	0%
20	4	31,7	16 870,6	1 544 579,4	30%	87,33	0%
	5	39,6	26 291	2 897 469,6	0%	-	10%

Tableau 3.2 – 10 instances de 6 personnes, 50% de préemption, 75% de disponibilité

Vu le nombre de variables et contraintes dans le programme linéaire présenté en section 3.2, la résolution de ce problème d'une façon exacte est très coûteuse et ne peut pas être envisagée pour répondre à un besoin réel, où la taille des instances pourrait atteindre plus d'une centaines de tâches. Pour mesurer les performances des méthodes approchées, des bornes inférieures sont proposées [Dhib *et al.*, 2010, Dhib *et al.*, 2011c, Dhib *et al.*, 2011a].

3.3 Bornes inférieures

En s'inspirant de la littérature [Bellenguez-Morineau, 2006], [Baptiste *et al.*, 1999], [Lopez *et al.*, 1992], nous avons développé des bornes inférieures de la durée du projet. Ces bornes inférieures sont nécessaires, afin de pouvoir évaluer la qualité des solutions approchées calculées par des méthodes heuristiques (Cf Section 3.4), ou encore pour la mise en place des méthodes exactes de type *Procédure par Séparation et Évaluation*. Nous allons donc présenter ces bornes inférieures, qui vont être, par la suite utilisées, pour évaluer les méthodes de résolution approchées présentées dans la partie 3.4.

3.3.1 Bornes inférieures simples

Dans cette partie, nous définissons deux bornes inférieures simples. Elle sont inspirées des bornes inférieures proposées pour le *RCPSP* et le *MSPSP*.

- *Chemin critique amélioré* : cette borne est inspirée de la borne de [Stinson *et al.*, 1978] pour le *RCPSP*. En utilisant le graphe de précédence, les dates de début au plus tôt r_i sont calculées (le chemin le plus long entre la source et le nœud représentant la tâche A_i). r_{n+1} est une borne inférieure pour la date de fin du projet (C_{max}).
  Cette borne peut être améliorée, si on fixe une date de fin impérative pour le projet, que nous notons D. Alors, à partir de cette valeur D, on calcule la date de fin imposée (deadline) de chaque tâche $\tilde{d}_i(D)$, la longueur du plus long chemin de i à $n+1$, moins p_i. Nous disposons donc pour chaque tâche A_i d'une fenêtre de temps $[r_i, \tilde{d}_i(D)]$. On cherche, pour chaque tâche $A_{i'}$ du chemin critique, les tâches n'appartenant pas au chemin critique qui ne peuvent pas s'exécuter en même temps que $A_{i'}$. Le manque de ressources peut donc être calculé pour quantifier la charge nécessaire pour exécuter toutes les tâches du chemin critique et $A_{i'}$ simultanément. On peut déduire de ces quantités la durée minimum à ajouter sur ce chemin pour avoir une nouvelle borne inférieure. L'adaptation proposée ici, consiste à considérer deux tâches ne pouvant pas s'exécuter simultanément si les contraintes de compétences et disponibilités des personnes l'interdisent.
- *Borne inférieure par capacité* : nous relaxons toutes les contraintes, sauf les contraintes de disponibilité des ressources. Pour chaque compétence, nous calculons la charge totale requise par toutes les tâches, divisée par le nombre de personnes maîtrisant la compétence. Le résultat est une borne inférieure valide pour le C_{max}. Plus formellement, nous avons :

$$LB_{cap} = \max_k \frac{\sum_i p_{ik}}{\sum_m S_{m,k}}$$

Notons que ces deux bornes ont été dominées par celles présentées dans la section 3.3.2. C'est pourquoi nous ne présentons pas de résultats expérimentaux dans cette partie. Elles sont néanmoins utiles pour initialiser les valeurs de recherche des bornes inférieures destructives présentées ci-dessous.

3.3.2 Bornes inférieures destructives

Dans cette partie, nous présentons trois bornes inférieures destructives basées sur des tests de faisabilité dont le raisonnement énergétique.

3.3.2.1 Borne inférieure basée sur un modèle de flot

Dans cette approche, les contraintes de précédence sont relaxées, on choisit une valeur D telle que $LB_1 < D < HZ$. Pour chaque tâche A_i, on calcule sa fenêtre d'exécution $[r_i, \tilde{d}_i(D)]$, où $\tilde{d}_i(D)$ est la date de fin impérative pour A_i en considérant D comme la date de fin du projet.
Pour chaque intervalle et chaque compétence, on calcule la charge nécessaire pour satisfaire toutes les tâches. Pour calculer cette charge, nous avons besoin de résoudre un problème

d'affectation. Ce problème d'affectation correspond à la recherche d'un flot maximal dans un graphe.

Cette borne est inspirée des travaux réalisés dans le cadre de l'ordonnancement des travaux sur des machines parallèles identiques [Brucker, 1995].

On considère :

- $I = \{I_1, ..., I_q\}$, l'ensemble de valeurs entières triées dans l'ordre croissant, tel que $\forall\ q$, I_q soit une date de début au plus tôt, une date de fin impérative, une date de début ou de fin de disponibilité d'une personne.

- $L = \{L_1, ..., L_{q-1}\}$ ensemble d'intervalles tel que $L_1 = [I_1, I_2], ..., L_{q-1} = [I_{q-1}, I_q]$
- Soit $L(i) = \{L_w = [I_w, I_{w+1}] \in L \text{ s.t } I_{w+1} \geq r_i \text{ and } d_i(D) \geq I_w\}$
- Soit $T(k) = \{T_i \in \mathcal{T} \text{ s.t } C_{ik} > 0\}$ l'ensemble de tâches nécessitant la compétence S_k pour son exécution
- Soit $E(k) = \{E_j \in \mathcal{E} \text{ s.t } CPT_{jk} > 0\}$: l'ensemble de personnes maîtrisant la compétence S_k

Le premier modèle proposé est un modèle de flot par compétence. Pour une compétence donnée S_k, soit $\mathcal{G}(S_k)$ le graphe construit comme suit (Figure 3.1) :

- un nœud source, un nœud-activité pour chaque $A_i \in T(k)$, un nœud-intervalle pour chaque $I_q \in I$ et un nœud puits ;
- un arc allant du nœud-source vers chaque nœud-activité, de capacité p_{ik} ;
- on lie chaque nœud-activité i, avec un nœud-intervalle $L_q \in L(i)$ par un arc de capacité définie par la longueur de l'intervalle, si la fenêtre d'exécution de l'activité chevauche l'intervalle ;
- un arc allant de chaque nœud-intervalle L_q vers le nœud puits, de capacité définie par la longueur de l'intervalle L_q ($length(L_q)$) multipliée par le nombre de personnes disponibles maîtrisant S_k ($nbdisp_q^k$), avec $nbdisp_q^k = |\{j \in E(k) \text{ tel que } A_j \text{ est disponible sur } L_q\}|$

Propriété 1 *Soit $\varphi^*(k)$ le flot maximum allant de S à P dans $\mathcal{G}(S_k)$, si $\varphi^*(k) < \sum_{i \in T(k)} p_{i,k}$ alors une infaisabilité est détectée et par conséquent $D+1$ est une borne inférieure valide.*

Si $\varphi^*(k) < \sum_{i \in T(k)} p_{i,k}$, alors on peut en déduire que les contraintes de ressources (leurs compétences et leurs disponibilités), ne permettent pas à satisfaire les besoins des tâches en compétence S_k.

Le second modèle considère la charge globale du projet. Ainsi, de la même façon, on résout un problème d'affectation, qui vérifie que les besoins de toutes les tâches peuvent être satisfaits par les ressources.

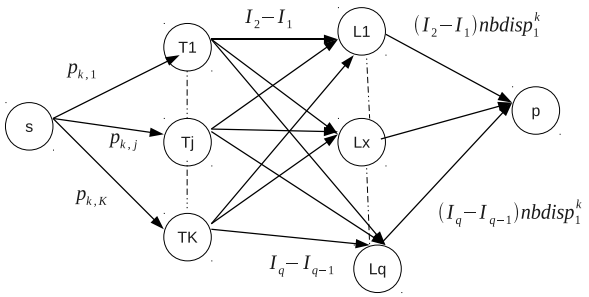


FIGURE 3.1 – Modèle de flot par compétence($\mathcal{G}(S_k)$)

3.3.2.2 Raisonnement énergétique adapté

En s'inspirant des travaux sur l'approche énergétique menés sur le *RCPSP* standard [Lopez *et al.*, 1992], [Baptiste *et al.*, 1999] et la borne inférieure basée sur le raisonnement énergétique proposée pour le *MSPSP* [Bellenguez-Morineau et Néron, 2004a], nous proposons des nouvelles bornes inférieures pour le problème $PMSPSP$. La spécificité principale du problème est la synchronisation des compétences de chaque tâche et la préemption généralisée (certaines tâches sont préemptives et d'autres ne le sont pas).
Dans le cas de préemption autorisée pour une tâche, la partie obligatoire d'une compétence de cette tâche sur un intervalle donné, correspond à la différence entre la charge requise et la partie de la compétence, qui peut s'exécuter en dehors de l'intervalle. Mais, à cause de la contrainte de synchronisation, on peut avoir, par exemple, une partie obligatoire égale à 1, malgré le fait que la totalité de la compétence puisse être exécutée en dehors de l'intervalle. Par exemple, dans la figure 3.4 (à droite), une tâche préemptive A_i demande les deux compétences S_{k_1} et S_{k_2}. La compétence S_{k_2} peut s'exécuter entièrement à l'extérieur de l'intervalle, mais la compétence S_{k_1} ne peut pas s'exécuter à l'extérieur de l'intervalle. A cause de la contrainte de synchronisation, la partie obligatoire de la compétence S_{k_2} devient 1. Le raisonnement énergétique proposé, qui prend en compte la synchronisation est noté LB_{ER}.
Une fois les parties obligatoires calculées, on résout le problème d'affectation associé. Il s'agit d'un problème d'affectation à coût minimum [Bellenguez-Morineau et Néron, 2004b]. La résolution du problème d'affectation à coût minimum est obtenue en temps polynomial par l'adaptation de l'algorithme de Busacker et Gowen [Busacker et Gowen, 1961] en réduisant au minimum le nombre d'itérations lors de la recherche d'un chemin améliorant. Cette amélioration est due à la structure du graphe qui est composé d'un nœud source, nœud puits et deux couches (nœuds compétences et nœuds personnes).

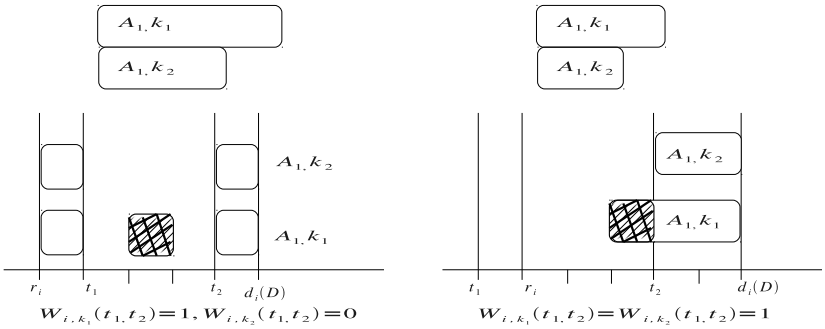


FIGURE 3.2 – Exemple de calcul des parties obligatoires avec préemption et synchronisation

3.3.2.3 Borne inférieure basée sur une formulation linéaire préemptive

Cette borne inférieure destructive est inspirée de [Carlier et Néron, 2000], [Kooli *et al.*, 2012]. On divise l'horizon de planification en intervalles successifs calculés à partir de r_i et $\tilde{d}_i(D)$. Pour chaque intervalle, nous avons d'un côté la demande totale pour chaque compétence et de l'autre, la disponibilité des ressources. On vérifie que la charge totale demandée par les tâches peut être réalisée par les ressources disponibles. Les contraintes de précédence relâchées sont ajoutées (Eq (3.22)). Dans l'équation (3.23) on modélise la contrainte énergétique.
Plus formellement, le modèle est décrit comme ci-dessous.

Données
- $[r_i, \tilde{d}_i(D)]$ pour discrétiser l'horizon de planification sur des intervalles successifs
- l indice de l'intervalle, $\mathcal{I}$ ensemble d'intervalles, $I(i)$ ensemble d'intervalles où la tâche A_i peut s'exécuter.
- $Dispo(m,l)$ la disponibilité de la personne P_m sur l'intervalle l

Variables
- $q_{i,l,m,k}$: temps passé par P_m à réaliser la compétence S_k de A_i sur l'intervalle $\mathcal{I}_l$

Contraintes
- chaque tâche doit être exécutée complètement

$$\forall\, i \in \{0,\ldots,n+1\}, \forall\, k \in \{1,\ldots,K\} \ \sum_m \sum_l q_{i,l,m,k} = p_{i,k} \tag{3.16}$$

- Une tâche ne peut pas s'exécuter en dehors de sa fenêtre $[r_i, \tilde{d}_i(D)]$

$$\forall\, l \ \in I \ \setminus I(i) \ : \ q_{i,l,m,k} = 0 \tag{3.17}$$

– Contraintes de compétences : une personne ne peut pas être affectée à une compétence qu'elle ne maîtrise pas

$$\forall\ i,l,m,k\ q_{i,l,m,k} \leq S_{m,k} \tag{3.18}$$

– Contraintes de disponibilité des personnes par intervalle

$$\forall\ m,l\ \sum_{i}\sum_{k} q_{i,l,m,k} \leq Dispo(m,l) \tag{3.19}$$

$$\tag{3.20}$$

– Contraintes de compétence par intervalle

$$\forall\ i,k,l\ \sum_{m} q_{i,l,m,k} \leq |I_l| \tag{3.21}$$

– Contraintes de précédence relâchées

$$\forall\ i,j \text{ s.t. } i \rightarrow j, \forall\ k,m\ \sum_{v=1}^{l} \frac{q_{i,l,m,k}}{p_{i,k}} \geq \sum_{v=1}^{l} \frac{q_{j,l,m,k}}{p_{j,k}} \tag{3.22}$$

– Contraintes de satisfaction des parties obligatoires

$$\forall\ l,i,k \quad \sum_{m} q_{i,l,m,k} \geq W_{PE}(i,k,t_l,t_{l+1}) \tag{3.23}$$

Propriété 2 *si le programme linéaire défini dans* (3.16) – (3.23) *n'admet pas de solution alors* $D+1$ *est une borne inférieure valide.*

3.3.2.4 Ajustement des fenêtres d'exécution et *shaving*

Afin d'améliorer la borne inférieure, Raisonnement Énergétique Amélioré (LB_{ER}) présentée ci-dessous, nous sommes inspirés de la procédure d'ajustement proposée dans [Baptiste *et al.*, 1999]. La spécificité, dans notre cas, est le calcul de la marge d'une compétence d'une tâche donnée pour un intervalle. En effet, les compétences sont inter-dépendantes du fait qu'elle partagent les mêmes ressources. Plus précisément, le choix d'une ressource pour une compétence donnée peut impacter les autres compétences, puisque les ressources ne sont pas homogènes. Donc, pour calculer la marge d'une compétence/tâche sur un intervalle, nous sommes obligés de résoudre un problème d'affectation à coût minimum.

Calcul des marges : soit A_i une tâche non préemptive, qui nécessite p_{ik} période de temps de la compétence S_k, $W_{i,k}(t_1,t_2)$ la partie obligatoire de la compétence S_k et $W^l_{i,k}(t_1,t_2)$, $W^r_{i,k}(t_1,t_2)$ respectivement les parties à gauche et à droite de la compétence S_k de la tâche A_i. On note $\mathcal{P}_k \subseteq \mathcal{P}$ l'ensemble de personnes maîtrisant la compétence S_k. Soit $Slk_{i,k}(t_1,t_2)$ la marge de la compétence S_k de la tâche A_i sur l'intervalle $[t_1,t_2]$. Elle est, par définition, la charge restante pour réaliser cette compétence après satisfaction de toutes les parties obligatoires sur l'intervalle $[t_1,t_2]$. A cause de l'hétérogénéité des ressources et de la variété de leurs compétences, une personne $P_m \in \mathcal{P}_k$ peut faire d'autres compétences, ou

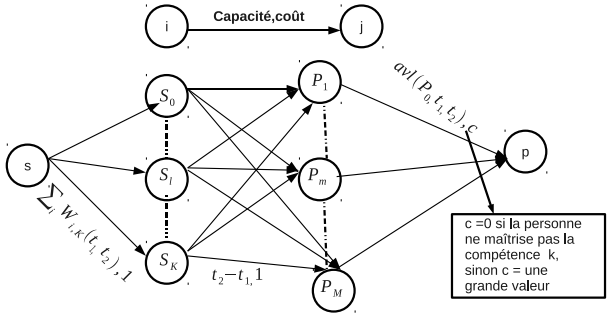


FIGURE 3.3 – Graphe de calcul de marge pour la tâche A_i et la compétence S_k

la même compétence pour d'autres tâches. Dans ce cas, pour trouver la marge restante pour réaliser la compétence S_k, après la réalisation de toutes les parties obligatoires, on résout le problème d'affection modélisé par un flot maximum à coût minimum comme présenté en Figure 3.3. Pour chaque personne $P_m \in \mathcal{P}_k$ le coût (valeur c dans la figure) est très élevé, et égal à 0 pour toutes les autres personnes. Cela implique que, nous n'affectons pas aux autres compétences, les personnes maîtrisant la compétence S_k, sauf s'il n'y a pas d'autres pouvant la réaliser. On note par $flw(m, t_1, t_2)$ le flot passé d'un nœud P_m aux puits, qui représente la durée de travail de la personne P_m durant l'intervalle $[t_1, t_2]$, et $avl(m, t_1, t_2)$ sa disponibilité sur cet intervalle. La marge est donc :

$$Slk_{i,k}(t_1, t_2) = \sum_{m \in \mathcal{P}_k} avail(m, t_1, t_2) - \sum_{m \in \mathcal{P}_k} flw(m, t_1, t_2) + W_{ik}(t_1, t_2)$$

Cette quantité est comparée avec les parties à gauche et à droite. Si elle est inférieure à l'une d'entre elles, la fenêtre d'exécution peut être ajustée [Baptiste *et al.*, 1999].
A partir de cette adaptation des ajustements énergétiques, on applique la procédure développée dans [Baptiste *et al.*, 1999] pour calculer la borne inférieure et mettre à jour les fenêtres d'exécution des tâches. On la note par LB_{ERA}.
Enfin, pour renforcer cette borne, nous avons adapté la procédure de *shaving* proposée dans [Haouari *et al.*, 2012] inspiré de [Péridy et Rivreau, 2009]. Dans notre cas, le *shaving* est appliqué seulement sur les tâches non préemptives. Cette borne est notée LB_{SHA}.

3.3.3 Résultats expérimentaux

3.3.3.1 Comparaison des bornes inférieures

Les résultats présentés en tableau 3.3 ont été menés sur le groupe de données *G_ 30_ 1* (Adaptation des premières 160 instances de 30 tâches de la *PSPLIB*). Les valeurs de la première colonne (LB_{FM}, LB_{LM}, LB_{ER}, LB_{ERA}) représentent respectivement les valeurs des bornes inférieures : modèle de flot, le modèle linéaire préemptif, le raisonnement énergétique, le raisonnement énergétique avec ajustement. Notons que le *shaving* a amélioré seulement une seule instance par rapport à la borne LB_{ERA} et coûte beaucoup de temps,

c'est pour cela que nous ne présentons pas ses résultats dans ce tableau. Les colonnes 2, 3 et 4 reportent respectivement la déviation moyenne de chaque borne inférieure par rapport à la meilleure des quatre bornes inférieures testées ici, le nombre de fois où la borne est strictement meilleure que toutes les autres, et le temps de calcul en secondes.

	(BI- Meil)/BI (%)	# Meil	Temps (sec)
LB_{FM}	1,31	-	0,01
LB_{LM}	1,25	-	1,21
LB_{ER}	0,07	-	0,01
LB_{ERA}	0	11	18

Tableau 3.3 – Résultats des bornes inférieures sur des instances adaptées de la *PSPLIB*(KSD30)

Les résultats présentés dans le tableau 3.3 montrent un écart moyen de moins de 0.1% entre la borne inférieure du raisonnement énergétique adapté (LB_{ER}) et la meilleure borne avec un temps de calcul moyenne de $0.01s$. L'ajustement des fenêtres d'exécution (LB_{ERA}) conduit à une amélioration sur la borne LB_{ER} mais avec un temps d'exécution incomparable ($18s$ contre $0.01s$).

3.3.3.2 Raisonnement énergétique adapté vs PLNE

Vus les résultats présentés dans le tableau 3.3.3, la borne inférieure du raisonnement énergétique adapté (LB_{ER}) semble un bon compromis entre qualité et temps de calcul. C'est pour cette raison que nous avons voulu mesurer sa qualité par rapport aux solutions optimales calculées par le modèle linéaire . Nous allons donc présenter les résultats de comparaison du modèle linéaire et de la borne inférieure LB_{ER}. Les résultats sont calculés seulement pour les instances pour lesquelles le programme linéaire a trouvé des solutions optimales. Les instances sont celles utilisées pour tester le modèle linéaire (10,16 et 20 tâches). La colonne 3 donne le nombre des solutions optimales, la colonne 4 donne la déviation moyenne entre la borne inférieure LB_{ER} et la solution retournée par le modèle linéaire.

# tâche	# compétence	# opt	(Opt-LB)/LB (%)
10	4	10	10,5
	5	8	21,3
16	4	5	5,9
	5	1	4,1
20	4	3	22,4
	5	0	-

Tableau 3.4 – LB_{ER} vs programme linéaire sur les instances de 10, 16 et 20 tâches avec disponibilité de 75%

# tâche	# compétence	# opt	(Opt-LB)/LB (%)
10	4	9	2, 15
	5	10	1, 5
16	4	9	0, 6
	5	6	1, 8
20	4	9	0
	5	2	3, 8

Tableau 3.5 – LB_{ER} vs programme linéaire sur les instances de 10, 16 et 20 tâches avec disponibilité de 100%

Analyses : des remarques peuvent être tirées des résultats présentés en tableaux 3.7 et 3.8 : nous pouvons constater par exemple, que les résultats de la borne inférieure sont meilleurs lorsque les personnes sont disponibles à 100% sur tout l'horizon de planification. Cela peut être expliqué par le fait que lors du calcul des parties obligatoires, nous ne prenons pas en compte le fait qu'une compétence d'une tâche doive être réalisée par la même personne. Par conséquent, les parties obligatoires de la même compétence peuvent être satisfaites par deux personnes (ou plus) différentes sur deux intervalles de temps. Lorsque les personnes sont toujours disponibles, la probabilité que ces parties obligatoires soient assurées par la même personne est forte.
Nous remarquons aussi que l'augmentation du nombre de compétences impacte largement les résultats de la borne inférieure. Dans le tableau 3.7, pour les instances de 10 tâches, avec un nombre de compétences de 5, la déviation moyenne est de $21,3\%$ par rapport à l'optimum. Cette déviation diminue (10%) lorsque le nombre de compétence devient 4. Selon la table 3.8, pour les instances de 16 tâches avec un nombre de compétences de 4, la déviation moyenne est de $0,6\%$. Cette déviation est égale à $1,8\%$ pour un nombre de compétences de 5. La déviation moyenne par rapport à l'optimum est importante du fait que lors du calcul des marges restantes, nous ne prenons pas en considération la contrainte de synchronisation et le fait que chaque compétence doive être réalisée par une personne différente. La relaxation de ces contraintes, explique la mauvaise qualité de la borne inférieure lorsque le nombre de compétences par rapport au nombre de personnes est important.

3.4 Méthodes approchées

Vue la complexité de ce problème, des solutions exactes ne pourront pas être envisagées dans un contexte industriel pour résoudre des instances de taille réelle. Par conséquent, nous proposons des méthodes approchées : algorithmes de liste et métaheuristiques.

3.4.1 Heuristique de liste

Dans cette partie, nous proposons de résoudre le problème *PMSPSP* par un algorithme dynamique basé sur différentes règles de priorité. Cette heuristique est inspirée du schéma de placement parallèle "*Parallel schedule generation scheme*", introduit pour la résolution

du *RCPSP* classique [Kelly, 1963], [Brooks et White, 1965], et également utilisé pour résoudre le *MSPSP* [Bellenguez-Morineau et Néron, 2004b]. Contrairement à la littérature, où il s'agit d'un algorithme glouton, nous nous autorisons de revenir sur une décision prise antérieurement durant la construction de la solution. En effet, une tâche préemptive, qui est en cours d'exécution pendant $[t, t+1]$ peut être interrompue pour ordonnancer une autre tâche de priorité plus élevée. Nous parlerons dans la suite de l'algorithme de *Preemptive Adapted Parallel Schedule generation Scheme*, noté *PAPSS*.

Avant de présenter le *PAPSS*, nous allons rappeler l'algorithme glouton de placement parallèle tel qu'il a été utilisé pour résoudre le *RCPSP*, ainsi que le *MSPSP*.

Pour le *RCPSP*, la méthode de placement parallèle résout une instance du *RCPSP* standard de n tâches en n itérations au maximum [Brooks et White, 1965]. Chaque itération est associée à un instant t. Pour chaque instant t, trois listes sont considérées : liste des tâches terminées au plus tard à t, (C_t) ; liste des tâches commencées avant t et en cours à t, (A_t) et la liste des tâches éligibles à t, (E_t). Une tâche est dite éligible si elle peut être ordonnancée à t sans violer ni les contraintes de précédence ni les contraintes de ressources. A l'instant $t = 0$ la liste des tâches terminées C_t contient seulement la tâche fictive du début de projet et la liste A_t est vide. Tandis que la liste E_t contient les tâches qui n'ont pas de prédécesseur et peuvent être ordonnancées sans violation des contraintes de ressources. A chaque itération, les deux étapes suivantes sont réalisées :

1. planifier les tâches de la liste E_t à t selon l'ordre de leur priorité et mettre à jour la disponibilité des ressources à chaque placement d'une tâche. Enlever la tâche de la liste E_t même si elle n'a pas été planifiée à t (une tâche dans E_t peut devenir inéligible à cause de manque de ressources causé par les tâches la précédant dans la liste) et ajouter toutes les tâches planifiées dans la liste A_t
2. calculer le prochain instant t', qui correspond à l'éligibilité de nouvelles tâches ; on enlève les tâches de la liste A_t dont les dates de fin sont inférieures ou égales à t' et on les place dans la liste C_t. Ces deux étapes sont répétées jusqu'à ce que toutes les tâches soient placées.

Pour le *MSPSP*, on peut citer les travaux présentés dans [Bellenguez-Morineau, 2006], où les auteurs ont proposé deux méthodes de placement parallèle, inspirées de la méthode décrite ci-dessus. Dans ces deux méthodes, l'instant du prochain évènement correspond à la disponibilité des ressources et l'éligibilité des tâches par rapport aux relations de précédence.
Dans la première méthode appelée *Méthode de placement parallèle stricte*, les tâches sont triées selon leur priorité. On sélectionne les premières e tâches éligibles tel que la $(e+1)$iéme tâche n'est pas éligible. Un problème d'affectation à coût minimum est par la suite résolu déterminant l'affectation des personnes aux tâches en respectant l'ordre de priorité [Bellenguez-Morineau et Néron, 2004b]. Des indicateurs de criticité sont utilisés pour favoriser l'utilisation des personnes, qui seront les moins sollicitées pour réaliser les autres tâches.
Dans la deuxième méthode (*Méthode de placement parallèle non stricte*), toutes les tâches éligibles à t sont sélectionnées même si une tâche éligible est moins prioritaire qu'une

autre, qui n'est pas encore éligible. Les tâches sont ordonnées selon l'ordre décroissant de leur priorité. Un problème d'affectation comme dans la première méthode est résolu. Si une affectation éligible est obtenue pour toutes les tâches, elles seront planifiées à l'instant t. Sinon, le flot attribué à la première tâche E_0 non satisfaite est calculé. Si ce flot est suffisant pour satisfaire une autre tâche plus prioritaire (apparaissant dans la liste des priorités, après E_0), alors E_0 est retirée de la liste et le problème d'affectation doit être résolu de nouveau. Cette méthode est réitérée jusqu'à l'obtention d'une liste de tâches à planifier simultanément à l'instant t.

Dans la suite, nous allons décrire notre démarche de résolution en tenant compte de la préemption des tâches.

3.4.1.1 Algorithme *Preemptive Adapted Parallel Scheduling Scheme*

A chaque période de temps t, les tâches éligibles (les tâches dont les prédécesseurs sont terminés avant t et dont la date de début au plus tôt r_i est inférieure ou égale à t) sont triées selon une règle de priorité donnée en respectant les relations de précédence. Après, on essaie d'ordonnancer la première tâche dans la liste en déterminant d'abord les personnes susceptibles de la réaliser. Vue la multiplicité des choix possibles pour affecter des personnes à une tâche, nous utilisons une heuristique d'affectation. Cela est établi en essayant d'affecter à la tâche, les personnes les moins critiques parmi celles qui ont les compétences nécessaires. Nous avons donc, un problème d'affectation à coût minimum qui est détaillé dans le paragraphe 3.4.1.3. Si aucune affectation réalisable n'est trouvée à l'instant t, la prochaine tâche de la liste est examinée. Par cette approche, la liste de priorité n'est pas strictement respectée. Une fois que toutes les tâches éligibles ont été examinées, on passe à la première période après t définie par le prochain évènement. Le prochain évènement correspond à une nouvelle disponibilité d'une personne ou à l'éligibilité d'une nouvelle tâche. La structure générale de cette heuristique est présentée dans Algorithme 1. On note par *PAPSS-R1* l'implémentation de l'algorithme *PAPSS* avec la règle de priorité $R1$.

L'algorithme utilise deux procédures : $TrySchedule(ES^t(i), t)$ que nous décrivions dans l'algorithme 2 ; $nextEvent()$ qui a pour objectif de calculer la date du prochaine évènement à partir de la période t. Le but du $TrySchedule(ES_t(i), t)$, comme le montre l'algorithme 2 est de :

1. résoudre le problème d'affectation à coût minimum.
2. selon l'affectation trouvée, la tâche $ES^t(i)$ est planifiée à partir de la période t, c'est-à-dire que tous les créneaux d'intervention des personnes affectées à cette tâche sont réservés.
3. si une personne doit interrompre une ou plusieurs tâches en cours d'exécution durant tous les créneaux considérés, elles doivent être reprises immédiatement par la même personne dès qu'elle a fini. La reprise des tâches interrompues se fait selon l'ordre de leur première date d'interruption.

Algorithme 1: Preemptive Adapted Parallel Schedule Scheme Algorithm(*PAPSS-R1*)

ENTRÉES: $\mathcal{A}$: l'ensemble de tâches à planifier
  $t \leftarrow 0$
  tantque $\mathcal{A} \neq \phi$ et $t < horizon$ **faire**
    ES^t : ensemble de tâches éligibles à t, triées selon la règle de priorité $R1$
    $i = 0$
    tantque $i < |ES^t|$ **faire**
      $feasible \leftarrow TrySchedule(ES^t(i), t)$
      si $feasible$ **alors**
        $\mathcal{A} \leftarrow \mathcal{A} \setminus ES^t(i)$
      finsi
      $i \leftarrow i + 1$
    fin tantque
    $t \leftarrow nextEvent()$
  fin tantque

Concernant le choix de la règle $R1$, des règles de priorité usuelles de la littérature [Hartmann et Kolisch, 2000], ainsi que certaines spécifiques au problème ont été considérées :

1. *Earliest Release Date (ERD)* : une tâche A_i est plus prioritaire qu'une tâche A_j si leur date de début au plus tôt vérifient $r_i < r_j$;
2. *Earliest Due Date (EDD)* : les tâches dont les dates de fin impératives (d_i) sont les plus petites sont placées en priorité ;
3. *Minimum Slack (MINSLK)* : les tâches sont triées dans l'ordre croissant de leur marge libre donnée par $\max(0, d_i - p_i - r_i)$;
4. *Most Number of Successors (MTS)* : les tâches ayant le plus grand nombre de successeurs sont placées en priorité ;
5. *Most number of skills (MSSKILL)* : les tâches demandant le plus grand nombre de compétences sont les plus prioritaires ;
6. *Total Man Power Needed (MAXCHRG)* : les tâches sont triées selon l'ordre décroisant de leur charge totale donnée par $\sum_k p_{i,k}$
7. *Criticality (CRITICITY)* : les tâches les plus critiques sont prioritaires. Le calcul de la criticité est décrit dans le paragraphe 3.4.1.2
8. *Greatest rank (GRNK)* : les tâches sont triées dans l'ordre décroissant selon la charge totale de leurs successeurs $\sum_{j/(i,j)\in E} \sum_k p_{j,k}$
9. *Greatest rank 2 (GRNK2)* : les tâches sont triées dans l'ordre décroissant selon le nombre total de compétences requises par leurs successeurs $\sum_{j/(i,j)\in E} NBSKIL(j)$
10. *Greatest rank 3 (GRNK3)* : les tâches sont triées dans l'ordre décroissant selon la durée totale de leurs successeurs $\sum_{j/(i,j)\in E} \max_k p_{j,k}$.
11. *Random priority (RAND)* : des priorités aléatoires sont associées aux tâches.

Algorithme 2: TrySchedule(tâche A_i, période t)

$flow \leftarrow$ flot calculé en résolvant le problème d'affectation dans 3.4.1.3
si $flow < |SK_i|$ **alors**
  retourner *faux*
finsi
fixer l'affectation de A_i en affectant à chaque compétence la personne dont un flot a été établie entre son nœud et le nœud de la compétence
si A_i n'est pas préemptive **alors**
  planifier chaque compétence S_k de A_i de t à $t + p_{i,k}$
sinon
  planifier chaque compétence S_k de A_i de t à $t + 1$
  pour tout compétence S_k de la tâche A_i **faire**
    si la charge restante $p_{i,k} - 1$ peut être planifiée **alors**
      planifier les $p_{i,k} - 1$ périodes restantes de S_k le plus tôt possible selon la disponibilité de la personne affectée
    sinon
      retourner *faux*
    finsi
  fin pour
finsi
pour tout tâche A_j interrompue par A_i **faire**
  si les disponibilités des ressources permettent de reprendre A_j **alors**
    reprendre A_j
  sinon
    retourner *faux*
  finsi
fin pour
retourner *vrai*

3.4.1.2 Indicateurs de criticité

Les indicateurs de criticité (les coûts considérés dans la recherche du flot maximum à coût minimum) utilisés pour l'affectation des personnes aux tâches sont inspirés de ceux proposés dans [Bellenguez-Morineau, 2006]. Nous donnons ici leur formulation en fonction des données de notre problème.

Soit $PSK(m) \subseteq \mathcal{S}$ l'ensemble de compétences maîtrisées par la personne P_m. $TSK(i) \subseteq \mathcal{S}$ l'ensemble de compétences requises par la tâche A_i. On note $SPE(k) \subseteq \mathcal{P}$, l'ensemble de personne maîtrisant la compétence S_k. La charge totale requise pour chaque compétence $S_k \in \mathcal{S}$, est

$$WL(k) = \sum_{\{i|A_i \in \mathcal{A}\}} p_{i,k}$$

1. Criticité d'une compétence : la criticité d'une compétence S_k est donnée par :

$$SCR(k) = \frac{\sum_{\{m|P_m \in SPE(k)\}} Dispo(m)}{WL(K)}$$

   où $Dispo(m) = \sum_t Dispo(m,t)$ est la disponibilité totale de la personne P_m sur l'horizon de planification.

2. Criticité d'une personne : la criticité d'une personne est la somme des criticités des compétences qu'elle maîtrise

$$PCR(m) = \sum_{\{k|S_k \in PSK(m)\}} SCR(k)$$

3. Criticité d'une tâche : la criticité de chaque tâche est la somme des criticités des compétences nécessaires pour son exécution. Elle est donnée par la formule suivante :

$$TCR(i) = \sum_{\{k|S_k \in TSK(m)\}} SCR(k)$$

Donc, à chaque période de temps t, la tâche (A_i) la plus prioritaire parmi les tâches éligibles est sélectionnée pour être ordonnancée à t, si une affection réalisable est possible. Les personnes affectées à la tâche A_i sont déterminées par la résolution du problème d'affectation à coût minimum, décrit la section suivante.

3.4.1.3 Affectation à coût minimum des personnes à une tâche

Afin de respecter les contraintes de synchronisation, les personnes affectées à la tâche doivent être toutes disponibles, au moins lors de la première période de temps d'exécution de la tâche. De plus, si la tâche n'est pas préemptive, chaque personne doit avoir une disponibilité continue qui correspond à la charge de la compétence qu'elle doit réaliser. Pour résoudre ce problème d'affectation, une modélisation sous forme d'un flot maximum à coût

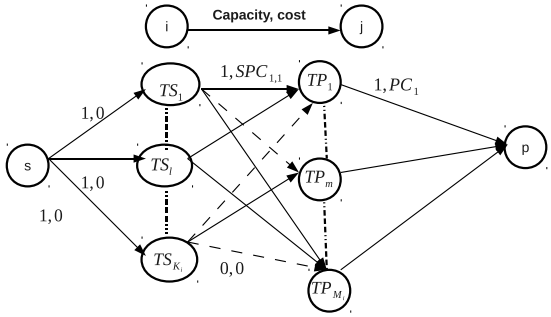


FIGURE 3.4 – Problème d'affectation des personnes à une tâche

minimum est utilisée. Le graphe déduit contient deux niveaux de nœuds : le premier niveau "*nœuds-compétences*" représente les compétences nécessaires pour exécuter la tâche. Tandis que le deuxième représente les personnes disponibles à la période t, et pouvant participer à la réalisation de la tâche. La capacité de chaque arc est égale à 1, ce qui permet d'éviter d'affecter la même personne à plusieurs compétences ou d'affecter plus d'une personne à la même compétence. Plus formellement, l'ensemble des nœuds du graphe est constitué de :

- s : le nœud source ;
- TS : nœuds-compétences ;
- TP : nœuds-personnes ;
- p : le nœud puits.

L'ensemble des arcs E, est constitué de :

- $\forall\ k \in TS$, $(s,k) \in E$. Pour chaque arc, on associe un coût $w_i = 0$ et une capacité $c_i = 1$
- $\forall (k,j) \in TS \times TP$ où $(k,j) \in E$ si la personne P_j est disponible tout au long de la période entre $[t, t+p_{i,k}]$ dans le cas où la tâche A_i n'est pas préemptive sinon P_j doit être disponible sur $[t, t+1]$. Pour chaque arc, on associe une capacité $c_{(k,j)} = 1$, un coût $w_{(k,j)} = 0$ si la personne n'était pas en cours d'exécution d'une tâche préemptive, qui pourra être interrompue par cette tâche, sinon une grande valeur est utilisée comme coût pour éviter autant que possible la préemption.
- $\forall j \in TP$, on a $(j,p) \in E$. Pour chaque arc, on associe une capacité $c_i = 1$ et un coût correspondant à la criticité (Cf 3.4.1.2) de la personne utilisée.

Un nœud-compétence et un nœud-personne sont liés par un arc si les conditions suivantes sont réunies :

- la personne P_m maîtrise la compétence S_k
- si A_i n'est pas préemptive, P_m doit avoir une disponibilité continue égale à $p_{i,k}$ à partir de la date t. Dans le cas d'une tâche préemptive, il suffit d'être disponible sur $[t, t+1]$.

La disponibilité ne tient pas compte des tâches préemptives, qui sont moins prioritaires que la tâche A_i. Plus précisément, on définit la disponibilité continue ($ContAvail(P_m, A_i, t)$)

d'une personne P_m à partir de la période t par rapport à la tâche A_i comme suit :

- $ContAvail(P_m, A_i, t) = 0$ si t est une période d'indisponibilité pour P_m, où la personne P_m exécute une tâche préemptive plus prioritaire que la tâche A_i ;
- $ContAvail(P_m, A_i, t) = 1$ si les conditions suivantes sont satisfaites :
  - t n'est pas une période d'indisponibilité pour P_m
  - P_m n'exécute aucune tâche sur $[t, t+1]$, sauf si la tâche qu'elle exécute est préemptive et en cours (elle a commencé avant t et est en train de s'exécuter entre t et $t+1$), dans ce cas l'une des deux conditions suivantes doit être vérifiée :
    - la tâche que la personne est en train d'exécuter est moins prioritaire que A_i
    - les deux tâches ont la même priorité, mais la tâche A_i a besoin de plusieurs compétences.

L'analyse des résultats expérimentaux montrant les performances de cette méthode est présentée dans la section 3.4.5.1.
Afin d'améliorer la qualité des solutions par rapport aux résultats obtenus par l'heuristique de placement parallèle préemptif, nous proposons des métaheuristiques de type Recherche tabou ou Algorithme génétique.

3.4.2 Métaheuristiques et ordonnancements de projets classiques

Les métaheuristiques sont des approches globales qui peuvent être décrites sans référence à un problème spécifique. Leur objectif est de trouver des solutions proches de l'optimum en explorant l'espace de recherche d'une façon itérative et en guidant cette recherche avec des modifications intelligentes pouvant transformer une solution en une autre éventuellement de meilleure qualité [Siarry et Michalewicz, 2007], [Talbi, 2009]. On peut citer comme métaheuristiques le recuit simulé, la recherche tabou, les algorithmes évolutionnaires, etc. Dans cette partie, nous allons présenter des métaheuristiques inspirés de la littérature de l'ordonnancement de projet adaptées à notre problème.

Nous avons étudié des méthodes de résolution de type métaheuristiques pour les problèmes d'ordonnancement de projet, ainsi que les métaheuristiques proposées pour résoudre des problèmes d'ordonnancement de projet multi-compétences. Dans cette partie, nous allons présenter quelques méthodes de la littérature, qui ont été sources de nos inspirations.

Dans [Hartmann, 1998], l'auteur propose un algorithme génétique pour résoudre le *RCPSP*. Les solutions sont représentées à l'aide d'une liste de priorité $I =< j_1^I, j_2^I, \ldots, j_n^I >$ où n est le nombre de tâches et j_i^I, la $i$ème tâche. L'ordre des tâches dans la liste respecte les relations de précédence (aucune tâche n'apparait avant l'un de ses prédécesseurs). Deux opérateurs de croisement ont été proposés :

1. croisement à un point : deux individus F et M sont sélectionnées pour faire un croisement et produire deux nouveaux individus, un numéro $1 \leq q \leq n$ est tiré aléatoirement. On construit le premier individu enfant D comme suit : les positions $i = \{1, \ldots, q\}$ dans la liste des tâches de D sont prises du parent M. Les positions $i = \{q+1, \ldots, n\}$ sont prises du parent F. Mais les tâches, qui sont déjà prises de M ne doivent pas être dupliquées, c'est-à-dire les tâches recopiées de F sont celles,

qui n'existent pas encore dans D, prises selon leur ordre d'apparition dans F. Le deuxième individu est construit symétriquement, en échangeant les rôles de M et F.

2. croisement à deux points : ici deux indices $1 \leq q_1 < q_2 \leq n$ sont tirées aléatoirement. Les positions $\{1, \ldots, q_1\}$ du D sont prises du parent M. Les positions de $\{q_1 + 1, \ldots, q_2\}$ sont dérivées du parent F de la façon suivante :

   - $J_i^D = J_i^F$ avec k le plus petit indice vérifiant $J_k^F \notin \{J_1^D, \ldots, J_{i-1}^D\}$
     Les positions restantes ($\{q_2 + 1, \ldots, n\}$) sont encore prises du parent M tel que :

   - $J_i^D = J_i^M$ avec k le plus petit indice vérifiant $J_k^M \notin \{J_1^D, \ldots, J_{i-1}^D\}$.
     De façon analogue, on construit le deuxième fils, en échangent les rôles de M et F.

L'opérateur de mutation utilisé fonctionne comme suit : pour toutes les positions $\{i = 1, \ldots, n-1\}$, les activités J_i et J_{i+1} sont échangées avec une probabilité P_{mut} si l'échange ne viole pas les contraintes de précédence.
Quatre stratégies de sélection ont été testées (sélection par rang, sélection proportionnelle, tournoi à 2 et tournoi à 3).

Si cette méthode est l'une des plus efficaces pour résoudre le *RCPSP*, ses limites quant à une application directe aux problèmes avec prise en compte des compétences sont évidentes : il s'agit ici de gérer efficacement l'affectation des personnes aux compétences.

Dans la thèse de [Bellenguez-Morineau, 2006], les auteurs ont proposé des métaheuristiques dont le codage se base sur une liste de priorité pour le problème *MSPSP*. Ils ont proposé une méthode de recherche tabou inspirée de celle de [Klein, 2000b]. L'opérateur de voisinage consiste à échanger 2 tâches dans la liste. L'échange est autorisé si (1) il respecte les contraintes de précédence et (2) si les deux tâches n'ont pas la même date de début et (3) si leur date de début ne sont pas dans l'ordre inverse des positions des tâches dans la liste de précédence. Le décodage des solutions fait appel à l'une des variations de l'heuristique proposée pour ce même problème [Bellenguez-Morineau et Néron, 2004b].
Deux algorithmes génétiques dont le codage est toujours une liste de priorité ont été proposées dans [Bellenguez-Morineau, 2006]. La première méthode utilise des opérateurs génétiques inspirés de ceux utilisés dans [Jou, 2005]. Concernant le croisement, pour reproduire un enfant à partir de deux parents, une liste de booléens est générée aléatoirement. Les positions dont les valeurs égales à zéro viennent de l'un des parents. Les autres positions sont complétées par les tâches de l'autre parent, qui ne sont pas encore dans l'individu fils en les prenant dans l'ordre où elles apparaissent dans ce parent. L'affectation de chaque tâche est recopiée avec elle. La génération de l'autre enfant se fait de la même façon en inversant les rôles des parents.
La mutation sur un individu fait une modification sur l'ordre des tâches en changeant aléatoirement deux tâches si l'échange respecte les relations de précédence. Notons que dans cette méthode les affectations des personnes aux tâches ne sont pas modifiées.
L'autre algorithme génétique proposé, utilise un codage appelé codage à affectation intégrée. Il s'agit de garder la liste de priorité comme dans la première méthode. Mais quand on croise deux individus, en plus de la liste des booléens utilisée dans la première méthode,

on utilise pour chaque tâche/compétence une liste de booléens correspondant au nombre de personnes affectées. Ainsi l'affectation des personnes n'est plus héritée du parent dont la tâche a été héritée. Elle est héritée des deux parents selon les valeurs booléennes pour chaque position. En revanche, le remplacement des personnes peut induire une infaisbilité d'affectation, un opérateur de réparation est utilisé pour éviter cette situation.
Pour muter, un individu deux probabilités sont utilisées, une probabilité pour échanger deux tâches comme dans la première méthode et une autre probabilité pour modifier l'affection d'une tâche. En fait, la modification de l'affectation se fait sur une seule personne tirée aléatoirement. Cette personne est remplacée par une autre si cet échange ne viole pas les contraintes du problème.
Dans les deux cas, la liste de priorité générée peut ne pas respecter les relations de précédence. Un opérateur de réparation est appliqué pour rendre la liste compatible avec les relations de précédence. Le décodage se fait en plaçant chaque tâche au plus tôt selon la disponibilité de ces ressources dans l'ordre de la liste de priorité. L'évaluation des individus utilise la date d'achèvement, mais la notion d'équilibrage de charge entre les personnes est utilisée pour choisir entre deux individus dont les dates d'achèvement sont égales.

Dans le problème *SWPSP* étudié par [Valls *et al.*, 2009] que nous avons présenté dans le chapitre précèdent, les auteurs proposent une approche basée sur un algorithme génétique et quelques procédures de recherche locale.
Le codage des individus se base sur une liste de priorité compatible avec les relations de précédence de composantes (appelées composantes fortes) au lieu des tâches. Comme le graphe de précédence considéré dans leur étude est cyclique, l'usage d'une liste de priorité classique est inapproprié. En se basant sur des composantes fortement connexes (chaque composante connexe est un circuit), les auteurs transforment alors le graphe de précédence en un graphe acyclique. Chaque sommet de ce graphe représente une composante connexe. Les affectations des personnes aux tâches (une personne par tâche) sont intégrées dans la solution. Le décodage d'un individu est effectué à l'aide d'une méthode de placement en série (*Serial Schedule Scheme*) appliquée sur les composantes. C'est-à-dire que les tâches d'une composante sont placées avant les tâches des composantes se trouvant après elle dans la liste de priorité. L'ordre d'exécution des tâches de la même composante est donné à l'aide d'une heuristique basée sur une règle de priorité dynamique. Pour planifier les tâches d'une composante deux valeurs sont déterminées :

1. la tâche i^* à placer : dans ce cas les tâches sont triées selon la règle $PDEST$ (*Partial Dynamic Early Start Time*). $PDEST(i)$ est la date de début au plus tôt de la tâche i calculée en prenant en compte seulement les prédécesseurs directs de i, qui sont déjà planifiés. Si aucun prédécesseur n'existe, alors $PDEST(i) = 0$,
2. la date de début t^* de i^* : pour déterminer cette valeur, on considère la première $t \geq PDEST(i)$ possible selon la disponibilité de la personne affectée à i^*, qui optimise dans l'ordre lexicographique le nombre de violation des contraintes des dates (date début minimum et maximum et date de fin minimum et maximum) et le nombre de violation des contraintes de disponibilité de la personne affectée à i^*.

Trois opérateurs de croisement ont été proposés [Valls *et al.*, 2009] :

1. croisement à un point sur l'ordre des composantes et l'affectation : ce croisement modifie la liste de priorité des composantes de la même façon que le croisement à un point [Hartmann, 1998], décrit ci-dessus. Pour modifier les affectations un autre indice q_2 est tiré aléatoirement, les affectations des premières q_2 composantes sont héritées du parent 1 et les autres sont hérités du parent 2.
2. croisement à deux points sur l'ordre des composantes et l'affectation : la modification de l'ordre des composantes dans la liste de priorité est similaire au croisement à deux points introduit dans [Hartmann, 1998]. Le changement des affectations est fait d'une manière analogique. C'est-à-dire les tâches des premières q_1 composantes héritent leur affectations du parent 1, puis les $(q_2 - q_1)$ héritent leur affectations du parent 2 et les dernières q_2 héritent leur affectations du parent 1.
3. croisement basé sur la sélection par roulette : pour chaque enfant, la sélection d'une composante est faite aléatoirement en choisissant entre les deux parents. Si par exemple, on doit sélectionner la composante L_i dans un enfant, et le résultat de tirage est le parent 1, alors $L_i = L_k$ tel que L_k est la composante du parent 1 où k est l'indice le plus petit vérifiant $L_k \notin \{L_1, \ldots, L_{i-1}\}$ (Les composantes de l'enfant déjà sélectionnées). L'affection est héritée directement du même parent que la composante.

La mutation modifie l'ordre des composantes et les affectations comme suit :
Pour chaque composante L_h, on la décale à gauche à la place de la composante L_k (le plus à gauche possible) telles que les contraintes de précédence entres les composantes restent respectées et l'intersection des listes de personnes dans les deux composantes L_h et L_k n'est pas vide. Cette modification est effectuée avec une probabilité p_{mut}.
La modification des affectations est effectuée sur chaque tâche avec la probabilité p_{mut} en choisissant aléatoirement une personne différente de l'actuelle.

L'évaluation des individus se fait en exécutant tout d'abord la procédure de placement en série, puis si l'individu n'est pas estimé de très mauvaise qualité, une recherche locale visant à optimiser des critères secondaires (violation des fenêtres de temps, violation de disponibilité des personnes) est appliquée. Cette recherche locale est basée sur les trois procédures suivantes :

1. un décalage à droite de planning : cette procédure a pour objectif de préparer le planning pour celle d'après. En fait, elle décale à droite toute tâche en avance par rapport à ses fenêtres de temps. Le décalage est effectué s'il n'implique pas de violation de contrainte de disponibilité des ressources ou de précédence.
2. des modifications sur les dates de début et sur l'affectation sont appliquées : cette procédure est une recherche locale, qui modifie les affectations et/ou les dates de début des tâches une par une afin d'améliorer la qualité de la solution. Le changement sur une tâche ne doit pas impliquer un changement sur d'autres tâches. Ces changements sont appliqués sur les composantes selon l'ordre croissant de la date de début de la première tâche, et pour les tâches de chaque composante les mouvements sont appliqués selon l'ordre croissant du numéro de la tâche. La recherche s'arrête lorsqu'aucune amélioration n'est possible.

3. un décalage à gauche du planning est appliqué à la fin, pour compacter le planning sans dégrader la qualité de la solution.

Les meilleurs individus de l'ensemble des solutions parents et enfants sont sélectionnés pour la génération suivante.

Notons que la meilleure solution trouvée par l'algorithme génétique est encore améliorée en terme de criticité (un critère basé sur les priorités des tâches donnée par les clients) en utilisant une recherche locale similaire à la procédure 2 décrite ci-dessus.

Nous nous sommes inspirés de cette méthode dans la recherche tabou présentée dans le paragraphe 3.4.4.

D'autres métaheuristiques ont été proposées pour résoudre des problèmes d'ordonnancement de projets, e.g, [Jarboui *et al.*, 2008], [Xiao *et al.*, 2013]

3.4.3 Recherche tabou et algorithme génétique pour le PMSPSP

Dans cette section, nous allons exposer les différents schémas dont la recherche tabou et l'algorithme génétique ont été développés. En effet, selon comment l'affectation est calculée, un codage des individus diffèrent est utilisé. Nous considérons donc trois approches différentes comme suit.

1. méthodes basées sur "une liste de priorité" & "affectation calculée",
2. méthodes basées sur "une liste de priorité" & "affectation héritée",
3. méthodes basées sur "une liste de priorité" & "affectation intégrée".

3.4.3.1 Méthodes basées sur une "liste de priorité" & "affectation calculée"

Dans cette partie, nous présentons des métaheuristiques dont le codage se base sur une liste de priorité, qui respecte les relations de précédence. L'affectation des voisins d'une solution dans le cas de la recherche tabou ou l'affectation des individus enfants dans le cas de l'algorithme génétique, est calculée à l'aide de l'heuristique *PAPSS* présentée dans la section 3.4.1), et est utilisée pendant la phase du décodage de la solution.

Recherche tabou : la recherche tabou est une méthode heuristique utilisée pour résoudre des problèmes combinatoires de complexité généralement *NP-difficile* tel que l'ordonnancement de projet multi-compétences. Le principe d'une recherche tabou est d'améliorer une solution de départ en y appliquant des mouvements comme dans les méthodes de recherche locale. Contrairement aux méthodes de descente locale classique, une recherche tabou ne s'arrête pas lorsqu'elle atteint un optimal local. Elle utilise une mémoire (liste tabou) pour pouvoir échapper aux optima locaux. Cette méthode a montré son efficacité pour trouver des solutions de bonne qualité en un temps de calcul raisonnable.

Nous proposons ici une méthode tabou pour la résolution du problème *PMSPSP*. Les composantes principales de cette méthode sont décrites comme suit.

1. **Représentations des individus** : un individu est représenté sous la forme d'une liste de tâches dans un ordre respectant les contraintes de précédence. Un élément de la liste représente seulement le numéro de la tâche comme dans la figure 3.5.

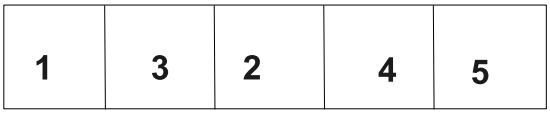


FIGURE 3.5 – Exemple du codage d'une solution basé sur une liste de priorité

2. **Solution initiale** : la solution de départ utilise la méthode *PAPSS* avec l'une des règles de priorité. Si cette solution n'est pas réalisable (non respect de la date de fin imposée), une autre solution générée aléatoirement est utilisée. La génération d'une solution aléatoire est éventuellement nécessaire, car l'heuristique ne calcule pas d'affectation aux tâches qu'elle n'a pas pu placer.

3. **Voisinage** : le voisinage utilisé est celui proposé dans [Hartmann, 1998]. Il s'agit de sélectionner deux indices i et j ($i < j$) dans la liste de priorité, puis faire un échange entre les deux sous-séquences extraites, si cela respecte les relations de précédence. Pour éviter de tester des mouvements non réalisables, nous cherchons les deux indices entre i et j les plus proches de ces dernières tel qu'un échange est possible.

4. **Liste tabou :** afin d'éviter de visiter la même solution plusieurs fois durant un certain nombre d'itérations, soit une liste de solutions tabou ou une liste de mouvements tabous est utilisés (les deux peuvent être utilisées simultanément). Dans notre cas, une liste conservant les derniers mouvements effectués est utilisée (le nombre de mouvements conservés est fixé expérimentalement). Très couteux en terme de temps de calcul, l'usage d'une liste de solutions tabou a été exclu.

5. **Diversification et intensification** : dans les métaheuristiques, on utilise un mécanisme, qui a pour objectif de permettre à la méthode de s'échapper aux optima locaux. Ce mécanisme appelé *diversification*, s'emploie après un certain nombre d'itérations sans que la solution courante soit améliorée. Dans notre cas, un paramètre (DiversifFreq) est introduit. Après *DiversifFreq* itérations sans amélioration de la solution courante, nous réinitialisons la solution courante par une solution calculée en utilisant le même algorithme qui a permis de générer la solution initiale. 11 règles sont utilisées dans l'ordre suivante (*MAXNBSKILL, CRITICITY, ERD, EDD, MAXCHARG, MTS, RAND, MINSLK, GRNK, GRNK1, GRNK2*) (cf Section 3.4.1). Après l'utilisation de toutes ces règles, la règle *RAND* est utilisée pour diversifier la solution courante.
   Quant à l'intensification, elle est utilisée dès que la solution en cours est améliorée. A partir du moment où les voisins d'une solution sont prometteurs, on peut espérer que l'on est proche d'un optimum global et par conséquent, on essaie d'intensifier la recherche autour de cette solution. Comme dans [Bellenguez-Morineau, 2006], nous augmentons le nombre de voisins à explorer. Pour cela, nous avons deux paramètres :

l'un pour le nombre de voisins à explorer dans une phase d'intensification (*MaxToVisit*) et l'autre dans le cas normal (*NeighbToVisit*).

Algorithme général de la Recherche tabou : l'algorithme 3, décrit le schéma général de la recherche tabou développée. La table 3.6 donne la définition de différents paramètres utilisés dans l'algorithme.

Paramètre	*Définition*
NbStep	nombre d'itérations. Il est utilisé dans le cas où le critère d'arrêt est le nombre d'itérations
TabSolSize	la taille de la liste tabou des solutions
TabMoveSize	la taille de la liste tabou des mouvements
NBMinute	limite de temps en minute, sa valeur peut être moins de 1. Il est utilisé dans le cas où le critère d'arrêt est le temps passé
MaxIterWithoutImpr	le nombre d'itérations sans améliorations. Il est utilisé comme critère d'arrêt
DiversifFreq	le nombre d'itérations sans améliorations après lequel une diversification doit être appliquée
UseDiver	un booléen pour préciser si la diversification doit être faite ou non
UseIntens	un booléen indiquant si une intensification doit être faite ou non
NeighbToVisit	le nombre de voisins à générer lors d'une phase normale (sans intensification)
MaxToVisit	le nombre de voisins à générer lors d'une phase d'intensification
SolTabouList	liste des solutions tabou
MoveTabouList	liste des mouvements tabou

Tableau 3.6 – Paramètres de la recherche tabou

Afin de pouvoir comparer la méthode tabou (Algorithme 3) avec d'autres approches, nous avons développé un algorithme génétique qui utilise des opérateurs de croisement et de mutation inspirés des mouvements utilisés dans cette méthode. Nous présentons dans ce qui suit cet algorithme génétique.

Algorithme génétique : cette approche utilise le même codage que la recherche tabou. Les opérateurs génétiques utilisés sont aussi inspirés des mouvements utilisés dans la méthode tabou. Le décodage des solutions utilise la même heuristique que la recherche tabou. Cependant des paramètres liés à l'algorithme génétique sont définis comme suit :

- Taille de population (*Taille_Pop*) : ce paramètre est très important dans un algorithme génétique. Il s'agit du nombre de solutions, qui constituent une génération donnée. C'est à partir de ces solutions que les solutions de la génération suivantes vont être reproduites. Mais un compromis entre le temps de calcul et la qualité des solutions que nous espérons obtenir doit être pris en compte. C'est pour cela que nous

Algorithme 3: Structure générale de la méthode Tabou

$S_{cour} \leftarrow PAPSS(EDD)$
$S_{best} \leftarrow S_{cour}$
$intensify \leftarrow true$
$stepWithoutEnh \leftarrow 0$
tantque critère d'arrêt n'est pas vérifiée **faire**
  $Nei \leftarrow getNeiberhood(S_{cour}, intensify)$
  $S_{cour} \leftarrow best(Nei)$
  si $S_{cour}.C_{max} < S_{best}.C_{max}$ **alors**
    $S_{best} \leftarrow S_{cour}$
    $stepWithoutEnh \leftarrow 0$
  sinon
    $stepWithoutEnh++$
    $intensify \leftarrow false$
  finsi
  si $tabouList.size = TABOULISTSIZE$ **alors**
    $tabouList.remove(0)$
  finsi
  $tabouList.add(S_{Cour})$
  si $stepWitEnh \neq 0et(stepWitEnhmodMaxStepWTHENH = 0)$ **alors**
    $S_{Cour} \leftarrow diversiy()$
  finsi
fin tantque
return S_{best}

avons lié ce paramètre à la taille du problème, notamment avec le nombre de tâches du projet. Après des tests sur des instances de taille entre 10 et 60 tâches, nous avons fixé ce nombre à : $\lfloor \frac{n}{2} \rfloor + \alpha$ avec $\alpha = 1$ si n est impair, sinon $\alpha = 0$. C'est-à-dire la taille de population est le plus petit nombre entier pair supérieure ou égal à $n/2$.

- Population initiale : pour générer une population initiale, nous utilisons les dix règles de priorité (cf 3.4.1). Chaque règle génère un individu (solution), puis le $(Taille\_Pop - 10)$ individus restant sont générés aléatoirement.
- Croisement et mutation : nous utilisons le croisement à deux points et la mutation introduits dans [Hartmann, 1998], décrit en section 3.4.3.
- Une sélection par tournoi à 2 est utilisée afin de remplacer les solutions courantes, ainsi que dans la sélection des parents pour la reproduction des nouveaux individus.

Comme dans la méthode tabou, il se peut que notre population soit piégée par des optimaux locaux. Afin de sortir de cette situation, nous remplaçons la moitié des solutions dans la génération courante par des solutions générées aléatoirement. Ce remplacement se fait après un nombre d'itérations sans amélioration de la solution.
La structure générale de cet algorithme est donnée dans l'algorithme 4.

Algorithme 4: Structure générale de l'algorithme génétique

$Taille\_Pop$: taille de la population
$0 \leq cProb \leq 1$: probabilité de croisement
$0 \leq mProb \leq 1$: probabilité de mutation
$P(i)$: la population de la génération i
i : indice des générations
$i \leftarrow 0$
générer aléatoirement $Taille\_Pop$ individus et les mettre dans $P(i)$
$BestSol \leftarrow$ Meilleure solution de $P(i)$
tantque critère d'arrêt n'est pas vérifiée **faire**
  $P'(i) \leftarrow \emptyset$ // population à reproduire de la population $P(i)$
  tantque le nombre d'individus dans P'(i) $< Taille\_Pop$ **faire**
    choisir I_1 et I_2 de la population P(i) tel que $I_1 \neq I_2$
    $F \leftarrow Tournoi(I_1, I_2)$ // choix du parent 1
    choisir J_1 et J_2 de la population P(i) tel que $J_1 \neq J_2 \neq F$
    $M \leftarrow Tournoi(J_1, J_2)$ // choix du parent 2
    `/* Croisement des deux parent selon la probabilité cProb`
    si M et F ne sont pas encore croisés **alors**
      tirer aléatoirement un réel c $(0 \leq c \leq 1)$
      si $c < cProb$ **alors**
        croiser M et F // faire le croisement
        $S \leftarrow$ premier enfant de M et F
                `/* muter S avec la probabilité mProb`

        tirer aléatoirement un réel m_1 $(0 \leq m_1 \leq 1)$
        si $m_1 < mProb$ **alors**
          $S \leftarrow$ Mutation de S // faire la mutation selon la probabilité mProb
        finsi
        $D \leftarrow$ deuxième enfant de M et F
        tirer aléatoirement un réel m_2 $(0 \leq m_2 \leq 1)$
        si $m_2 < mProb$ **alors**
          $D \leftarrow$ Mutation de D // faire la mutation
        finsi
      sinon
        $S \leftarrow$ Mutation de F
        $D \leftarrow$ Mutation de M
      finsi
      $P'(i) \leftarrow P'(i) \cup \{S, D\}$ // ajouter les deux nouvelles individus à la population enfant
    finsi
  fin tantque
   `\* Sélection de la prochaine génération par tournoi`
  $Q \leftarrow P(i) \cup P'(i)$
  $i \leftarrow i + 1$
  $P(i) \leftarrow \emptyset$
  tantque le nombre d'individus de $P(i)$ $< Taille\_Pop$ **faire**
    choisir I_1 et I_2 de la population Q tel que $I_1 \neq I_2$
    $K \leftarrow Tournoi(I_1, I_2)$ // choix de l'individu qui reste dans la génération suivante
    si K est meilleur que $BestSol$ **alors**
      $BestSol \leftarrow K$ // mettre à jour BestSol
    finsi
    $P(i) \leftarrow P(i) \cup \{K\}$
  fin tantque
fin tantque
retourner $BestSol$

3.4.3.2 Méthodes basées sur "une liste de priorité" & "une affectation héritée"

Dans les deux méthodes présentées dans la section 3.4.3.1, le codage se base sur une liste de priorité qui respecte les relations de précédence et leur décodage fait appel à la méthode *PAPSS* pour déterminer l'affectation des personnes aux tâches (cf paragraphe 3.4.1).
Dans cette partie, une variation est utilisée en recopiant l'affectation de chaque tâche directement de la solution originale dans le cas de la méthode tabou. Dans le cas de l'algorithme génétique, l'affectation utilisée est héritée du parent dont est issue la tâche. Cette approche est inspirée de l'algorithme génétique [Bellenguez-Morineau et Néron, 2004a] appelée *Placement parallèle avec affectation héritée* présenté ci-dessus. La version utilisée de l'heuristique ne résout pas le problème d'affectation, vue que celle-ci est donnée pour chaque tâche. Le voisinage et les opérateurs génétiques utilisés sont identiques à ceux utilisés dans les deux méthodes présentées dans la section précédente.

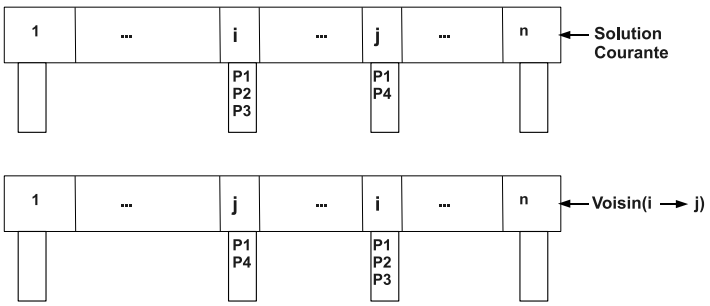


FIGURE 3.6 – Voisinage à affectation héritée

Notons que pour ces deux métaheuristiques, les affectations sont celles construites avec la solution initiale (ou dans la population initiale), ou bien lors du remplacement des solutions suite à l'appel de la procédure de diversification.

3.4.3.3 Méthodes basées sur une "liste de priorité" & "une affectation intégrée"

Dans ces méthodes, l'affection est intégrée au codage des solutions. Par conséquent, les opérateurs de voisinage ou les opérateurs génétiques (croisement et mutation) interviennent au niveau de l'affectation aussi bien que sur l'ordre des tâches dans la liste de priorité. Le décodage d'une solution avec cette représentation fait appel à la méthode *PAPSS* tout en conservant l'affectation établie.

Recherche tabou : la structure générale, ainsi que les paramètres utilisés sont identiques à ceux présentés dans la section 3.4.3.1. En revanche, l'opérateur de voisinage utilisé est diffèrent. Par cette approche, l'échange de deux tâches dans la liste de priorité est réalisé s'il est faisable, de plus, on reconstruit la liste des personnes affectées à chaque tâche en se basant sur l'affectation d'une autre.

Voisinage : deux indices différents i, et j sont tirées aléatoirement avec $(i < j)$. Si l'échange entre les deux tâches A_i et A_j dont les positions dans la liste de priorité sont respectivement i et j est possible, nous les échangeons puis l'affectation de chacune est modifiée comme suit : soit TP_i la liste des personnes affectées à la tâche A_i triées dans l'ordre croissant des indices des compétences qu'elles exercent et TP_j celles de la tâche A_j. Soit $Diff_i = A_j \backslash A_i$ l'ensemble de personnes affectées à la tâche A_j qui ne sont pas affectées à la tâche A_i et $Cand_i = Diff_i \cup TP_i$ la liste des personnes issues de la concaténation des deux listes $Diff_i$ et TP_i dans l'ordre. Pour constituer la nouvelle affectation de A_i, on prend les compétences dans l'ordre et pour chaque compétence on affecte la première personne P dans la liste $Cand_i$ capable de la faire. Puis on enlève cette personne de la liste $Cand_i$. De cette façon, une nouvelle affectation réalisable garantissant le respect des compétences, ainsi que la contrainte "une personne une compétence", est obtenue. On fait la même procédure pour la tâche A_j. La figure 3.7, montre un voisin d'une solution où les deux indices sélectionnés sont 2 et 3.

Cet opérateur de voisinage a l'avantage de ne pas nécessiter une réparation pour avoir une affectation réalisable.

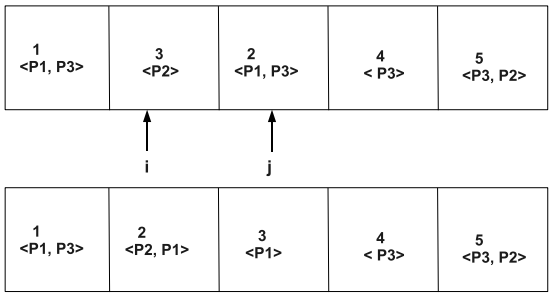


FIGURE 3.7 – Exemple de voisinage à affectation intégrée

Algorithme génétique : l'algorithme génétique que nous présentons ici, se distingue des autres algorithmes génétiques décrits ci-dessus par l'intégration de l'affectation dans le codage de la solution. Ainsi, ces deux méthodes sont très dépendantes de la qualité des affectations calculées par l'algorithme de placement. En effet, les deux opérateurs génétiques croisement et mutation, utilisent toujours les mêmes modifications sur les tâches (croisement à deux points et échange des tâches lors de la mutation). Par contre, pour le croisement, nous utilisons le même mécanisme que celui utilisé dans la recherche tabou,c'est-à-dire que pour chaque tâche venant du parent M, nous associons une affectation calculée comme dans le mouvement de la méthode tabou, en utilisant la tâche à la même position dans l'autre parent. De la même façon, lors d'une mutation on échange deux positions entre elles. Nous utilisons une autre probabilité pour appliquer ou non le même mécanisme que dans le croisement. Les autres paramètres, ainsi que le décodage des solutions sont identiques à la méthode précédente. Une exemple de croisement est illustré en Figure 3.8. Dans cet exemple, la tâche 1 nécessite les compétences 1 et 2, la tâche 2, nécessite les compétences 2 et 3, les tâches 3 et 4 nécessitent respectivement les compétences 1 et 2, et

la tâche 5 nécessite les compétences 1 et 3. Les personnes $(P1, P2)$ maîtrisent la compétence 1, les personnes $(P1, P2, P3, P4)$ maîtrisent la compétence 2 et la compétence 3 est maîtrisée par les personnes $(P1, P2, P3)$.

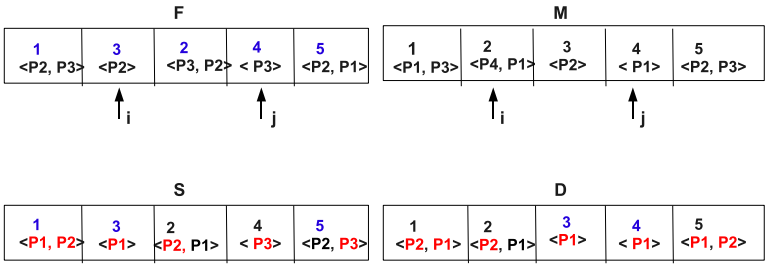


FIGURE 3.8 – Exemple de croisement à deux points avec changement des affectations

3.4.4 Recherche tabou à codage direct pour le PMSPSP

Dans cette partie, nous présentons une recherche tabou dont le codage se base sur les périodes d'exécution des tâches avec une affectation intégrée. La solution initiale est calculée comme dans les autres méthodes par l'heuristique *PAPSS*. Les mouvements se basent sur des décalages par tâche, tâche/compétence ou des décalages globaux, ainsi que des changements d'affectation.

L'originalité de cette méthode est d'autoriser la construction des solutions éventuellement non réalisables.

Codage d'une solution : une solution est représentée par une liste des dates de début de chaque tâche. Pour les tâches préemptives, une liste de périodes d'exécution est associée à chaque tâche/compétence lorsque le besoin est supérieur à une période de temps. L'affectation est intégrée au codage comme dans les méthodes présentées dans les paragraphes précédents. Une liste de personnes affectées à chaque tâche est utilisée.

Mouvements : les mouvements utilisés sont de deux types. Des mouvements basés sur le temps où il s'agit de décaler une tâche à gauche ou à droite, ou bien décaler à gauche ou à droite une partie d'une tâche préemptive. Un décalage de ce type n'est autorisé que s'il respecte les contraintes de précédence, mais il peut ne pas respecter les contraintes de ressources. Le deuxième type de mouvement interagit sur les affectations. Il s'agit de remplacer une personne affectée à une tâche par une autre personne. Ce mouvement n'est autorisé que si la nouvelle personne maîtrise la compétence concernée, et n'est pas déjà affectée à la tâche.

Décodage et évaluation : avec le codage proposé ici, nous n'avons pas besoin d'une procédure de décodage, car les fenêtres d'exécution, ainsi que l'affection des tâches sont

connues. En revanche, une évaluation est nécessaire. Notons que les solutions ne sont pas toutes réalisables, car le respect des disponibilités des personnes peut être violé par les mouvements de type décalage. L'évaluation des solutions prend comme premier critère le nombre de contraintes violées puis le C_{max} de la solution. Nous utilisons la fonction suivante : $(HZ \cdot NumberOfViolated + C_{max})$. Cette évaluation assure que toute solution S_1 dont le nombre de contraintes violées est supérieur au nombre de contraintes violées dans la solution S_2, est moins bonne que la solution S_2. Pour le calcul du C_{max}, nous n'avons besoin de connaitre que la date de fin maximum dans la solution courante. Pour chaque voisin généré par un décalage, nous appliquons les règles suivantes :

- si le décalage concerne une compétence, qui ne se terminait pas avec la fin du projet, nous testons si sa nouvelle date de fin est supérieure au C_{max} actuel. Si c'est le cas, nous actualisons le C_{max} par la date de fin de cette compétence et nous la mettons seule dans la liste des compétences marquant la fin du projet. Si sa date de fin est égale au C_{max}, nous la rajoutons dans la liste des tâches/compétences, qui marquent la fin du planning ;
- si le décalage concerne une compétence, qui était seule dans la liste des compétences de fin de projet et si sa nouvelle date de fin dépasse le C_{max}, on met à jour le C_{max}. Si sa nouvelle date de fin est plus petite que celle dans la solution courante, nous recalculons de nouveau le C_{max} et mettons à jour la liste des compétences, qui marquent la fin du projet ;
- si un décalage concerne une compétence dans la liste des compétences de fin du projet et si sa nouvelle date de fin est plus grande que le C_{max}, elle deviendra la seule dans la liste et le C_{max} est ajusté. Si sa date est devenue strictement inférieure à C_{max}, elle est supprimée de la liste.

Concernant le calcul du nombre de contraintes violées au sens de la disponibilité, partant d'une solution S_0, nous avons, pour chaque personne/période, la charge qui lui est affectée (peut être supérieure à 1). Pour chaque voisin issu d'un mouvement de type décalage ou changement d'affectation, ces charges sont ajustées. Le nombre de contraintes violées et égal au nombre de contraintes violées dans la solution S_0 plus ou moins le nombre de violations générées ou réparées par ce mouvement.

Procédures d'amélioration de faisabilité : suite à nos expérimentations, cette méthode ne converge pas rapidement vers des bonnes solutions. La difficulté vient principalement de la faisabilité, car les mouvements mentionnés ci-dessus, n'intègrent pas de mécanisme pour éviter d'appliquer des mouvements générateurs d'infaisabilité. Nous avons donc rajouté une recherche locale inspirée du [Valls *et al.*, 2009] afin de pallier à ce problème. La procédure est appliquée après un nombre d'itérations sans amélioration sur la faisabilité de la solution courante. Elle se résume en quatre phases : (1) un décalage à droite de toutes les tâches avec une période δ ; (2) des changements d'affectation sont appliqués sur les tâches afin de minimiser le nombre de violation de contraintes ; (3) un décalage à gauche de toutes les tâches est appliqué : on n'autorise aucun décalage qui augmente le nombre de contraintes violées. (4) nous ré-appliquons la même procédure dans la phase 2.

Diversification et intensification : la diversification suit le même mécanisme que dans les autres méthodes. Il s'agit de réinitialiser la solution courante en utilisant une règle de

priorité parmi les règles décrites dans la section 3.4.1. Cette initialisation est faite après un nombre d'itérations sans amélioration sur le C_{max}. Pour l'intensification, nous utilisons toujours la même approche : un nombre de voisins plus élevé sont explorés lors de cette phase.

3.4.5 Résultats expérimentaux

Dans cette partie, nous présentons les résultats expérimentaux des méthodes approchées présentées dans les sections précédentes.

3.4.5.1 Résultats de l'algorithme *PAPSS*

Dans cette section, nous présentons une comparaison entre l'algorithme *PAPSS* en utilisant la règle de priorité (*EDD*) et le modèle linéaire et la borne inférieure du raisonnement énergétique adapté(LB_{ER}). La comparaison avec le modèle linéaire concerne seulement les instances résolues de manière optimale. Les instances utilisées ici sont celles présentées dans la section 3.3.3. La colonne 3 donne le nombre de solutions optimales, colonne 4 donne la déviation entre le *PLNE* et LB_{ER}. Tandis que les dernières colonnes donnent respectivement la déviation entre l'heuristique et le *PLNE* et l'heuristique et la borne inférieure. Les déviations sont calculées en utilisant seulement les instances pour lesquelles le modèle a trouvé des solutions optimales.
Dans les tables 3.9 et 3.10, nous comparons l'heuristique avec la borne inférieure (LB_{ER}), en utilisant toutes les instances de 10, 16 et 20 tâches. La colonne **gap** donne la déviation entre la borne supérieure calculée par l'heuristique et la borne inférieure, alors que la colonne $\#\mathbf{(UB = LB)}$ donne le nombre de fois où le C_{max} calculé par l'heuristique est égal à la valeur de la borne inférieure. Dans la dernière colonne, nous donnons le temps de calcul.

# tâche	# compétence	# opt	(Opt-LB)/LB (%)	(UB-Opt)/opt (%)	(UB-LB)/LB (%)
10	4	10	10,5	17,7	30,0
	5	8	21,3	13,4	32,7
16	4	5	5,9	20,6	32,6
	5	1	4,1	21,8	50,4
20	4	3	22,4	7,0	28,6
	5	0	-	-	50,5

Tableau 3.7 – 10 instances de 6 personnes, 50% de préemption, 75% de disponibilité

Dans le tableau 3.9 (pas d'indisponibilité), environ 23% des instances étaient résolues ($UB = LB$). La déviation est inférieure à 10% pour les instances à 4 compétences, et la déviation maximum (20.83%) est atteinte pour les instances à 20 tâches, 5 compétences.

# tâche	# compétence	# opt	(Opt-LB)/LB (%)	(UB-Opt)/opt (%)	(UB-LB)/LB (%)
10	4	9	2,1	5,3	8,7
	5	10	1,5	6,9	11,3
16	4	9	0,6	6,2	6,5
	5	6	1,8	7,1	13,5
20	4	9	0	7,2	9,2
	5	2	3,8	7,6	20,8

Tableau 3.8 – 10 instances de 6 personnes, 50% de préemption, 100% de disponibilité

# tâche	# compétence	gap (%)	#(UB = LB)	UB temps (sec)
10	4	8,7	5	0,06
	5	11,3	3	0,05
16	4	6,5	2	0,03
	5	13,5	0	0,05
20	4	9,2	4	0,02
	5	20,8	0	0,03

Tableau 3.9 – UB vs LB : 10 instances de 6 personnes, 50% de préemption, 100% de disponibilité

Alors que dans le cas d'indisponibilité de personnes (Tableau 3.10) la déviation devient plus important même avec le nombre de compétences de 4.

Analysons maintenant les performances de *PAPSS* en considérant les règles de priorité (cf Section 3.4.1). A cause du nombre très élevé de paramètres du problème, nous nous limitons ici à l'impact des deux paramètres PA et APA que nous allons faire varier en fixant les autres paramètres.

1. Variation de (APA)
   Dans le tableau 3.11, trois groupes de jeux de données de 160 instances chacune sont utilisées. On utilise les premières 160 instances de 30 tâches, de la *PSPLIB*. Pour chaque groupe, ($NR = 10, NS = 5, SpP = 50, SpT = 25, PA = 25\%$). Les valeurs de la disponibilité moyenne (APA) sont respectivement, 95%, 75% et 50%. Pour chaque groupe on présente la déviation moyenne par rapport à la solution optimale pour chaque règle (Gap(%)) et le temps de calcul moyen en secondes (Time(S)).
2. Différents pourcentages de préemption (PA)
   Dans le tableau 3.12, 5 groupes de données de 160 instances pour chacune sont utilisés. On utilise les premières 160 instances de 30 tâches de la *PSPLIB*. Pour chaque groupe, nous avons : $NR = 10, NS = 5, SpP = 50, SpT = 25, APA = 95\%$. Les taux moyens de pourcentage (PA) utilisés sont respectivement, 0%, 25%, 50%, 75% et 100%. Pour chaque groupe, on présente la déviation moyenne de chaque règle (Gap(%)) par rapport à l'optimal et le temps de calcul moyen en secondes (Time(S)).

# tâche	# compétence	gap (%)	#(UB = LB)	UB temps (sec)
10	4	30	1	0,03
	5	32,7	0	0,04
16	4	32,6	0	0,03
	5	50,4	1	0,05
20	4	28,6	2	0,03
	5	50,5	0	0,04

Tableau 3.10 – UB vs LB : 10 instances de 6 personnes, 50% de préemption, 75% de disponibilité

priorité	Avail =95%		Avail =75%		Avail =50%	
	Gap (%)	*Temps(s))*	*Gap (%)*	*Temps(s))*	*Gap (%)*	*Temps(s))*
CRITICITY	**5,72**	0,01	7,94	0,12	10,36	0,04
EDD	**5,62**	0,01	9,16	0,13	**8,18**	0,08
ERD	6,34	0,01	8,27	0,12	9,08	0,05
GRNK	9,48	0,01	9,85	0,12	9,08	0,05
GRNK1	6,59	0,01	9,32	0,12	8,75	0,05
GRNK2	9,74	0,01	9,69	0,12	8,82	0,04
MAXCHRG	9,95	0,01	**4,48**	0,12	**8,74**	0,05
MAXNBSKIL	**5,2**	0,01	**4,2**	0,12	**8,59**	0,04
MINSLK	11,5	0,01	9,71	0,12	9,55	0,05
MTS	7,17	0,01	10,01	0,12	9,03	0,05
RAND	15,7	0,01	**5,75**	0,12	11,12	0,04

Tableau 3.11 – Règles de priorité avec plusieurs taux de disponibilités

Dans les deux tableaux 3.12 ,3.11, nous signalons en gras les premières 3 meilleures règles de priorité en termes de qualité de solution (*Gap*(%)). La première remarque concerne la règle *MAXNBSKIL*. Nous pouvons voir que cette règle apparaît dans les trois meilleures méthodes dans tous les groupes à l'exception d'un seul groupe ou la préemption n'est pas autorisée. Mais dans ce cas, les déviations de toutes les règles sont proches et sont toutes en dessous de 5%. Dans 5 groupes, cette règle a été la meilleure parmi toutes.
Dans la table 3.12, on peut remarquer que la règle *MAXNBSKIL* n'était pas vraiment dépendante de la préemption (ses déviations étaient en dessous de 6% dans tous les groupes). Lorsque le taux moyen de disponibilité change, on ne voit pas un impact sur *MAXNBSKIL* (ces rangs sont respectivement : 1, 2, 2 dans les trois groupes comme indiqué dans la table 3.11).
Une autre remarque importante concerne la règle de priorité *CRITICITY*. Elle est dans les trois meilleures méthodes dans 4 groupes sur 8. Comme *MAXNBSKIL*, le taux de disponibilité APA n'avait pas d'impact réel sur cette règle en terme du rang. Cependant nous avons une augmentation de la déviation lors de l'augmentation du taux de préemption. Contrairement à la règle *MAXNBSKIL*, *CRITICITY* sort des trois meilleures méthodes lorsque le taux de disponibilité augmente (les dernières 2 groupes dans le tableau 3.11).
Comme on peut le voir dans les tables 3.11 et 3.12, les règles GRNK, GRNK1,GRNK2, MINSLK et MTS n'apparaissent pas dans les trois meilleures méthodes sauf dans le pre-

priorité	PA =0%		PA =25%		PA =50%		PA =75%		PA =100%	
	Gap (%)	*Temps(s)*	*Gap (%)*	*Temps(s)*	*Gap (%)*	*Temps(s)*	*Gap (%)*	*Temps(s)*	*Gap (%)*	*Temps(s)*
CRITICITY	4,6	0,02	**5,72**	0,01	**5,98**	0,01	**6,65**	0,01	**8,86**	0,01
EDD	**1,31**	0,02	**5,62**	0,01	10,1	0,02	14,51	0,01	18,43	0,02
ERD	4,79	0,017	6,34	0,014	**7,53**	0,01	**9,51**	0,02	**12,86**	0,01
GRNK	**3,34**	0,02	9,47	0,01	14,72	0,01	24,74	0,01	29,32	0,01
GRNK1	4,28	0,01	6,59	0,01	10,45	0,01	13,84	0,01	16,77	0,01
GRNK2	3,57	0,01	9,74	0,01	14,42	0,01	26,71	0,01	30,04	0,01
MAXCHRG	4,04	0,02	9,95	0,01	18,34	0,01	25,69	0,02	32,84	0,02
MAXNBSKIL	4,77	0,01	**5,2**	0,01	**5,47**	0,01	**5,96**	0,01	**5,16**	0,01
MINSLK	**1,66**	0,016	11,5	0,01	21,87	0,01	29,42	0,01	34,46	0,01
MTS	4,19	0,01	7,17	0,01	9,63	0,01	14,94	0,01	18,34	0,01
RAND	4,45	0,01	15,7	0,01	24,22	0,01	32,85	0,01	39,21	0,01

Tableau 3.12 – Règles de priorités avec plusieurs taux de préemption

mier groupe du 3.12 où toutes les règles étaient bonnes.
La règle de priorité *MAXCHRG* donne des mauvais résultats lors de l'augmentation des taux de préemption. Mais si le taux d'indisponibilité augmente, l'impact n'est pas très important et la déviation reste en dessous de 10% (cf Tableau 3.12).
Pour la règle *EDD*, le taux de disponibilité n'avait pas d'impact important, elle apparait dans les trois meilleures méthodes, trois fois. Mais à partir des résultats dans la table 3.12 l'impact du taux de préemption est important.
Pour la règle de priorité *ERD*, la différence principale par rapport à *EDD* est l'impact de taux de préemption qui n'était pas significatif dans le cas de la règle *ERD*.
Enfin, on peut dire que ces règles sont en tout cas bénéfiques par rapport à un ordre aléatoire. La déviation dans le cas de la règle *RAND* est souvent très élevée (entre 11% et 40% à l'exception de deux groupes).

L'impact du taux de disponibilité sur le temps de calcul peut être vu avec la table 3.12. Le temps de calcul augmente avec la diminution du taux de disponibilité. Cela est principalement dû au fait que le problème d'affectation est résolu plusieurs fois pour chaque tâche quand les ressources ne sont pas très disponibles. Nous remarquons aussi une légère amélioration du temps de calcul lors de l'augmentation du taux de préemption. Cela peut être expliqué par le fait qu'avoir beaucoup de tâches préemptives peut faire diminuer le nombre de fois qu'une affectation non réalisable est rencontrée lors de la résolution du problème d'affectation d'une tâche.

On constate aussi que le temps de calcul est plus faible (dans 90% des cas) quand on utilise la règle de priorité *CRITICITY*. Nous pouvons déduire ce résultat de la même façon dans le cas du taux de disponibilité. Il apparait que l'usage des indicateurs de criticité aide à diminuer le nombre de fois que nous essayons de résoudre le problème d'affectation sans réussite.

En conclusion, même si cette heuristique arrive à trouver de bons résultats dans certains cas, nous trouvons que la qualité des solutions reste assez modeste en termes de déviation par rapport à la borne inférieure. C'est pour cela que des métaheuristiques ont été proposées. Nous présentons ci-dessous les résultats obtenus par ces méthodes.

3.4.5.2 Performances des métaheuristiques

Dans cette partie nous présentons les résultats expérimentaux des métaheuristiques proposées. Nous comparons tout d'abord la qualité des solutions de la méthode tabou à codage basé sur la liste de priorité et l'algorithme génétique avec le même codage (tableau 3.13), en utilisant les déviations de chaque méthode par rapport à la borne inférieure (LB_{ER}). Les mêmes méthodes sont comparées en termes de temps d'exécution (tableau 3.14). Dans le tableau 3.15, nous présentons les résultats de la méthode tabou présentée dans la partie 3.4.4.
Après des expérimentations menées sur plusieurs instances, nous avons utilisé 150 itérations sans amélioration comme critère d'arrêt pour les métaheuristiques à codage indirect (recherche Tabou et algorithme génétique) et une liste tabou de taille 20 pour la recherche tabou. Ainsi, dans le cas de la recherche Tabou le nombre de voisins à explorer lors d'une phase normale $NeighbToVisit$ est fixé à n (n est le nombre de tâches) et le nombre de voisins à explorer lors de l'intensification $MaxToVisit$ est fixé à $2n$.
Pour les deux approches recherche tabou et algorithme génétique, le nombre d'itérations sans amélioration après lequel une diversification est appliquée $DiversifFreq$ est fixé au n.
Concernant la recherche tabou à codage direct, un nombre d'itérations sans amélioration égal à 400 est utilisé, et la taille de la liste tabou est fixé à 50. les paramètres d'intensification et de diversification sont identiques à ceux utilisées pour les autres méthodes.

Dans les deux tableaux (Tableau 3.13 et Tableau 3.14), les colonnes : Affect Cal, Affect Her et Affect Integ représentent respectivement les méthodes à affectation calculée, héritée et intégrée.

Groupe	Méthode Tabou (%)			Algo Génétique (%)		
	Affect Cal	Affect Her	Affect Integ	Affect Cal	Affect Her	Affect Integ
10 tâches	9,6	15,9	14,93	8,22	15,17	16,22
16 tâches	15,05	19,78	22,63	22,13	22,34	18,09
20 tâches	14,66	24,02	23,51	17	19,24	21,89
30 tâches	2,27	5,36	3,99	2,39	3,41	33,62
60 tâches	42,13	51,78	51,24	-	51,33	51,52

Tableau 3.13 – Qualité des solutions : Tabou1 vs AG en utilisant l'heuristique *PAPSS*

Les résultats du tableau 3.13 et tableau 3.14 amènent à quelques remarques :

Groupe	Méthode Tabou			Algo Génétique		
	Affect Cal	Affect Her	Affect Integ	Affect Cal	Affect Her	Affect Integ
10 tâches	1,84	0,230	0,45	15,97	2,75	2,30
16 tâches	3,71	0,85	0,91	28,73	4,9	5,11
20 tâches	7,12	1,22	1,18	51,02	6,83	7,25
30 tâches	8,61	4,06	5,56	42,95	17,20	17,72
60 tâches	1325,02	33,46	27,57	-	395,34	379,02

Tableau 3.14 – Temps d'exécution : Tabou 1 vs AG en utilisant l'heuristique *PAPSS*

Groupe	Déviation(%)	Temps
10 tâches	19,78	2,73
16 tâches	19,58	5,56
20 tâches	27,51	8,84
30 tâches	6,08	9,10
60 tâches	54,12	126,42

Tableau 3.15 – Résultats expérimentaux de la méthode tabou à codage direct

1. Le codage à affectation calculée coûte trop de temps, que ce soit avec la méthode tabou ou l'algorithmique génétique. Il donne cependant des solutions de bonne qualité par rapport aux deux autres approches. Cela montre que les indicateurs de criticité sont capables de trouver des bonnes solutions ;

2. Le codage à affectation intégrée est meilleur en termes de qualité de solution que le codage à affectation héritée, à l'exception de quelques cas. En revanche, le temps de calcul est moins important dans le cas d'une affectation héritée. Cela peut s'expliquer par la nature du voisinage et les opérateurs génétiques utilisés avec l'affectation intégrée ;

3. Nous pouvons constater aussi que les algorithmes génétiques nécessitent beaucoup plus de temps que les méthodes tabou. Malgré ce temps, les algorithmes génétiques ont donné des résultats moins bons pour les instances de taille 30 et 60.

Les résultats présentés dans le tableau 3.15, montrent que la méthode tabou à voisinage direct n'est pas efficace par rapport aux autres méthodes de codage indirect. Remarquons que le temps de calcul soit très faible par rapport aux codages indirects (le temps de calcul après 400 itérations sans amélioration est moins important que la méthode la plus rapide parmi celles utilisant un codage indirect et avec seulement 150 itérations sans amélioration du résultat). Le constat que nous avons établi est que le nombre de voisins non faisables en termes de violations de contraintes de disponibilité des ressources est très important. Les procédures d'amélioration de faisabilité que nous avons utilisées ne sont pas assez efficaces pour échapper à ce problème.

3.5 Approche réactive

Nous avons étudié le cas réactif du problème du *PMSPSP*. De point de vue pratique, ce cas est particulièrement intéressant car les données disponibles lors de la réalisation du premier planning sont amenées à changer. Plusieurs sources de perturbations existent, en l'occurrence la sous estimation des charges par le chef du projet conduit souvent à une réévaluation de la durée qu'une ressource passera à réaliser une tâche. D'autres tâches peuvent aussi être introduites dans un projet en cours d'exécution. L'absence des personnes peut aussi arriver d'une manière imprévisible, etc. A l'inverse, des aléas comme la sur-estimation de la charge, la suppression d'une tâche, la présence d'une ressource qui n'était pas prévue lors du démarrage du projet, etc., peuvent aussi survenir. Nous renvoyons à [Billaut *et al.*, 2010] pour une description complète des aléas possibles. Face à cette multiplicité de sources d'incertitude et en l'absence des données statistiques sur l'arrivée de chaque type d'incertitude, il ne sera pas possible de concevoir des méthodes robustes capable d'absorber ces aléas.

Nous avons donc décidé de développer une approche purement réactive sans se baser sur des solutions de base pro-actives [Billaut *et al.*, 2010].

Le modèle que nous proposons ici a l'avantage d'être générique dans la mesure où il est adapté à plusieurs type d'aléas. L'idée principale est d'ajouter un deuxième critère permettant de minimiser les perturbations des plannings du projet. Dans la partie 3.5.1 nous définissons formellement ce modèle, puis nous décrivons dans la section 3.5.2 le fonctionnement d'un générateur d'instances que nous avons utilisé. Dans la partie 3.5.3 un modèle linéaire est proposé. Ce modèle linéaire est utilisé dans la méthode $\epsilon-$contrainte proposée dans 3.5.4 pour calculer les fronts de Pareto optimaux. Dans la partie 3.5.5.1 nous proposons un algorithme de type NSGA-II pour résoudre ce problème. Des résultats expérimentaux sont présentés dans la partie 3.5.6.

3.5.1 Description du problème

Nous avons ici un projet comme défini dans la partie 3.2 qui est en cours de réalisation. A l'instant τ, nous décidons de le re-planifier suite aux changements survenus sur les données utilisées pour définir le planning précédent. Nous avons donc n tâches à planifier. Parmi ces n tâches, un ensemble est en cours d'exécution. Pour les tâches qui sont en cours d'exécution nous distinguons 2 types : les tâches non préemptives en cours d'exécution notées $\mathcal{IPNP}$ et celles préemptives $\mathcal{IPP}$. Les autres tâches soit étaient planifiées et ne sont pas encore commencées ou bien sont de nouvelles tâches qui viennent d'être identifiées. En plus des contraintes du problème initial où nous cherchions au démarrage du projet à trouver un ordonnancement de C_{max} minimum, nous rajoutons les contraintes suivantes :

- toute tâche non préemptive en cours à l'instant τ, doit continuer son exécution jusqu'à sa fin sans interruption et sans changement d'affectation. Les charges pouvant être modifiées et nous prenons en compte les informations à l'instant τ
- Pour chaque compétence d'une tâche préemptive en cours d'exécution l'affectation est fixée, mais les fenêtres d'exécution peuvent changer.

Nous cherchons donc à trouver une nouvelle solution qui respecte toutes les contraintes (les contraintes du problème initial et les nouvelles contraintes) telle que la nouvelle date de fin de projet soit minimum et tel que le nombre de changements d'affectation maximum soit minimale. Le critère de changement d'affectation est important du point de vue organisationnel, car dans des projets tels que des projets informatiques, une personne affectée à une tâche peut être mise en relation avec un client ou un autre interlocuteur de l'extérieur d'ou l'importance de ne pas changer cette personne trop fréquemment. Un autre argument justifie ce critère : si une personne est affectée à une tâche, il arrive souvent que cette personne se prépare pour être efficace une fois la tâche commencée. Par exemple, si la réalisation d'une compétence d'une tâche donnée demande des idées qui ne sont pas triviales, la personne arrive parfois à développer des idées pendant son temps libre avant de commencer cette tâche.
Nous négligeons ici le changement éventuel du séquencement des tâches par personne. Cela nous paraît moins important, car si deux tâches sont échangeables entre elles, la personne peut le faire elle même sans respecter le planning fourni par l'algorithme.
Le modèle est donc un cas particulier du modèle prédictif. L'ajout du deuxième critère complexifie le problème et par conséquent les méthodes proposées dans la partie 3.4 ne pourront plus être utilisées.

3.5.2 Génération des instances

Des instances du problème réactif sont générées en introduisant des perturbations quant au déroulement du projet prédictif. Les perturbations sont appliquées à partir d'un instant τ qui est un paramètre de notre approche. Nous avons donc développé un générateur d'instances, qui, à partir d'une instance du problème initial, génère un modèle du problème réactif en résolvant l'instance originale avec l'heuristique présentée en 3.4.1.1. Les perturbations que nous avons considérées sont : les changements des durées des compétences d'une tâche, le changement de disponibilité ou d'indisponibilité d'une personne. Les jeux de données que nous avons utilisés sont les instances de $10, 16$ et 20 tâches.
La procédure suivante décrit le processus de la génération d'une instance réactive.

- Planning actuel : calculé à partir de l'heuristique avec la règle de priorité EDD, on note la durée du projet C^0_{max}
- Dates de ré-planification : $\tau \in \{0, \lfloor \frac{C^0_{max}}{5} \rfloor, 2 \times \lfloor \frac{C^0_{max}}{5} \rfloor, \cdots, C^0_{max} - 1\}$
- Évènements considérés : augmentation de la durée d'une tâche/compétence, l'absence ou la présence d'une personne.
- Pour toutes les tâches, qui finissent entre $\tau + 1$ et $\tau + \lfloor \frac{C^0_{max}}{5} \rfloor$, il y a une probabilité de 20% d'avoir une augmentation de charge $1 \leq \delta \leq \max(1, 0.3 \times p_{i,k})$
- Pour toute personne m et période $t \in [\tau, \tau + \lfloor \frac{C^0_{max}}{5} \rfloor - 1]$, si la personne était disponible, elle a une probabilité de 10% pour devenir indisponible. Sinon, elle a une probabilité de 5% pour devenir disponible.
- Une vérification de faisabilité de la solution, pour les tâches non préemptives en cours d'exécution dans l'intervalle $[\tau, \tau + 1]$ est appliquée à la fin de la procédure de génération d'instances. Elle peut remettre en cause des aléas générés pendant les deux étapes précédentes.

3.5.3 Modèle linéaire

Dans cette partie, nous proposons un modèle linéaire à variables mixtes pour résoudre ce problème de ré-ordonnancement à partir de l'instant τ. Le modèle linéaire proposé est basé sur le *PLNE* du mode prédictif (cf paragraphe 3.2). En raison du nombre très important de variables générées par ce modèle, il est impossible de résoudre des instances de grande taille en temps raisonnable. Cependant, nous l'avons utilisé dans la méthode $\epsilon-$contrainte décrite dans la section 3.5.4 afin de calculer le front de Pareto pour des instances de petite taille.

Données supplémentaires : en plus des données des instances initiales, nous définissons ici les données d'une instance réactive.

- $oy_{i,k,m} = 1$ si la personne P_m était affectée à la compétence S_k de la tâche A_i dans le planning initial ; 0 sinon
- $ip_{i,k} = 1$: si la compétence S_k de la tâche A_i est en cours d'exécution, 0 sinon
- os_i : la date début des tâche A_i dans le planning actuel
- τ : date début du ré-ordonnancement
- $\overline{x}_{i,m,k,t} = 1$ si la compétence S_k de la tâche A_i s'exerçait par la personne P_m pendant $[t, t+1]$; 0 sinon

Variables supplémentaires

- $ny_{i,k,m} = 1$ si la personne P_m est affectée à la compétence S_k de la tâche A_i dans le nouveau planning ; 0 sinon
- $chg_{i,k,m} = 1$ si $oy_{i,k,m} \neq ny_{i,k,m}$; 0 sinon. Ce booléen indique si l'affectation de la compétence S_k de la tâche A_i à l a personne P_m dans le nouveau planning est différente de celle dans le planning de base.
- $chg_m = \sum_i \sum_k chg_{i,k,m}$. Cette variable compte le nombre d'affectations des compétences des tâches à la personne P_m, qui n'apparaissaient pas dans le planning de référence (planning initial).

Contraintes

1. fixer le planning avant τ :

$$x_{i,m,k,t} = \overline{x}_{i,m,k,t} \quad \forall i,k,m, \quad 0 \leq t \leq \tau - 1 \tag{3.24}$$

2. ne pas interrompre une tâche non préemptive en cours d'exécution pendant $[\tau, \tau+1]$

$$x_{i,m,k,t} = oy_{i,k,m} \quad \forall \quad (i,k) \in Ip \qquad \forall i \in \mathcal{A}_{\overline{p}} \quad \forall m \quad os_i \leq t \leq os_i + p_{i,k} \tag{3.25}$$

$$ny_{i,k,m} = oy_{i,k,m} \quad \forall \quad i,k,m \quad os_i < \tau \tag{3.26}$$

3. $ny_{i,k,m} = 1$ si la personne P_m exerce la compétence S_k de la tâche A_i

$$ny_{i,k,m} = \frac{\sum_t x_{i,m,k,t}}{p_{i,k}} \; \forall i,k,m \; (p_{i,k} > 0) \tag{3.27}$$

4. définition des contraintes de changement d'affectation par rapport au planning prédictif

$$chg_{i,k,m} \geq ny_{i,k,m} - oy_{i,k,m} \tag{3.28}$$
$$chg_{i,k,m} \geq oy_{i,k,m} - ny_{i,k,m} \tag{3.29}$$
$$chg_{i,k,m} \leq 2 - (oy_{i,k,m} + ny_{i,k,m}) \tag{3.30}$$
$$chg_{i,k,m} \leq oy_{i,k,m} + ny_{i,k,m} \tag{3.31}$$

5. contrainte sur le changement total d'affectation d'une personne aux tâches

$$chg_m = \sum_i \sum_k chg_{i,k,m} \tag{3.32}$$

Objectifs : nous minimisons à la fois la date d'achèvement du projet C_{max} (Equation 3.33) et le nombre maximum de changements d'affectation (Equation 3.34).

$$\min C_{max} \tag{3.33}$$

$$\min \max_m chg_m \tag{3.34}$$

3.5.4 Génération du front de Pareto optimal

A partir du modèle linéaire, la génération du front de Pareto est obtenue selon l'approche $\epsilon-$contrainte décrite par l'Algorithme 5. Nous résolvons d'abord le modèle linéaire en optimisant seulement le deuxième critère ($\max chg_m$). Si une solution faisable est trouvée avec une date de fin de projet C^0_{max}, nous résolvons de nouveau le modèle avec une contrainte additionnelle $C_{max} \leq C^0_{max} - 1$. Cette procédure est répétée jusqu'à ce que le problème devienne non réalisable. Cette approche a permis d'énumérer les fronts de Pareto pour les instances générées de petite taille. Nous nous en servirons, notamment pour la comparaison avec les résultats de l'algorithme $NSGA-II$ proposé dans la partie 3.5.5.1. Les résultats de cette méthode sont présentés dans la partie 3.5.6.

Algorithme 5: Epsilon contrainte

$FP \leftarrow \phi$
$S \leftarrow$ Opt(maxChang)
$feasible \leftarrow S.feasible$
tantque $feasible$ **faire**
  $NC_{max} \leftarrow S.c_{max}$
  $NMaxChang \leftarrow S.maxChang$
  $FP \leftarrow FrontPareto(FP, (NC_{max}, NMaxChang))$
  $S \leftarrow Opt(maxChang, c_{max} < NC_{max})$
  $feasible \leftarrow S.feasible$
fin tantque

3.5.5 Méthodes approchées

La méthode exacte décrite ci-dessus ne permet pas de résoudre des instances de taille réelle. Par conséquent, nous proposons un algorithme génétique de type NSGA-II afin de calculer une approximation de front de Pareto.

3.5.5.1 Algorithme NSGA-II

Une implémentation de l'algorithme NSGA-II proposée dans [Deb *et al.*, 2000] est proposée afin de trouver un front Pareto approché. L'algorithme NSGA-II a largement été utilisé pour résoudre des problèmes d'optimisation multi-critères. Récemment, [Ballestín et Blanco, 2011] ont mené une étude sur le RCPSP multi-objectif dans laquelle ils ont comparé l'efficacité du NSGA-II par rapport à d'autres métaheuristiques multi-objectifs (*SPEA2* et *PSA*). Leur étude donne un large avantage à NSGA-II par rapport à l'algorithme *PSA* et un légère avantage par rapport à la méthode *SPEA2*. Étant une méthode évolutionnaire à base de population, l'algorithme NSGA-II part d'un ensemble de solutions (population initiale) et les améliore de façon itérative jusqu'à ce qu'une condition d'arrêt soit satisfaite (nombre d'itérations ou limitation du temps de calcul, par exemples). Le principe de l'algorithme NSGA-II peut être défini comme suit :

1. générer la population initiale Q_0 avec N individus,
2. reproduire une population Q_1 de taille N à partir de Q_0 en utilisant les opérateurs de reproduction (croisement et mutation),
3. fusionner les deux populations en une seule $Q' = Q_0 \cup Q_1$,
4. sélectionner N individus de Q' en utilisant une stratégie de sélection et les mettre en Q_0,
5. répéter les étapes $(2-4)$ jusqu'à la satisfaction du critère d'arrêt.

Afin de construire une nouvelle génération, la méthode NSGA-II utilise une stratégie élitiste. Elle regroupe les $2 * N$ individus résultants de la reproduction dans des fronts $F_0, \ldots, F_n$ tel qu'aucune solution dans un front F_i ne domine l'autre et les solutions du front F_{i+1} sont dominées par celles du front F_i. Les individus à sélectionner sont pris des fronts $F_0, \ldots F_k$. L'indice k correspond au $k$ieme front tel que le nombre total des individus des fronts $F_0, \ldots F_{k-1}$ est inférieur à N et le nombre total des individus des fronts $F_0, \ldots, F_k$ est supérieur ou égal à N. Ainsi tous les individus appartenant aux $F_0, \ldots, F_{k-1}$ sont donc pris. Pour compléter la population à N individus, des solutions sont à les prendre du front F_k. La méthode NSGA-II utilise naturellement un mécanisme de dispersion. Ce mécanisme intervient lors de la sélection des individus du dernier front F_k. La sélection se fait dans l'ordre décroissant selon la distance de *Crowding* (une autre distance peut être utilisée). Cette distance est grande pour une solution si la densité des solutions qui l'entourent est faible et petite si la densité est forte.

3.5.5.2 Algorithmes générales pour la détermination de fronts de Pareto

Algorithme 6: Construction des fronts

pour tout $p \in P$ **faire**
  pour tout $q \in P$ **faire**
    si p *domine* q **alors**
      $S_p \leftarrow S_p \cup \{q\}$
    sinon
      si q *domine* p **alors**
        $n_p = n_p + 1$
      finsi
    finsi
  fin pour
  si $n_p = 0$ **alors**
    $\mathcal{F}_1 = \mathcal{F}_1 \cup \{p\}$
  finsi
fin pour
$i \leftarrow i + 1$
tantque $\mathcal{F}_i \neq \emptyset$ **faire**
  $\mathcal{H} = \emptyset$
  pour tout $p \in \mathcal{F}_i$ **faire**
    pour tout $q \in S_p$ **faire**
      $n_q = n_q - 1$
      si $n_q = 0$ **alors**
        $\mathcal{H} = \mathcal{H} \cup \{q\}$
      finsi
    fin pour
  fin pour
  $i = i + 1$
  $\mathcal{F}_i = \mathcal{H}$
fin tantque

Algorithme 7: Boucle principale du $NSGA - II$ [Deb *et al.*, 2000]

$\mathcal{R}_t \leftarrow \mathcal{P}_t \cup \{\mathcal{Q}_t\}$ // fusionner les deux populations (parente et ses enfants)
$Fast - Nondominated - Sort(\mathcal{R}_t)$ // trier $\mathcal{F} = \{\mathcal{F}_1, \mathcal{F}_2, \ldots\}$: tous les fronts non dominés de $\mathcal{R}_t$
tantque $\mathcal{P}_{t+1} < N$ **faire**
  crowding-distance-assignment($\mathcal{F}_i$) // calculer les distances de *crowding* de $\mathcal{F}_i$
  $\mathcal{P}_{t+1} \leftarrow \mathcal{P}_{t+1} + \mathcal{F}_i$
fin tantque
Trier $\mathcal{P}_{t+1}$ selon la distance de *crowding*
$\mathcal{P}_{t+1} \leftarrow$ *les* N premier individus de $\mathcal{P}_{t+1}$
$\mathcal{Q}_{t+1} \leftarrow$ créer une nouvelle population à partir du $\mathcal{P}_{t+1}$
$t \leftarrow t + 1$

3.5.5.3 Population initiale

L'initialisation d'un algorithme génétique est une phase importante et peut avoir un impact sur la qualité des solutions calculées au cours des autres phases. Si, par exemple, les solutions initiales ne sont pas assez diverses, la méthode peut converger rapidement et de façon prématurée. Pour avoir une population de départ assez diverse, nous exécutons l'algorithme glouton $2 * N$ fois avec les 10 règles de priorité présentées en section 3.4.1.1 et avec la règle RND $(2 * N - 10)$ fois. Puis on utilise la même sélection qui est employée dans la méthode $NSGA - II$, c'est-à-dire nous utilisons la sélection par rang puis selon la distance de *crowding* appliquée sur le k ième front, nous complétons la population à N individus.

3.5.5.4 Codage et opérateurs génétiques

Le codage utilisé ici est le même que celui présenté dans la section 3.4, qui correspond au codage avec affectation intégrée. Nous représentons donc une solution par une liste de priorité qui respecte les relations de précédence avec une liste de personnes associées à chaque tâche, dédiées donc à sa réalisation. La liste de personnes est triée par ordre de compétence pour laquelle la personne est utilisée. Nous utilisons les mêmes opérateurs de croisements et mutation que ceux décrits dans la partie 3.4.3.1. Les tâches préemptives déjà commencées ne sont pas concernées par le changement d'affectation. Le décodage de la solution est obtenu par la méthode *PAPSS*, sans calcul d'affectation.

3.5.5.5 Sélection des parents pour la reproduction

Dans les algorithmes génétiques, deux méthodes de sélection sont nécessaires (éventuellement identiques), la sélection d'une nouvelle génération pour remplacer celle en cours et la sélection des parents à reproduire. Dans notre cas, pour la sélection de la nouvelle population, nous utilisons la méthode native du NSGA-II comme décrite ci-dessus. Pour le choix des individus à utiliser dans la reproduction, nous avons utilisé la sélection par tournoi à 2 comme dans les algorithmes génétiques mono-critères présentées en section 3.4.3. La fonction de fitness d'une solution utilisée ici n'est autre que son rang correspondant au front de Pareto auquel elle appartient. Les solutions des premiers rangs ont donc plus de chance d'être sélectionnées. En revanche, deux solutions du même rang ont la même chance de sélection (on n'utilise pas le critère de dispersion pour distinguer entre deux solutions du même rang).

3.5.6 Résultats expérimentaux

Dans ce qui suit, nous présentons les résultats de la méthode NSGA-II. Nous comparons d'abord ses résultats avec les fronts optimaux calculés par l'approche exacte, à l'aide de la métrique *distance générationnelle* (cf paragraphe 3.5.6.1). Pour les instances ou nous ne disposons pas de fronts optimaux, nous étudions la performance de la méthode NSGA-II à l'aide de la métrique *hypervolume* (cf paragraphe 3.5.6.1)

3.5.6.1 Métriques d'évaluation utilisées

Contrairement aux problème mono-critères, l'évaluation de la qualité d'un ensemble de solutions potentiellement Pareto optimal n'est pas triviale. Il existe plusieurs mesures de performance des méthodes multi-objectifs [Zitzler *et al.*, 2003]. Dans [Veldhuizen et Lamont, 2000], les auteurs ont classifié ces mesures en trois groupes :

1. selon la proximité de Pareto optimal ;
2. selon la diversité des solutions ;
3. selon les deux critères en même temps.

Dans notre cas, nous utilisons la métrique **Distance Générationnelle**, qui est une métrique de la première catégorie, pour comparer les résultats du NSGA-II avec les optima de Pareto obtenus par l'approche $\epsilon-$contrainte. Afin de mesurer la performance de la méthode NSGA-II sur d'autres instances où nous ne connaissons pas le front optimal, nous utilisons la métrique **hypervolume**, qui mesure à la fois la diversité et la distance de l'optimal.

Distance Générationnelle : cette métrique calcule la distance entre le front Pareto approché OP^* et le front de référence OP selon la formule suivante [Veldhuizen et Lamont, 2000] :

$$DG = \frac{(\sum_1^{|OP^*|} d_i^p)^{\frac{1}{p}}}{|PO^*|})$$

où p est le nombre de critères à optimiser. Dans le cas de deux critères :

$$d_i = \min_{k=1}^{k=|PO|} \sqrt{\sum_1^p (f_j^i - f_j^k)^2}$$

.

Cette distance est calculée en utilisant des cordonnées normalisées pour les deux critères. Pour chaque instance, la valeur maximum de chaque critère pris des deux fronts approché et optimal est utilisée.

Hypervolume : cet indicateur mesure le volume engendré par les points du front à évaluer par rapport à un point de référence, qui est dominé par tous les points du front (cf Figure 3.9). Puisque nous minimisons les objectifs, la qualité des fronts approchés est bonne lorsque cet indicateur est grand. En effet, dans ce cas, les solutions sont loin du point de référence. Il est évident que le choix du point de référence est important. Pour obtenir ce point de référence, nous devons calculer une borne supérieure pour chaque critère. Concernant la date de fin de projet C_{max}, nous considérons l'horizon maximum de planification comme borne supérieure . Pour le deuxième critère, le maximum de changement d'affectation pour une personne est atteinte lorsque celle-ci est affectée à toutes les tâches pour lesquelles elle n'était pas impliquée initialement, et pour une compétence différente de celle qu'elle faisait avant pour cette même tâche. La borne supérieure pour le deuxième critère est donc le maximum de ces valeurs. A noter aussi que les deux critères ne sont pas toujours

du même ordre de grandeur. Donc, on utilise toujours les valeurs normalisées en divisant sur les valeurs des bornes supérieures. Le point de référence a donc les cordonnées $(1, 1)$.

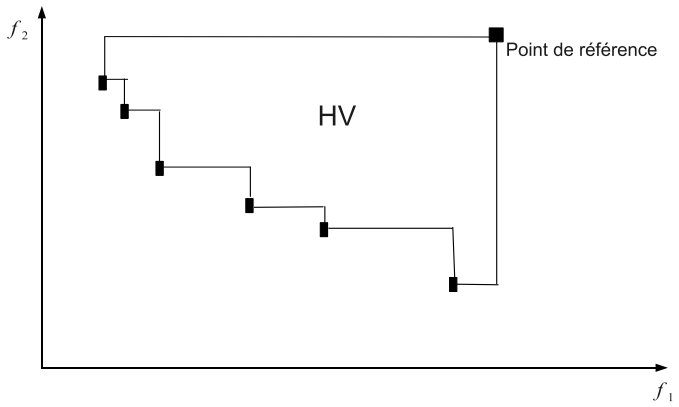


FIGURE 3.9 – L'hypervolume engendré par un ensemble de points non dominés (cas biobjectif)

3.5.6.2 Algorithme NSGAII et $\epsilon - contrainte$

Dans cette partie, nous mesurons la qualité des solutions calculées par l'algorithme génétique en comparant les fronts de Pareto de cette méthode avec les optima de Pareto calculés à l'aide de l'approche $\epsilon - contrainte$. A partir des instances initiales pour le cas prédictif, les expérimentations ont été menées en utilisant 10 instances de 10 tâches. Pour chaque instance, 3 instances réactives ont été générées aux périodes $\{0, 3 \times \lfloor \frac{C^0_{max}}{5} \rfloor, 5 \times \lfloor \frac{C^0_{max}}{5} \rfloor\}$. Les paramètres de la méthode $NSGA - II$ utilisés sont :

- population initiale = 20 individus,
- nombre d'itérations : 250,
- probabilité de croisement : $0,8$,
- probabilité de mutation : $0,05$.

évènement	distance	Temps NSGA-II	Temps EC
0	0,66	1,02	429,67
3	0,39	1,02	19,9
5	0,31	0,89	5,03

Tableau 3.16 – NSGAII vs $\epsilon - contrainte$

3.5.6.3 Performance du NSGA-II selon l'indicateur Hypervolume

Dans le tableau 3.17, nous présentons les résultats obtenus par le $NSGA - II$ sur les instances de 16 tâches en utilisant des perturbations générées aux périodes $\{0, \times \lfloor \frac{C^0_{max}}{5} \rfloor, 2 \times$

$\lfloor \frac{C^0_{max}}{5} \rfloor\}$. Cela fait 120 instances (3×40). Les paramètres du $NSGA-II$ sont :
- population initiale = 32 individus
- nombre d'itérations : 250
- probabilité de croisement : $0,8$
- probabilité de mutation : $0,05$

évènement	HV	Temps NSGA-II
0	0,59	1,73
1	0,54	1,69
2	0,54	1,40

Tableau 3.17 – NSGAII en utilisant les instances de 16 tâches

Les résultats présentés dans le Tableau 3.16, montrent une diminution de la distance du front approché par rapport à l'optimum lorsque les perturbations arrivent plus tard. Par exemple, la distance était à $0,6619$ à l'instant $\tau = 0$, alors que cette distance était seulement à $0,3136$ pour les instances correspondant aux perturbations à $\tau = 5 \times \lfloor \frac{C^0_{max}}{5} \rfloor$. Nous notons que l'approche $\epsilon-$contrainte lors de la résolution des instances perturbées au démarrage du projet ou à un instant t proche du 0, a trouvé des solutions meilleures que la solution initiale en terme de C_{max}, mais en produisant des changements sur les affectations initiales.
Une autre remarque importante, qui peut mettre en cause l'utilisation de la distance générationnelle comme indicateur de performance, est le nombre de solutions. Par exemple, dans certains cas l'algorithme NSGA-II trouve un front de Pareto d'un seul point appartenant au front optimal, ce qui donne par la suite, une distance de 0, alors que le front optimal contient plus d'un point.

Dans le Tableau 3.16, la deuxième colonne présente les valeurs de l'hypervolume (HV) moyen engendré par les points non-dominés calculés par l'algorithme NSGA-II. Nous pouvons constater que cette valeur est plus grande pour les instances correspondant à $\tau = 0$ ce qui n'est pas attendu si l'on se réfère aux résultats présentés dans le Tableau 3.16. Cela est dû en partie au fait que la borne supérieure du $MaxChg$ est souvent plus élevée au début du projet, car elle dépend du nombre de tâches non terminées.

3.6 Conclusion

Un problème de gestion de projets à contraintes de personnel multi-compétences a été étudié. Nous avons proposé un modèle mathématique à variables mixtes. Ce modèle a été testé en utilisant *CPLEX*, il peut résoudre des instances de petite taille allant jusqu'à 30 tâches. Dans une autre phase de cette étude, nous avons proposé des bornes inférieures destructives adaptées à ce modèle. L'analyse des résultats expérimentaux de ces bornes inférieures montrent qu'elles donnent de bons résultats lorsque le nombre de personnes nécessaires par tâche n'est pas très élevé par rapport au nombre total de personnes ou lorsque le taux de disponibilité des personnes est élevé.

Un algorithme d'ordonnancement parallèle préemptif basé sur des règles de priorité est proposé pour résoudre ce problème. L'analyse des résultats de cette méthode a permis de classifier ces règles de priorité selon la qualité des solutions trouvées.

Nous avons également proposé des métaheuristiques basées sur un codage indirect pour la résolution de ce problème. Des métaheuristiques de type Recherche tabou et algorithmes génétiques sont considérées. Selon les résultats expérimentaux obtenus, nous pouvons affirmer l'efficacité des méthodes tabou par rapport aux algorithmes génétiques développés. Une autre méthode tabou à codage indirect a été étudiée. Les résultats obtenus ne sont pas motivants par rapport aux approches à codage indirect.

Dans la dernière partie de ce chapitre, nous avons étudié la version réactive du problème *PMSPSP*. Nous proposons une approche globale, qui ne dépend pas de la nature des aléas survenus au planning en cours. Nous proposons de résoudre ce problème en considérant deux critères : le critère initial minimisation de date de fin de projet et un deuxième critère minimisant le nombre maximum de changements d'affectation par rapport à la solution en cours.
Pour résoudre le cas réactif du problème considéré, nous proposons une approche exacte basée sur un modèle linéaire et une méthode approchée de type *NSGA-II*.
Les premiers résultats obtenus sont assez satisfaisants. Il serait donc intéressant d'effectuer d'autres expérimentations pour analyser plus finement les performances des méthodes de résolution proposées dans le cas réactif.

Dans ce modèle, nous avons considéré seulement la minimisation de la durée du projet, comme objectif, alors que dans certains cas, le directeur de projet peut s'intéresser à la fois à un critère temporel et d'autres critères, comme les critères optimisant le coût du projet. Nous nous sommes donc intéressés à étudier un deuxième modèle proche de celui étudié dans ce chapitre, et qui prend en considération deux critères à optimiser, dans sa version prédictive. L'étude de ce modèle sera présentée dans le chapitre suivant.

Chapitre 4

Problème d'ordonnancement de projet avec ressources multi-compétences et tâches mono-compétence

Dans ce chapitre, nous étudions le problème *MSRMST* dont la définition complète a été exposée dans le chapitre 1.3.2. Nous allons tout d'abord modéliser ce problème sous forme d'un programme mathématique linéaire à variables mixtes. Bien que ce modèle mathématique n'ait pas été testé expérimentalement, il reste utile pour la compréhension des diverses contraintes du problème *MSRMST*.
Dans la partie 4.4 nous exposons des méthodes de résolution proposées pour le résoudre. Dans la Section 4.3.1, nous décrivons les jeux de données utilisés et les divers paramètres appliqués pour la génération des instances. Les résultats expérimentaux de ces méthodes sont présentés dans 4.3.2.

4.1 Modèle mathématique pour le MSRMST

Le problème étudié ici a été présenté en détail dans le Chapitre II. Il s'agit d'un autre problème d'ordonnancement de projet diffèrent du *PMSPSP* étudié dans le Chapitre III où les besoins des tâches ne sont pas multiples. Une tâche nécessite une seule compétence et par conséquent une seule personne. Contrairement au premier modèle *PMSPSP* où la préemption était définie par tâche (certaines peuvent être préemptées et d'autres non), pour le *MSRMST*, une tâche ne peut être interrompue que lorsque la personne affectée devient indisponible (congés, absence, etc.). Dans ce cas la tâche est interrompue et reprise à la première prochaine période de disponibilité de la personne. Nous avons aussi introduit des dates de disponibilité (r_i) et des dates de fin souhaitée (d_i) pour les tâches. Un autre aspect que nous traitons dans ce modèle est le coût des prestations des personnes. Ainsi, on associe à chaque personne un coût unitaire par compétence. Étant donné qu'une tâche nécessite seulement une seule compétence, ce coût peut être considéré par personne/tâche.

On note par la suite $w_{i,m}$, le coût de réalisation d'une unité de temps de la tâche A_i par la personne P_m.

L'objectif est de proposer au décideur un ensemble de solutions de compromis, qui minimisent à la fois le retard maximum T_{max} et le coût global du projet. Cet ensemble doit évidement respecter toutes les contraintes du problème, qui sont : besoin des tâches, relations de précédence entre les tâches, date de disponibilité de chaque tâche, disponibilité et compétences de chaque personne.
Le problème *MSRMST*, peut être formulé par le programme linéaire ci-dessous.

Variables :

–

$$\forall i \in \{0,\dots,n+1\}, \forall m \in \{1,\dots,M\}, \forall t \in \{1,\dots,T\}$$
$$x_{i,m,t} = \begin{cases} 1 & \text{si } P_m \text{ fait } A_i \text{ durant } [t, t+1[\\ 0 & \text{sinon} \end{cases}$$

– T_{max} : retard maximum.

Contraintes :

$$s.t \quad \sum_{i \in \mathcal{A}} x_{imt} \leq disp(m,t) \quad \forall\, m, \forall\, t \tag{4.1}$$

$$s_i \geq t - T \times \sum_{q=0}^{max(0,t-1)} x_{imt} \quad \forall\, i,m,t \tag{4.2}$$

$$s_i \leq t \times x_{imt} + (1 - x_{imt}) \times T \quad \forall\, i,m,t \tag{4.3}$$

$$c_i \geq x_{imt} \times (t+1) \quad \forall\, i,m,t \tag{4.4}$$

$$x_{imt} + x_{im't'} \leq 1 \quad \forall\, i, (m \leq m'), (t \leq t') \tag{4.5}$$

$$c_i \leq s_j \quad \forall\, (i,j) \in E \tag{4.6}$$

$$disp(m,t_2) \times x_{imt_1} + disp(m,t_2) \times x_{imt_3} \leq disp(m,t_2) \times x_{imt_2} + 1 \quad \forall m,i,(t_1 < t_2 < t_3) \tag{4.7}$$

$$\sum_m \sum_t t_{imt} = p_i \quad \forall\, i \tag{4.8}$$

$$s_i \geq r_i \quad \forall\, i \tag{4.9}$$

$$T_{max} \geq c_i - d_i \quad \forall\, i \tag{4.10}$$

$$T_{max} \geq 0 \tag{4.11}$$

Objectifs :

$$Min \sum_m \sum_i \sum_t x_{imt} \times w_{im} \tag{4.12}$$

$$Min\ T_{max} \tag{4.13}$$

La contrainte 4.1 vérifie la cohérence entre la charge affectée à une personne et sa disponibilité. Les contraintes 4.2, 4.3 et 4.4 lient la date de début s_i et la date de fin c_i aux variables x_{imt}. La contrainte 4.5 force une tâche à être affectée à une et seulement une personne. La contrainte 4.7 a pour but d'interdire la préemption d'une tâche par une autre tâche (une personne ne peut interrompre une tâche sauf pendant ses périodes d'indisponibilité). Les contraintes de précédence sont formulées dans la contrainte 4.6. La contrainte 4.8 force une tâche à s'exécuter entièrement. La contrainte 4.9 à pour objectif de modéliser le respect des dates de disponibilité, et donc exiger que les dates de début des tâches doivent être ultérieures ou égales à leurs dates de disponibilité. Les contraintes 4.10 et 4.11 définissent le T_{max}.

4.2 Méthodes de résolution approchées pour le *MSRMST*

Dans cette section, nous proposons des méthodes approchées de type algorithmes de liste et métaheuristiques (Recherche tabou et NSGA-II) pour la résolution du problème bicritère. Dans la sous-section 4.2.1, nous proposons des heuristiques basées sur des règles de priorité. Deux schémas sont proposés pour résoudre le problème : séquentiel et global. Pour le schéma séquentiel, nous déterminons d'abord l'affectation puis nous résolvons le problème de séquencement. Quant au schéma global, l'affectation et le séquencement des tâches se font en parallèle. Par la suite, nous proposons de résoudre le problème bicritère par des métaheuristiques (cf sous-section 4.2.2). Une Recherche tabou énumérant le front de Pareto en considérant itérativement l'approche $\epsilon-$contrainte est proposée. En outre, un algorithme NSGA-II où les deux critères sont à minimiser en même temps est étudié.

4.2.1 Algorithmes de liste

Dans cette partie, nous proposons de résoudre le problème en minimisant le coût d'affectation global sous contrainte que le retard maximum est inférieur à une valeur Q. Cela revient à définir pour chaque tâche une date de fin impérative $\tilde{d}_i = d_i + Q$. Deux stratégies sont utilisées pour affecter les ressources aux tâches :

1. l'affectation est faite indépendamment au séquencement (cf. 4.2.1.1) ;
2. l'affectation est calculée pendant le séquencement des tâches (cf. 4.2.1.2).

Ainsi, dans le premier cas, il s'agit d'une approche de résolution séquentielle (résoudre le problème d'affectation puis l'ordonnancement) et dans le deuxième, d'une approche globale où l'affectation et l'ordonnancement sont considérés en même temps.

4.2.1.1 Approche séquentielle

Dans cette approche, nous isolons le problème d'affectation du problème de séquencement des tâches. On affecte d'abord les personnes aux tâches en respectant les contraintes de compétences, puis on résout le problème d'ordonnancement en utilisant le schéma d'ordonnancement parallèle donné par l'Algorithme 8 où une règle de priorité est fixée. Trois méthodes d'affectation des ressources aux tâches ont été testées :

1. affectation aléatoire ($H1$) : pour chaque tâche, on choisit aléatoirement une personne parmi celles, qui sont capables à la réaliser
2. affectation avec prise en compte de coût ($H2$) : pour chaque tâche, on lui affecte la personne dont le coût est le moins élevé
3. affectation avec prise en compte de coût total et équilibrage de charge ($H3$) : dans cette heuristique d'affectation, le choix des ressources se fait en prenant en compte le coût tout en favorisant l'équilibrage de charge entre les personnes.

Notons que la liste de personnes capables de réaliser une tâche à une date t prend en considération la disponibilité des personnes à cette date. c'est- à-dire une personne, qui n'a pas la disponibilité suffisante pour finir la tâche avant sa date de fin imposée n'appartient pas à la liste des personnes susceptibles de réaliser la tâche.

Algorithme 8: Ordonnancement des tâches avec une affectation connue

$\mathcal{A}$: Liste des tâches
$t \leftarrow 0$
tantque $\mathcal{A} \neq \phi$ **faire**
  $\mathcal{E}L(t)$: Tâches éligibles à t
  pour tout ressource R avec des tâches $\in \mathcal{E}L(t)$ **faire**
    $RT \leftarrow$ tâches éligibles affectées à R
    trier RT selon une règle de priorité
    calculer la date de début et fin pour chaque tâche de RT selon la disponibilité de R
    mettre à jour les disponibilités de la personne R
    $\mathcal{A} \leftarrow \mathcal{A} \setminus RT$
  fin pour
  $t \leftarrow$nextEvent(t)
fin tantque

4.2.1.2 Approche globale

Dans cette approche, l'affectation est calculée au cours du séquencement des tâches. L'idée est d'ordonnancer à chaque instant t le nombre maximum de tâches éligibles en tenant compte des priorités. Le schéma général de cette méthode est décrit en Algorithme 9. La procédure *MaxBestAssign(A(t))* appelée à chaque instant t, se charge de résoudre le problème d'affectation des tâches éligibles à la date t. Ce problème est modélisé sous forme d'un flot maximum à coût minimum (cf Figure 4.1) et résolu en utilisant l'algorithme de *Busacker & Gowen* [Busacker et Gowen, 1961] adapté à la structure du graphe considéré. La prise en compte de la règle de priorité est considérée en utilisant des coûts sur les arcs allant de la source du graphe vers les nœuds tâches. Ce coût est calculé en fonction des coûts des personnes/tâches. On utilise une valeur H suffisamment grande par rapport aux coûts associés aux personnes/tâches. Nous prenons $H > \max_{(i,m)} w_{i,m} \times p_i$. Ainsi, avec l'introduction de H, on privilégie de minimiser le coût d'affectation plutôt que d'ordonnancer une tâche plus prioritaire, si le choix se présente et même si cette affectation concerne des tâches moins prioritaires. Pour chaque tâche, on associe un coût défini par $(rang \times H)$ où

rang est le rang de la tâche dans la liste de priorité. L'objectif est d'ordonnancer le maximum des tâches possibles avec un coût minimum respectant au mieux possible la priorité des tâches.

Algorithme 9: Procédure globale

$\mathcal{A}$: Liste des tâches
$t \leftarrow 0$
tantque $\mathcal{A} \neq \phi$ **faire**
  $\mathcal{A}(t)$: tâches éligibles à t
  $\mathcal{B} = MaxBestAssign(A(t))$
  pour tout $B_i \in \mathcal{B}$ **faire**
    calculer la date de début et fin pour B_i
    mettre à jour les disponibilités de la ressource
    $\mathcal{A} \leftarrow \mathcal{A} \setminus \{B_i\}$
  fin pour
  $t \leftarrow$ nextEvent(t)
fin tantque

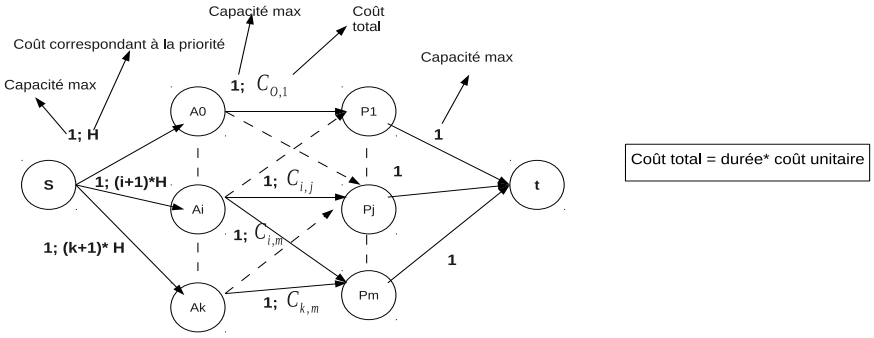


FIGURE 4.1 – Affectation des ressources aux tâches (MaxBestAssign(A(t))

4.2.2 Métaheuristiques

Dans cette partie, nous proposons une méthode tabou (cf 4.2.2.1) pour optimiser le coût total du projet tout en respectant la contrainte sur la valeur Q du retard maximum T_{max} tolérée ($T_{max} \leq Q$). Puis cette méthode utilise une approche $\epsilon-$contrainte (cf Algorithme 10) de manière itérative pour trouver une approximation du front de Pareto pour le problème bicritère. Dans la partie 4.2.2.2, nous proposons également un algorithme génétique de type NSGA-II pour la résolution du problème *MSRMST*.

Avant de présenter nos diverses méthodes de résolution, nous allons rappeler avec plus ou moins de détails, quelques études de la littérature ayant proposé des métaheuristiques multi-critères pour des problèmes connexes, dont nous nous sommes inspirés.

Dans [Viana et de Sousa, 2000], les auteurs ont étudié deux métaheuristiques multi-objectifs pour résoudre le problème du *RCPSP* multi-mode. Trois critères ont été considérés : (1) minimisation du C_{max} ; (2) minimisation de la somme du retard maximum(T_{max}) pondéré et (3) minimisation de violation des contraintes de ressources. Les deux métaheuristiques proposées ne sont autres que l'adaptation de l'algorithme *PSA* [Czyzak et Jaszkiewicz, 1998] et de la Recherche tabou multi-critères *MOTS* proposée par [Hansen, 1997].

L'idée de l'algorithme *MOTS* proposé par [Hansen, 1997] est d'avoir un ensemble de solutions courantes au lieu d'une seule solution dans une recherche tabou classique. Ces solutions sont choisies de façon à ce qu'elles soient dispersées dans l'espace de solutions. Cela a pour but de partager l'espace de recherche entre les solutions. En fait, à chaque étape de la méthode, on sélectionne pour chaque solution courante X_i, le voisin non supporté le plus loin de l'ensemble des solutions courantes et ne dominant pas X_i.
Une stratégie de diversification est utilisée. Elle consiste à remplacer une solution courante aléatoirement par une autre solution courante tirée aléatoirement.

Dans [Gürbüz, 2010], l'auteur a étudié le problème du *MSPSP* avec des niveaux hiérarchiques en introduisant un deuxième critère en plus de la minimisation de la date de fin du projet. Le deuxième critère à optimiser concerne la perte des compétences de ressources. On suppose que demander à une personne d'exercer une compétence ne la maîtrisant complétement par rapport à une autre où elle est totalement à l'aise, peut être une source de démotivation ce qui justifie l'intérêt porté de ce critère.
Un algorithme génétique de type NSGA-II est proposé. Les solutions sont codées en utilisant une liste de priorité pour les tâches et une liste de personnes affectées à chaque tâche. Selon les expérimentations réalisées, cette méthode donne de très bons résultats par rapport aux solutions optimales sur des jeux de données de petites taille. Pour des instances de grande taille allant de 17 à 90 tâches, les auteurs montrent que le $NSGA-II$ proposé domine largement une recherche aléatoire.

4.2.2.1 Recherche tabou

Dans cette partie, nous illustrons notre démarche qui nous a conduit à calculer le front de Pareto par la recherche tabou. Comme il s'agit d'un problème bicritère, le calcul du front de Pareto est effectué selon l'approche $\epsilon-$contrainte. Pour une valeur Q donnée du retard maximum, nous cherchons une solution de meilleur coût global d'affectation. On définit des dates de fin impérative comme suit : $\tilde{d}_i = d_i + Q$. Le problème revient à chercher une solution non-dominée avec $T_{max} = 0$ où seules les dates de fin impérative nouvellement calculées sont considérées. Lors de cette seconde phase, nous faisons ainsi appel à l'algorithme de liste présenté dans la section 4.2.1.2.

Codage et décodage des solutions : le codage de solution proposé est basé sur une règle de priorité avec affectation intégrée. Un individu est représenté par : une règle de priorité, une liste de personnes et pour chaque personne la liste des tâches qu'elle doit

réaliser. Dans la figure 4.2, nous représentons un individu de 4 personnes et 8 tâches. Dans

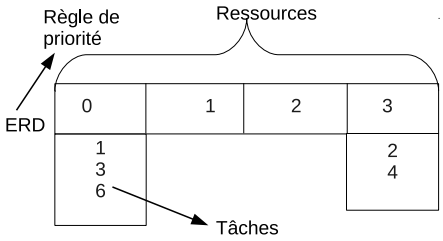


FIGURE 4.2 – Exemple de représentation d'une solution

cet exemple, la personne P_0 doit réaliser les tâches (A_1, A_3, A_6), les tâches A_2 et A_4 sont affectées à la personne P_3. A chaque phase de l'algorithme d'ordonnancement parallèle, les tâches sont triées selon la règle ERD (le numéro de la tâche est utilisé en cas d'égalité des valeurs selon la la règle ERD).

Une solution telle quelle est représentée dans la figure 4.2 est décodée à l'aide de l'algorithme 8 présentée dans la section 4.2.1.

Opérateurs de voisinage Les opérateurs de voisinage utilisés sont :

1. changement de règle de priorité (V1)
2. déplacement d'une tâche d'une ressource vers une autre, sous condition que cette dernière est apte à réaliser la tâche (V2)
3. composition des deux voisinages précédents : V3

Un exemple montrant ces trois voisinages est représenté dans la figure 4.3.

Solution initiale : par l'approche globale (cf 4.2.1.2) utilisant la règle de priorité EDD, nous déterminons une solution minimisant les deux critères. Par la suite, nous exécutons la Recherche tabou avec cette solution pendant un certain nombre d'itérations pour trouver le premier point proche du front de Pareto optimal. Dans cette phase, la meilleure solution voisine est celle qui minimise le coût global d'affectation. En cas d'égalité, la solution de meilleur retard maximum est privilégiée.

Intensification et diversification : une stratégie d'intensification et de diversification similaire à celle présentée dans la section 3.4.3.1 est utilisée. Dans cette méthode on explore un nombre de voisins égal au nombre de tâches (n) lors de l'intensification et $n/2$ lors d'une phase normale. après $n/2$ itérations sans améliorations de la solution, nous réinitialisons la solution courante en appelant l'approche globale (cf 4.2.1.2) avec une règle de priorité différente chaque fois.

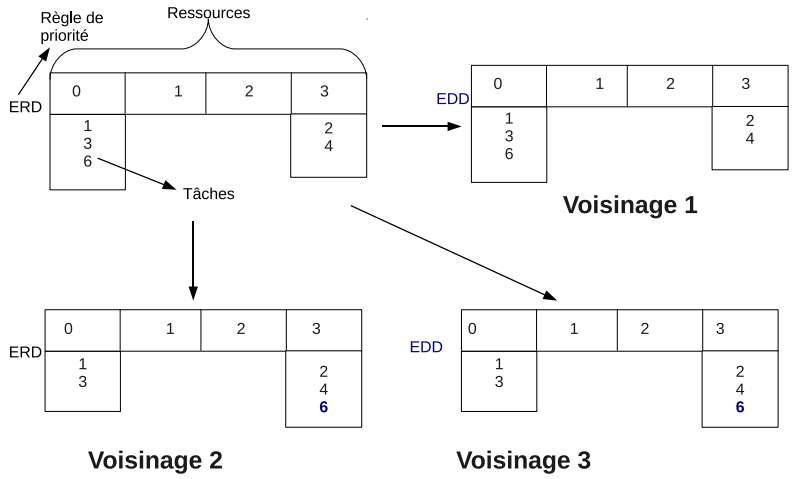


FIGURE 4.3 – Exemple de voisinages utilisés

Liste tabou : dans la Recherche tabou développée, nous avons limité la taille de la liste tabou à $TBMOVESIZE$. Cette Liste contient les mouvements et leur opposés, qui ont conduit à générer les dernières solutions courantes. Un mouvement tabou devient sélectionnable après $TBMOVESIZE$ itérations.

La méthode décrite ici est utilisée dans un schéma $\epsilon-$contrainte pour approximer le front de Pareto (cf Algorithme 10).

Algorithme 10: ϵ-contrainte

1: $P = tabu(T_{max}, cost)$
2: $MaxT_{max} \leftarrow bestT_{max}$
3: **pour** $L = MaxT_{max} - 1$ à 0 **faire**
4: $\quad S = tabu(T_{max} \leq L, cost)$
5: $\quad MaxT_{max} \leftarrow S.bestT_{max}$
6: $\quad P = ParetoFront(S, P)$
7: **fin pour**

Dans la ligne 1 du pseudo-code présentés dans l'algorithme 10, seul le coût total est optimisé par la Recherche tabou. Le T_{max} est considéré seulement lorsque plusieurs solutions de même coût sont générées. On note $MaxT_{max}$ le retard maximum de la solution calculée par la Recherche tabou. Les étapes du (3) à (7) ont pour objectif de trouver une solution minimisant le coût total et ayant un $T_{max} < MaxT_{max}$. $MaxT_{max}$ est mis à jour chaque fois qu'une solution faisable est trouvée.

4.2.2.2 Algorithme NSGA-II

Nous utilisons l'algorithme NSGA-II que nous avons décrit dans la partie 3.5.5.1 avec le codage et le décodage tel qu'il a été introduit pour mettre en place la Recherche tabou (cf 4.2.2.1) . La population initiale est générée de la même façon décrite dans la partie réactive du chapitre III (cf 3.5.5.1). Nous générons $2 * N$ (N : la taille de population) individus nécessaires, puis nous sélectionnons par élitisme N individus en basant sur les rangs et la distance de *crowding*. La sélection des individus pour participer à la reproduction de la nouvelle population n'est autre que la sélection par tournoi à 2 à l'aide du rang seulement, comme celle donc utilisée dans l'algorithme NSGA-II présenté dans la section 3.5.5.1 pour la résolution du *PMSPSP* réactif.

En ce qui concerne les opérateurs de croisement et mutation, ils sont spécifiques à notre problème. Ainsi, nous avons introduit de nouveaux opérateurs.

Croisement : soit F et M les deux parents à croiser pour produire deux nouveaux individus S et D. On choisit aléatoirement deux personnes dont les indices sont p et q ($p < q$). Notons n_p^F et n_p^M le nombre de tâches affectées à la personne p respectivement dans les individus F et M, et n_q^F et n_q^M le nombre de tâches affectées à la personne q respectivement dans les individus F et M. Soit :

1. $FT_p = \{FT_p^1, \ldots, FT_p^{n_p^F}\}$: l'ensemble des tâches affectées à la personne numéro p dans l'individu F,
2. $FT_q = \{FT_q^1, \ldots, FT_p^{n_q^F}\}$: l'ensemble des tâches affectées à la personne numéro q dans l'individu F,
3. $MT_p = \{MT_p^1, \ldots, MT_p^{n_p^M}\}$: l'ensemble des tâches affectées à la personne numéro p dans l'individu M,
4. $MT_q = \{MT_q^1, \ldots, MT_p^{n_q^M}\}$: l'ensemble des tâches affectées à la personne numéro q dans l'individu M.

La construction de l'enfant S est réalisée comme suit :

1. pour toutes les ressources différentes de p et q, les tâches sont héritées du parent F,
2. pour la personne p, on lui affecte toutes les tâches affectées à la personne q dans le parent M . Cette affectation est établie si et seulement si la personne p a les compétences nécessaires.
3. pour la personne q : on lui affecte toutes les tâches affectées à la personne p dans le parent M qu'elle peut réaliser,
4. on supprime tous les doublons générés. En effet, les tâches affectées à p ou à q peuvent être déjà attribuées à une autre ressource, et donc on les supprime de cette ressource
5. insertion des tâches "perdues" : les tâches, qui étaient affectées à p ou q dans le parent F peuvent être perdues sauf si elles font partie des tâches apportées par M. Nous les affectons aux même ressources que celles auxquelles elles sont affectées dans le parent M

La construction de l'enfant D est faite de la même façon en inversant le rôle des deux parents. La règle de priorité de l'enfant S est héritée du parent F et la règle de priorité de D est héritée du parent M.
La figure 4.4 décrit le croisement de deux parents. Il s'agit d'un exemple de 10 tâches et 5 personnes. Les valeurs de p et q sont respectivement 2 et 4. On suppose que la personne 2 ne peut faire que les tâches $(1, 3, 4, 5, 6, 8)$ et la personne 4 les tâches $(2, 3, 4, 5, 7, 8)$.

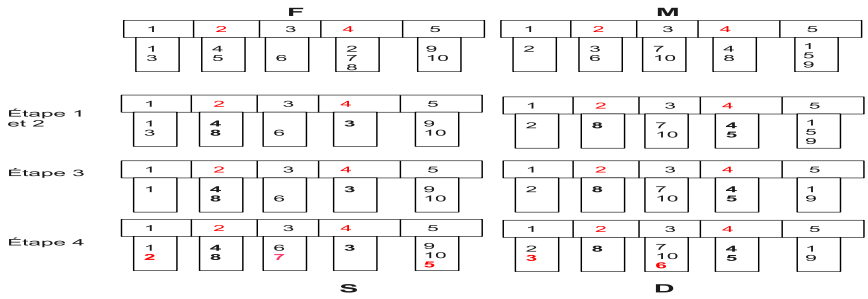


FIGURE 4.4 – Exemple de croisement de deux individus

Mutation : la mutation d'un individu applique les mêmes changements que dans le croisement. Deux personnes p et q différentes sont choisies, on affecte à la personne q (respectivement p) les tâches de p (respectivement q), si q (respectivement p) a les compétences avec un coût égal ou moins élevé. Les tâches affectées à p que q ne peut pas faire (ou si le coût pour les réaliser est plus élevé) restent à la charge de p. En même temps, la règle de priorité est changée aléatoirement.

4.3 Campagne de tests

4.3.1 Jeux de données

Dans ce modèle, les tests ont été menés en utilisant cinq groupes de données adaptés des instances de la *PSPLIB* [Kolisch *et al.*, 1996]. Pour les quatre premiers groupes, nous avons fixé le nombre de personnes à 5, le nombre de compétences à 3 et le taux de disponibilité de personnes à 95%. Le nombre moyen de tâches ayant une date de d'arrivée est fixé à 20% et le nombre moyen de tâches ayant des dates de fin souhaitée est de 25%. Le nombre moyen de compétences maîtrisées par personne est de 75% du nombre total de compétences nécessaires.
Le premier groupe de données ($DS1$) n'est autre que les 40 premières instances de 30 tâches de la *PSPLIB*. De façon similaire, le deuxième groupe ($DS2$) est l'adaptation des 40 premières instances de 60 tâches. Le troisième groupe ($DS3$) contient les 40 premières instances de 90 tâches et le quatrième groupe contient les 40 premières instances de 120 tâches.
Pour générer les dates de disponibilité des tâches, 20% des tâches sont tirées aléatoirement. Nous calculons les dates de début au plus tôt (ES_i) et dates de fin au plus tard (LF_i) en utilisant le graphe de précédence et l'horizon de planification comme date de fin imposée

du projet. Au début, la date de disponibilité r_i de chaque tâche est égale à sa date de début au plut tôt ES_i et sa date de fin souhaitée d_i est égale à sa date de fin au plus tard LF_i. Pour chaque tâche parmi celles sélectionnées, on tire une date de disponibilité dans l'intervalle $[ES_i, \frac{d_i - p_i + ES_i}{2}]$. Cette valeur de r_i est utilisée pour recalculer de nouveau les dates de début au plus tôt et dates de fin au plus tard en utilisant le graphe de précédence. Il est à noter que l'attribution d'une date de disponibilité à une tâche peut impliquer que d'autres tâches aient des nouvelles dates de disponibilité différentes de leur date de début au plus tôt. Pour les dates de fin souhaitée, nous procédons de la même façon, en sélectionnant aléatoirement 25% des tâches. Puis, pour chaque tâche sélectionnée, nous tirons aléatoirement une date de fin souhaitée dans l'intervalle $[\frac{s_i + p_i + d_i}{2}, d_i]$. On propage également cette modification sur le graphe de précédence pour mettre à jour les dates de fin souhaitée pour des autres tâches éventuelles. Une date de fin souhaitée ou une date de disponibilité d'une tâche n'est pas modifiée lorsque sa modification induit une infaisabilité du problème. Par exemple, si la date de fin souhaitée devient plus petite que la date de disponibilité de la tâche plus sa durée opératoire, le changement sur la date de disponibilité ou sur la date de fin souhaité n'est pas pris en compte.

Le cinquième jeu de données vise à générer des instances plus difficiles en termes de taille de fenêtres d'exécution et en termes de disponibilité de personnes. En effet, nous avons repris les mêmes paramètres que ceux utilisés pour générer les autres groupes de données à l'exception de taux de disponibilité des personnes que nous avons fixé à 50% au lieu de 95% et les intervalles de date de disponibilité et date de fin souhaitée pour la taâche i, qui deviennent respectivement $[r_i, d_i - p_i]$ et $[r_i + p_i, d_i]$. Nous avons généré un seul groupe de données $DS5$ à partir des 40 premières instances de 30 tâches de la *PSPLIB* mode simple.

4.3.2 Résultats expérimentaux

Dans cette section, nous présentons les résultats expérimentaux permettant d'analyser les performances des méthodes de résolution proposées. Les premiers résultats présentés dans ce paragraphe ont pour objectif de comparer les solutions obtenues par les algorithmes de liste lorsqu'aucun retard n'est toléré ($T_{max} = 0$), i.e la valeur de Q est égale à 0. Ainsi, les dates de fin souhaitée sont considérées comme des dates de fin impératives. Nous présentons dans le tableau 4.1 les résultats de l'heuristique $H3$. Les résultats des autres heuristiques de type séquentiel ne sont pas présentés vus qu'ils sont moins bons que H_3. Dans ce tableau, la deuxième colonne est constituée par le nombre d'instances où l'heuristique n'a pas trouvé une solution réalisable ; la troisième colonne donne la valeur moyenne des coûts des instances ayant un $T_{max} = 0$. Dans la dernière colonne, le temps d'exécution en secondes est indiqué.
Les tableaux 4.2 et 4.3 présentent les mêmes résultats obtenus respectivement par l'approche de résolution globale et la méthode tabou. Les résultats des deux heuristiques de liste $H3$ et l'approche globale ont été menés avec la règle EDD, car c'est celle-ci, qui a donné les meilleures résultats. Pour la Recherche tabou, les résultats présentés dans cette section, n'utilisent pas l'approche itérative, il s'agit donc de la solution de départ de l'approche itérative. Après des tests expérimentaux, nous avons limité le nombre d'itérations

à 150 et la taille de la liste tabou à 20.
Ces résultats montrent que les deux heuristiques n'ont pu résoudre aucune instance du groupe de données $DS3$. Pour les deux autres groupes, l'approche globale a résolu plus d'instances que l'heuristique $H3$. La méthode tabou était capable de résoudre 24 instances sur 40 du groupe de données $DS3$ et 36 du groupe de données $DS2$. Concernant le coût global, la méthode tabou a donné des meilleurs résultats par rapport aux deux heuristiques. Tandis que la différence entre les deux heuristiques en termes de coût n'est pas très significative puisque sur le jeu de données $DS1$, la méthode $H3$ est meilleure que l'approche globale et inversement pour le jeu de données $DS2$. La deuxième campagne de tests effec-

instances	# ($T_{max} > 0$)	coût moyen	temps(sec)
(DS1)	6	$340,38$	$0,013$
(DS2)	32	$698,22$	$0,041$
(DS3)	40	–	–

Tableau 4.1 – Résultats de l'heuristique H3

instances	# ($T_{max} > 0$)	coût moyen	temps(sec)
(DS1)	6	$354,57$	$0,015$
(DS2)	29	$691,61$	$0,045$
(DS3)	40	–	–

Tableau 4.2 – Résultats de l'heuristique globale

instances	# ($T_{max} > 0$)	coût moyen	temps(sec)
(DS1)	–	$327,18$	$1,321$
(DS2)	4	$549,78$	$7,025$
(DS3)	16	$912,47$	$17,489$

Tableau 4.3 – Résultats de la Recherche tabou

tuée, est dédiée au cas bicritère : minimiser le T_{max} et le coût d'affectation global.
Dans cette partie, nous présentons les résultats des heuristiques et de la méthode tabou sur les mêmes instances sans considérer la contrainte de $T_{max} = 0$. Nous nous intéressons surtout à la moyenne de chacun des deux critères : T_{max} et coût total. Dans les tableaux 4.4, 4.5, et 4.6), on reporte les valeurs du T_{max} minimum, moyen et maximum, ainsi que le coût global moyen pour chacune des méthodes sur les 5 jeux de données. Les résultats de ces tableaux peuvent être résumés comme suit :

1. La séparation de l'affectation de l'ordonnancement est moins efficace que l'approche globale, sur les deux critères. Ainsi, l'approche globale domine l'approche séquentielle ;
2. L'usage de la Recherche tabou a amélioré les résultats de l'heuristique globale sur les deux critères, moyennement en temps de calcul un peu plus important, mais il reste raisonnable. En effet, pour les jeux de données de 120 tâches, ce temps est de moins de 30 secondes ;

3. une dernière remarque est celle de l'amélioration du coût d'affectation global obtenu par les heuristiques en considérant des dates de fin souhaitée et non des dates de fin impératives. Cela est dû en fait au nombre de choix des personnes possibles. Car dans le cas des dates de fin impérative, les personnes aptes à réaliser une tâche peuvent ne pas y être affectées, si leur affectation entraîne un retard de la tâche (une infaisabilité dans ce cas).

instances	$(Min(T_{max})$)	$Avg(T_{max})$	$Max(T_{max})$	coût moyen	temps(sec)
(DS1)	0	0,45	9	367,271	0,013
(DS2)	0	0,55	15	637,025	0,043
(DS3)	0	0,625	17	959,25	0,1
(DS4)	0	2,4	50	1259,85	0,204
(DS5)	0	9,053	61	313,5	0,03

Tableau 4.4 – Résultats de l'heuristique H3

instances	$(Min(T_{max})$)	$Avg(T_{max})$	$Max(T_{max})$	coût moyen	temps(sec)
(DS1)	0	0,025	9	259,625	0.017
(DS2)	0	0,325	8	544,825	0.048
(DS3)	0	0,45	9	810,875	0.118
(DS4)	0	0,45	11	1069,775	0.269
(DS5)	0	3,368	61	285,131	0,023

Tableau 4.5 – Résultats de l'heuristique globale

instances	$(Min(T_{max})$)	$Avg(T_{max})$	$Max(T_{max})$	coût moyen	temps(sec)
(DS1)	0	0	0	204,325	1,25
(DS2)	0	0	0	400,975	6,626
(DS3)	0	0,25	5	614,175	15,976
(DS4)	0	0,3	7	850,15	29,23
(DS5)	0	0,892	17	211,054	1,479

Tableau 4.6 – Résultats de la Recherche tabou

Analysons maintenant les performances et la convergence du NSGA-II vers des solutions non strictement dominées. Cette analyse est faite en se basant sur la métrique hyper-volume (HV) introduite dans la section 3.5.6.1. Comme cette mesure nécessite un point de référence, qui doit être dominé par toutes les solutions, nous calculons deux bornes supérieures simples pour chacun des deux critères. Ces bornes supérieures sont utilisées comme des cordonnées du point de référence.

1. pour calculer une borne supérieure pour le T_{max}, nous posons l'horizon maximum du projet comme date de fin impérative du projet. Puis nous déduisons les dates de fin imposée de chaque tâche par propagation des durées sur le graphe de précédence. Le retard de chaque tâche est calculé en utilisant sa date de fin imposée comme date de fin réelle. Le retard maximum calculé est une borne supérieure valide de l'instance.

2. pour le coût total, on utilise pour chaque tâche la personne compétente pour la réaliser dont le coût est le plus élevé.

Nous normalisons les points du front de Pareto en divisant la valeur de chaque critère sur la borne supérieure. Le point de référence devient le point $(1, 1)$.

Nous avons effectué trois types de tests :

1. Choix des paramètres de l'algorithme : l'objectif de ces tests est de fixer le nombre de générations, la taille de population, le taux de croisement et le taux de mutation. Nos présentons dans le tableau 4.7, des tests, qui ont été effectués sur 12 instances de 30 tâches, nous reportons la valeur de l'hypervolume (HV) engendrée par le front de Pareto pour les différentes combinaisons des taux de croisement et mutation. Le nombre de générations, ainsi que la taille de la population, sont fixés respectivement à 400 et 100.

crois(%)	mut(%)	HV
80	1	0,697
80	5	0,733
80	8	0,756
80	10	**0,761**
80	15	0,758
100	1	0,691
100	5	0,724
100	8	0,746
100	10	0,746
100	15	0,755

Tableau 4.7 – Paramétrage du NSGA-II

A partir des expérimentations du tableau 4.7, nous avons fixé le taux de croisement à 80% et celui de la mutation à 10%.

2. Condition d'arrêt : l'objectif de ces tests est surtout de pouvoir fixer un nombre d'itérations permettant à l'algorithme de générer les meilleures solutions et d'être, ainsi le plus proche possible du front de Pareto optimal. Nous reportons dans le tableau 4.8 des tests menés sur les mêmes instances que dans le test de paramétrage. Le taux de croisement, ainsi que le taux de mutation sont fixés à 80% et 10% (résultat du tableau 4.7). Dans ce tableau, nous présentons le front de Pareto après chaque 50 générations en reportant la valeur de l'hypervolume correspondante.
   Même si NSGA-II continue pour certaines instances, à améliorer les résultats après 400 itérations. Cependant, nous avons constaté que ces améliorations ne sont pas significatives. Ainsi, dans les résultats expérimentaux présentés dans 4.3.1, le critère d'arrêt est fixé à 400 générations.
3. Test de stabilité : ces expérimentations ont pour but de mesurer la stabilité des résultats calculés par le NSGA-II. Nous utilisons les 12 mêmes instances, qui ont permis de

Inst	0	50	100	150	200	250	300	350	400
1	0,473	0,613	0,658	0,658	0,675	0,698	0,698	**0,768**	0,768
2	0,536	0,783	0,786	0,786	**0,797**	0,797	0,797	0,797	0,797
3	0,624	0,744	0,745	**0,801**	0,801	0,801	0,801	0,801	0,801
4	0,597	0,759	0,759	0,759	0,759	0,759	**0,778**	0,778	0,778
5	0,311	0,734	0,773	0,795	0,795	0,796	0,796	**0,805**	0,805
6	0,354	0,557	0,639	0,681	0,681	0,686	0,693	**0,7**	0,7
7	0,672	0,74	**0,804**	0,804	0,804	0,804	0,804	0,804	0,804
8	0,646	0,792	0,792	0,792	0,792	0,792	**0,819**	0,819	0,819
9	0,542	0,827	0,827	0,827	0,83	**0,843**	0,843	0,843	0,843
10	0,53	0,655	**0,787**	0,787	0,786813	0,787	0,787	0,787	0,787
11	0,332	0,405	0,475	0,484	0,49	0,503	0,504	0,504	0,512
12	0,545	0,717	0,717	0,717	**0,717**	0,717	0,717	0,717	0,717

Tableau 4.8 – Convergence du NSGA-II

reporter les résultats expérimentaux présentés dans les autres tableaux. Pour chaque instance, Nous exécutons le NSGA-II 10 fois et nous reportons les hypervolumes minimum, moyen et maximum pour chacune d'elles.

Comme nous le constatons dans la table 4.9, les résultats obtenus ne nous permettent

inst	HV_{MIN}	$\frac{(HV_{MOY}-HV_{MIN})}{HV_{MOY}}(\%)$	HV_{MOY}	HV_{MAX}	$\frac{(HV_{MAX}-HV_{MOY})}{HV_{MOY}}(\%)$
1	0,713	2,588	0,732	0,768	5,002
2	0,74	3,529	0,767	0,797	3,989
3	0,792	3,690	0,822	0,877	6,660
4	0,778	4,439	0,814	0,857	5,239
5	0,767	4,372	0,802	0,85	6,037
6	0,652	3,176	0,673	0,702	4,307
7	0,794	2,182	0,811	0,835	2,953
8	0,771	5,972	0,82	0,914	11,563
9	0,807	2,922	0,8313	0,848	2,016
10	0,681	4,468	0,713	0,787	10,324
11	0,472	6,627	0,505	0,5199	2,902
12	0,705	4,223	0,736	0,765	3,987

Tableau 4.9 – Test de stabilité du NSGA-II

pas de conclure clairement sur la stabilité du NSGA-II proposé.

Néanmoins une campagne de tests a été menée, étudiant la performance de l'algorithme par rapport à l'approche tabou 4.2.2.1. Le tableau 4.10 présente donc les résultats du $NSGA-II$ menés sur les 5 jeux de données où la métrique d'hypervolume est utilisée. Dans ce tableau, la première solution de la méthode tabou est calculée avec les mêmes paramètres générant les résultats du tableau 4.6, alors que pour chaque itération le nombre d'itérations est fixé à 50. Une liste tabou de taille égale à 20 est utilisée

Ces résultats montrent une nette domination du $NSGA-II$ par rapport à la méthode tabou. La qualité des solutions est très souvent meilleure et le temps d'exécution est beau-

coup moins élevé lorsqu'on utilise le $NSGA-II$. Nous remarquons cependant que le $NSGA-II$ donne des résultats moins bons avec les instances de plus grand nombre de tâches.

Jeux de données	NSGA-II		Recherche tabou	
	HV	Temps (sec)	HV	Temps (sec)
$DS1$	$0,728$	$17,494$	$0,441$	$59,601$
$DS2$	$0,678$	$52,875$	$0,414$	$239,526$
$DS3$	$0,656$	$135,314$	$0,367$	$176,686$
$DS4$	$0,642$	$247,38$	$0,352$	$555,48$
$DS5$	$0,707$	$23,334$	$0,486$	$30,519$

Tableau 4.10 – Résultats du NSGA-II et de la Recherche tabou sur différents jeux de données

4.4 Conclusion

Dans ce chapitre, nous avons étudié un modèle de gestion de projet à contraintes de ressource multi-compétences. Comme pour le cas de gestion de projets multi-compétences préemptif étudié en Chapitre III, ce modèle était défini conjointement avec des experts de gestion de projets d'informatique auprès de la société Néréide. Nous avons proposé une modélisation linéaire à variables mixtes, qui peut être utilisée pour résoudre des instances de petite taille.

Dans un deuxième temps, nous avons développé des méthodes heuristiques pour résoudre ce problème. Ces heuristiques ont été utilisées pour générer une seule solution du front de Pareto, notamment les deux points extrêmes du front. Elle peuvent être utilisées pour générer l'ensemble des solutions non-dominées. Vues les limites des heuristiques de liste, nous nous sommes logiquement orientés vers une étude des problèmes d'optimisation bicritère. Notre objectif est donc de développer des approches, qui nous permettent d'énumérer les fronts de Pareto. Ainsi, nous avons proposé un algorithme génétique de type $NSGA-II$, que nous avons comparé avec une Recherche tabou où l'approche considérée est l'$\epsilon-contrainte$. Les résultats expérimentaux montrent l'efficacité du $NSGA-II$ par rapport à la Recherche tabou.

Dans le cadre de cette thèse, des travaux ont été intégrés au progiciel de gestion intégrée *OFBiz*. Nous présentons dans le chapitre suivant, les différentes réalisations, qui ont été intégrées à *OFBiz*.

Chapitre 5

Implémentation logicielle

Dans ce chapitre, nous allons présenter l'implémentation des différentes méthodes et modèles que nous avons étudiés durant cette thèse. Nous abordons les points suivants :

1. Le modèle de gestion de projet existant,
2. Les technologies et outils utilisés,
3. Modules intégrés sur *OFBiz*.

5.1 Modèle de gestion de projet existant

Le début : les premières réflexions sur l'intégration d'un module de gestion de projet au sein d'OFBiz ont eu lieu vers le début mars 2007. Le premier *Jira*[1] créé pour cet objectif date du 08 Mars 2007. Une discussion autour de ce sujet est disponible ici https ://issues.apache.org/jira/browse/OFBIZ-796.

Parmi les objectifs initiaux annoncés par la communauté on trouve :

1. la simplicité de le planification et l'organisation du projet,
2. traitement de demandes d'une façon meilleure,
3. rajout de la possibilité d'intégration transparente avec la facturation,
4. etc.

Modèle de données : le modèle se base principalement sur les entités suivantes : projet, phase, tâche, compétence, ressource. Il y a d'autres entités pour la déclaration du temps passé par ressource.
Un projet dans OFBiz est découpé en plusieurs phases. Chaque phase contient une ou plusieurs tâches. Il y a des relations de précédence de type (fin-début) entre les tâches.
Une tâche est modélisée comme suit :

- Une tâche peut être associée à une ou plusieurs compétences avec une durée et un nombre de personnes. La durée d'une tâche est la somme des durées de ces compétences divisée sur le nombre de personnes.

1. un système de gestion et de suivi de projets

– S'il n' y a pas de compétence associée à la tâche, une durée de 24 heures est utilisée lors de la planification d'un projet.
– Une tâche doit appartenir à une phase.

Dans le modèle, on peut associer des ressources au projet ou à une tâche. Les ressources associées à une tâche sont utiles lors de la saisie du temps réel passé sur la tâche. Il n'y a pas de contraintes de compétences lors de l'association. La figure 5.1, présente un diagramme de classes représentant le modèle de gestion de projet multi-compétences. Notons que les noms des classes, ne correspondent pas forcément aux même noms dans *OFBiz*. Par exemple les classes **Project, Task, Phase** correspondent à l'entité **WorkEffort** dans *OFBiz*, et se distinguent par l'attribut *workEffortTypeId*. La classe **ProjectPeriodicity** a été rajoutée afin de pouvoir utiliser plusieurs périodicités (heure, jour, etc.), selon la taille ou nature du projet. L'attribut *projectPeriodicityId* a été donc rajouté à la classe *Project*. Dans la classe **Task**, on a un nouveau attribut afin de définir si une tâche est préemptive ou non.

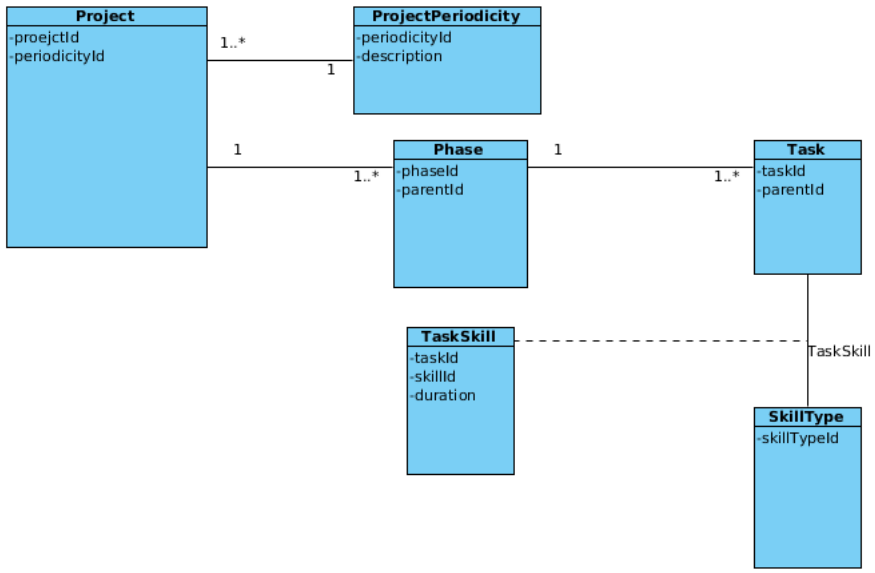


FIGURE 5.1 – Diagramme de classe pour un projet multi-compétences

Planification : il existe déjà un service qui fait la planification des tâches de projets qui prend en compte les relations de précédence et les durées de chaque tâche. Les contraintes de ressources ou de compétence ne sont pas considérées. La planification se base sur les hypothèses suivantes :

– une journée de travail est de 8H,
– une semaine est de 40 heures et 5 jours,
– l'ordre d'exécution respecte les relations de précédence,
– la date de début du projet est la date du jour,
– la durée par défaut d'une tâche est de 3 jours (24 heures).

Le tableau 5.1 donne les principaux points de différence entre le modèle existant et le modèle multi-compétences préemptif étudié en chapitre III.

Existant	Modèle multi-compétences préemptif
Plusieurs personnes par tâche/compétence	Une seule personne par tâche/compétence
Les compétences s'exécutent l'une après l'autre (car le temps des compétences est additionné)	Il s'exécutent parallèlement et si la préemption est autorisée la synchronisation est obligatoire
L'affectation des ressources est indépendante des compétences nécessaires	Les ressources doivent satisfaire les compétences de la tâche
Pas d'ordonnancement, seulement une planification	Des algorithmes d'ordonnancement sont implémentés
Pas de prise en considération de disponibilité des ressources	Prise en compte d'indisponibilité des ressources
Pas de préemption lors de la planification	On peut avoir des morceaux de tâche/comp si c'est autorisée
L'unité de mesure de temps est l'heure	Ici on peut la configurer (heure, demi journée, journée, etc.)

Tableau 5.1 – Modèle existant vs le modèle multi-compétences préemptif

Nouvelles classes pour l'ordonnancement : Afin de pouvoir tracer l'historique de l'exécution d'un algorithme d'ordonnancement, nous avons besoin d'une table (*MspspSchedule*) dans la base de données qui pour chaque lancement d'un algorithme d'ordonnancement, enregistre l'identifiant de cette exécution, les valeurs de chaque critère (*projectDuration*, *maxAssignChg* (le cas réactif)), la faisabilité de la solution calculée (isFeasible). Avec chaque ordonnancement, on associe le planning calculé (Table (*ProjectProposedPlanning*)). Dans le cas réactif, chaque solution dans le front de Pareto correspond à un ordonnancement. La figure 5.2 présente le diagramme de classes correspondant à l'ordonnancement.

5.2 Technologies utilisées

Mis à part le développement des algorithmes pour l'extraction des résultats qui a été réalisé en *Java*, les technologies et outils utilisés pour le développement ou l'intégration des modèles d'ordonnancement proposés ici sont principalement les technologies utilisées en *OFBiz* en plus de l'utilisation du moteur de règles *Drools*.

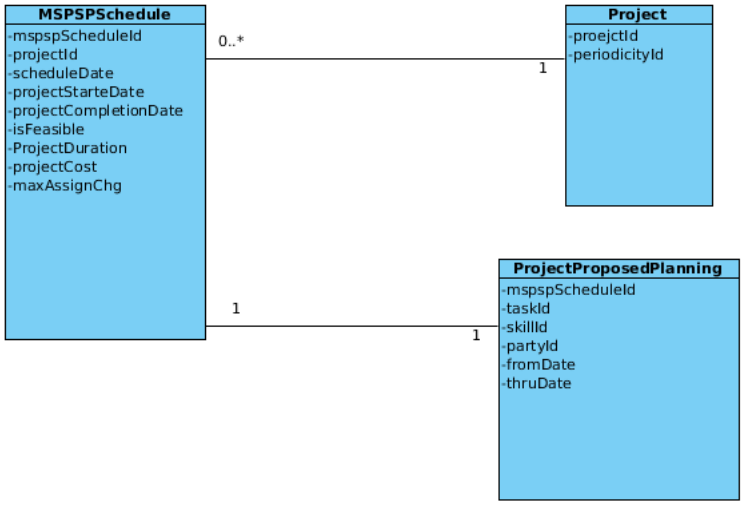


FIGURE 5.2 – Diagramme de classes pour l'ordonnancement d'un projet multi-compétences

5.2.1 Outils de développement *OFBiz*

OFBiz, d'un point de vue de développement, est un *framework* technique et fonctionnel. Il est fonctionnel du fait de vastes modules fonctionnels qu'il intègre par défaut (la gestion des acteurs, la gestion de stock, le commerce électronique et la gestion des points de vente, etc.). D'un coté, *OFBiz* propose un environnement technique très puissant qui permet la réalisation des nouveaux composants ou la personnalisation des composants existants. Le *framework* technique d'OFBiz est divisé en trois parties principales :

1. moteur d'entités : le modèle de données en *OFBiz* est basé sur le modèle relationnel de l'ouvrage *The Data Model Resource Book* [Silverston, 2011]. Le *framework* offre une gestion de base de données indépendante du système de gestion de base de données utilisé (*Oracle, Postgres, etc.*). Cette gestion est mise à disposition du développeur via l'utilisation des fichiers .xml pour la modélisation de la base (création des tables, des relations entre tables, modifications sur la structure de la base, etc.) et via des librairies java pour accéder aux données indépendamment de système de base de données.

2. moteur de services : les traitements de données se font via un moteur de services intégré. Les traitement peuvent être réalisés en utilisant un min-langage spécifique à *OFBiz* ou en développement en *Java* ou *Groovy*.

3. moteur d'affichage : l'interface graphique d'OFbiz est réalisée à l'aide des fichiers .xml qui se traduisent par le moteur de rendu de widgets (écrans, formes, menus) au code *Html* correspondant. Autre outils peuvent être utilisés pour réaliser des écrans, notamment le *FreeMarker Template Language (FTL)*.

5.2.2 Moteur de règles :

Un moteur de règles est un outil issu de l'application de l'intelligence artificielle sur la gestion des règles de métier. Le principe des moteurs de règles est : on définit d'une part un ensemble de règles (correspondant généralement à la logique d'un métier), et d'autre part, un ensemble de faits. L'application de ces règles sur ces faits, permet de déduire où inférer de nouveaux faits. Sur les nouveaux faits, les règles peuvent s'appliquer de nouveau et ainsi de suite, jusqu'à ce qu'il n'existe plus de règles applicables. L'inférence des faits se fait en utilisant un algorithme d'inférence, comme par exemple l'algorithme *Rete* [Zhou *et al.*, 2008].

Le moteur de règles que nous avons utilisé est celui de *JBoss*. Il s'agit de *Drools*. Le choix a été restreint par la licence et la facilité d'intégration à *OFBiz*. *Drools* est sous la licence **Apache**, on peut donc l'utiliser sans problème de licence. D'autre part *Drools* est écrit en java ce qui facilitera aussi son intégration à *OFBiz*. L'algorithme d'inférence utilisé à *Drools* est le *ReteOO* [Zhou *et al.*, 2008]. Il s'agit d'une version orientée objet de l'algorithme *Rete*.

L'idée initiale d'utiliser un moteur de règles, était entre autres de pouvoir exprimer des contraintes du problème sous forme des règles. Ceci permettra d'avoir à la fois :

1. la flexibilité des contraintes, i.e. on peut rajouter ou supprimer des règles sans changer la structure du problème et sans même avoir besoin de re-compiler le code, car ces règles sont stockées dans des fichiers texte (.drl ou .xml).
2. un autre avantage est la facilité de compréhension de ces règles. Car on peut utiliser un langage spécifique au métier (*Domain Specific Language* (*DSL*)) qui utilise la terminologie du métier, ce qui permet aux consultants fonctionnels par exemple, de comprendre facilement les règles et même les modifier ou créer des nouvelles règles.

Malgré ces avantages, les problèmes que nous voulons résoudre sont des problèmes *NP-difficile*, dont la résolution nécessite des algorithmes et des calculs souvent compliqués et coûteux en temps. On s'est intéressé d'abord à proposer des méthodes de résolution puis nous avons procédé à les implémenter en Java dans un premier temps. Dans un deuxième temps, nous nous sommes intéressés à voir si une implémentation par *Drools* est possible ou même bénéfique.

5.2.2.1 Première approche

La première approche que nous avons adoptée est d'utiliser le moteur de règles *Drools* pour implémenter certaine logique des algorithmes d'ordonnancement proposés. Cette approche a été utilisée pour implémenter l'algorithme *PAPSS* présenté en 3.4.1. On l'a utilisée aussi pour implémenter un algorithme d'ordonnancement des tâches sur une seule personne. Ce dernier algorithme est utilisé actuellement à Néréide pour planifier les tâches par personne.

Un inconvénient majeur pour utiliser *Drools* est l'obligation d'utiliser des objets et ne pas pouvoir utiliser des types de données primitives. Cela a un impact important sur le

temps d'exécution des règles. Par conséquent, la version de l'algorithme en utilisant *Drools* était plus coûteuse en termes de temps que la version implémentée en code java classique. Les résultats restent identiques dans les deux implémentations.

5.2.2.2 Deuxième approche (*Drools Planner*)

Heureusement, la communauté de *Drools* a un *framework* dédié à la planification. Ce *framework* appelé *Drools Planner* est un *framework* de métaheuristiques. Notamment -au moins pour le moment- une recherche tabou. D'ailleurs, plusieurs problèmes sont implémentés en *Drools Planner*, citons par exemple : le problème de rotation des *shifts* (*Employee shift rostering*), problème de planification dans le domaine d'éducation (planifications des leçons, des cours, des examens, etc.), problème de tournée de véhicules, etc.

L'idée de la méthode tabou implémentée en *Drools* est le suivant :

1. avoir une solution initiale avec un codage direct,
2. définir des mouvements ou utiliser les mouvements par défaut (échange entre deux attributs de la solution)
3. deux scores sont utilisés pour évaluer une solution :
   (a) hard score (pour les contraintes dures) : le critère principal à optimiser, et en général, on cherche à avoir une valeur de 0 si le but est d'avoir des solutions sans aucune contrainte dure violée.
   (b) soft score (pour les contraintes non dures ou les critères à optimiser) : cette fonction est considérée, si deux solutions ont la même valeur en termes du premier critère. Son usage correspond naturellement à une fonction objectif dans un problème d'optimisation monocritère.

   Des règles sont utilisées pour calculer ces scores. L'implémentation de ces règles dépend du problème. Souvent, un poids est associé à chaque contrainte violée et la somme de ces valeurs (poids) est calculée. Si les contraintes doivent toutes être respectées, le même poids peut être associé à toutes les contraintes.
   Le calcul des scores peut se faire d'une façon simple, i.e. vérifier toutes les contraintes violées pour les contraintes dures et calculer la valeur totale des contraintes légères. Cette méthode peut être coûteuse selon le problème. Un autre mécanisme, qui est utilisable, est celui de calcul incrémental qui permet d'induire le score d'une solution résultant d'un mouvement effectué sur une autre solution pour laquelle le score est connue. Cette dernière méthode est plus efficace, surtout si le codage des solutions est direct.

Actuellement, nous avons implémenté la méthode Tabou du problème *MSRMST* en utilisant *Drools planner*. Nous n'avons pas utilisé un codage direct, ce qui est le plus adapté à la philosophie de *Drools planner*. Cela dû au nature du problème qui est très combinatoire et le passage par un codage direct peut coûter beaucoup de temps en explorant des solutions non réalisables. Ceci dit, il serait intéressant de mener une étude expérimentale pour confirmer la dominance du codage direct par rapport au codage indirect.

5.3 Modules intégrés à *OFBiz*

Dans cette partie, nous présentons les différentes réalisations liées à la gestion et planification de projet qui ont fait l'objet de notre étude durant cette thèse.

5.3.1 *Add-on* de planification par personne

Les habitudes quotidiennes de Néréide en matière de gestion de projet consistent à affecter des tâches par personne et puis laisser la personne gérer le séquencement de ses tâches selon leur priorité. Cette priorité est définie par le responsable du projet ou par la personne créatrice de la tâche. Nous avons intégré un algorithme de séquencement des tâches par personne qui prend en compte les créneaux de disponibilité de la personne, la priorité des tâches, les durées et les relations de précédence. L'algorithme peut planifier toutes les tâches du projet ou d'une phase du projet affectées à une personne. Dans ce cas, les tâches à planifier peuvent avoir des prédécesseurs qui n'appartiennent pas à la même personne ou à la même phase. Pour prendre en considération ces contraintes de précédences, pour chaque tâche, nous introduisons une date de disponibilité égale à la date de fin maximum de ces prédécesseurs, même si elles n'appartiennent pas à la même personne ou ni à la même phase.

Dans cette méthode, nous utilisons des règles de mise à jour des priorités. Ces règles sont basées essentiellement sur le temps d'attente des tâches pour être planifiées. Des règles implémentées en *Drools* sont utilisées pour réévaluer la priorité d'une tâche selon ce critère.

Cet *add-on* est mis en production sur le système d'information de Néréide. Les détails de la méthode décrite ci-dessus sont bien expliqués dans l'*add-on sTaskfulfillMngmt (http ://addons.neogia.org/addons-12.04/sTaskfulfillMngmt/)* dans le fichier *HELP_ADDON_ sTaskfulfillMngmt_ FR.xml* sur le dossier *helpdata*.

5.3.2 *Add-ons* implémentant le modèle *PMSPSP*

Deux méthodes de résolution pour le cas prédictif ont été intégrées via des *add-ons*. Il s'agit de l'heuristique de liste (cf Section 3.4.1) et la méthode tabou à codage hérité (cf Section 3.4.3.2). Ainsi la méthode $NSGA-II$ pour le cas réactif (cf Section 3.5) est en cours d'intégration à *OFBiz*. Les *add-ons*qui intègrent ces méthodes ainsi que les interfaces graphiques et modèle de données nécessaires sont :

1. **projectPeriodMgr** (http ://addons.neogia.org/addons-dev/projectPeriodMgr/) : dans cet *add-on*, on trouve le code java nécessaire pour convertir un horizon de planification dont la date de début et date de fin sont connues. Ces dates sont données sous forme des instants de l'horizon de planification discrétisé en périodes de même longueur. La périodicité peut être de l'échelle heure, demi-journée ou encore journée. les fonctions principales sont :
   (a) calculer le numéro de la période dont une date donnée fait partie

(b) calculer la date de début et de fin à partir du numéro de la période

(c) calculer la période précédente (suivante) d'une période donnée

2. **mspsMapping** (http ://addons.neogia.org/addons-dev/mspsMapping/) : L'objectif de cet *add-on* est de faire un interface entre la base de données et les classes java du modèle. Les fonctions principales sont :

   (a) lecture des données d'un projet à partir de la base de données via des services java

   (b) alimenter les classes java utilisées dans les algorithmes d'ordonnancement à partir de ces données en utilisant le premier *add-on* pour transformer les dates à des périodes.

   (c) récupérer les résultats de l'ordonnancement et les transformer à une structure de données qui correspond au modèle de base de données

   (d) Appliquer les modifications nécessaires induites par le résultat de l'ordonnancement sur la base de données.

3. **mspspAlgo** (http ://addons.neogia.org/addons-dev/MSPSPMgr/) : dans cet *add-on*, nous avons migré le code java et structures de données que nous avons utilisés pour la mise en œuvre des algorithmes concernés. Cette migration a entrainé des fois quelques petites modifications sur le code original ou le rajout de code supplémentaire (par exemple, pour utiliser des logs).

Dans cette *add-on*, nous avons aussi l'interface graphique du modèle de gestion de projet multi-compétences étudié. Dans la Figure 5.3, nous avons une portlet permettant de planifier un projet en sélectionnant entre les deux méthodes (heuristique ou tabou). Puis, selon le choix de la méthode, on peut choisir des paramètres. Par exemple, dans le cas de la méthode tabou, on peut choisir entre trois critères d'arrêt (nombre d'itérations, nombre d'itérations sans améliorations, limite de temps). Dans le cas de l'algorithme heuristique on peut choisir une règle de priorité parmi un ensemble de règles.

FIGURE 5.3 – *Portlet* pour planifier un projet *PMSPSP*

Les résultats de l'application d'un algorithme d'ordonnancement sur un projet sont présentés de la façon suivante :

1. un enregistrement dans une table de la base de données est stocké pour chaque exécution de l'algorithme, et est affiché dans la *portlet* à droit en haut comme présenté dans la Figure 5.4. Cet enregistrement affiche l'identifiant de l'exécution, la date de début de planification, la faisabilité de l'ordonnancement, la date de début et de fin de projet proposées par l'ordonnancement

FIGURE 5.4 – *Portlets* d'affichage de résultats

2. la portlet en dessous, affiche la répartition proposée des charges entre les ressources (cf Figure 5.3)

FIGURE 5.5 – *Portlet* d'affichage du planning des activités

3. la portlet d'après (charges non planifiées), affiche les charges restantes à planifier par compétence. Ce portlet contient des informations indiquant la non faisabilité de l'ordonnancement. Dans ce cas, nous affichons les tâches/compétences que l'algorithme n'a pas pu placer. La dernière portlet affiche le détail de charges non planifiées par tâche/compétence
4. le détail de planning est affiché dans une autre portlet sur la même page (cf Figure 5.5). Dans cette portlet, on trouve le nom de la tâche, le nom de la compétence, la personne affectée (acteur) et la date de début et de fin du créneau d'exécution.
5. **mspspReactive** (http ://addons.neogia.org/addons-dev/mspspReactive/) cet *add-on* se base sur ceux présentés ci-dessus. Son objectif est d'apporter les modifications sur le modèle de données nécessaires pour le cas réactif, rajouter des services de lecture de données d'un projet en cours d'exécution, fournir l'interface graphique permettant de lancer et exploiter les résultats de l'algorithme NSGA-II présenté dans la Section 3.5, et intégrer le code de cette algorithme dans *OFBiz*. La figure 5.7 présente la

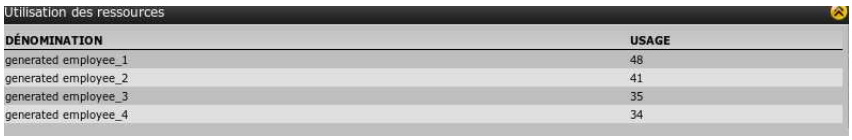


FIGURE 5.6 – *Portlet* de répartition des heures par personne

FIGURE 5.7 – *Portlet* de paramétrage de l'algorithme NSGA-II réactif

portlet utilisée pour lancer la méthode NSGA-II proposée pour l'ordonnancement. Les autres portlets présentant les résultats sont les mêmes que celles présentées ci-dessus pour le cas prédictif. La seule différence est que le lancement de cette méthode génère autant de plannings que de solutions dans le front de Pareto. L'utilisateur peut afficher chaque planning, ou simplement consulter les valeurs des deux critères de chaque planning proposé. Puis un seul planning sera validé.

5.4 Conclusion

Un modèle de base de données basé sur celui existant déjà en *OFBiz* a été développé. Des interfaces graphiques pour certaines méthodes ont été aussi réalisés.
Différentes méthodes ont été intégrées au *framework OFBiz* via des *add-ons*. Nous avons ainsi utilisé le moteur de règles *Drools* pour implémenter certains méthodes.

Conclusion générale et perspectives

Nos travaux, durant cette thèse, ont porté sur des problématiques de gestion de projets à contraintes de personnel multi-compétences. Deux problèmes d'ordonnancement de projets multi-compétences originaux ont été étudiés.

Dans le premier problème ($PMSPSP$), nous avons étudié un modèle de gestion de projet multi-compétences qui prend en considération la préemption de certaines tâches et la synchronisation des compétences d'une tâche au début de son exécution. Outre ces deux caractéristiques, nous considérons une charge qui peut être différente par compétence et nous imposons l'exécution d'une compétence d'une tâche par une unique personne. A notre connaissance, ces spécificités distinguent ce modèle des modèles d'ordonnancement de projets classiques. Dans un cadre global, nous cherchons à proposer une solution de base efficace en optimisant la date de fin de projet. Il s'agit de définir un planning prédictif. Nous avons étudié le cas réactif (dynamique), où un planning en cours doit être ajusté suite à des perturbations imprévues. Dans ce dernier cas, un problème bicritère est étudié.

Dans un deuxième temps, nous nous sommes intéressés à un modèle ($MSRMST$) dans lequel, nous considérons une seule compétence par tâche et une seule personne par tâche et par compétence. La notion de préemption a été modifiée par rapport au premier modèle : nous considérons ici, qu'aucune tâche ne peut être interrompue sauf si la personne en charge de son exécution n'est pas disponible (congés, formation, intervention sur un autre projet). D'autres contraintes ont été prises en considération. Il s'agit de dates de disponibilité et de fin souhaitée pour les tâches. De même, les coûts de réalisation ont été considérés pour ce deuxième modèle. Ainsi, l'exercice d'une compétence par une personne pendant une unité de temps est associé à un coût. Nous cherchons à trouver l'ensemble de solutions réalisables optimisant à la fois le retard maximum T_{max} et le coût global du projet. Ce coût peut modéliser soit un coût financier soit l'affinité des personnes à exercer une compétence. A notre connaissance, les problèmes d'ordonnancement de projet multi-compétences prenant en compte plusieurs critères ont été très peu abordés dans la littérature.

Nous avons donc défini formellement les deux modèles rappelés ci-dessus. Ensuite, nous avons établi une étude bibliographique afin d'identifier les problèmes de la littérature connexes aux deux cas étudiés.

L'étude du premier modèle, nous a amenés à proposer un programme mathématique linéaire à variables mixtes dont l'utilisation a permis, entre autres, de mesurer la performance de certaines méthodes approchées sur des instances de petite taille. Par la suite, nous avons proposé des bornes inférieures de la durée du projet. Ces bornes inférieures ont été également utilisées pour mesurer les performances des méthodes approchées sur des instances de taille plus importante. Comme méthodes de résolution approchées, une heuristique de liste basée sur des règles de priorité a été proposée dans un premier temps. Dans un second temps, nous avons élargi l'ensemble de solutions en proposant des métaheuristiques qui sont des algorithmes de type "Recherche tabou" et des algorithmes à base de population "Algorithmes génétiques". Trois recherches tabou à codage indirect et trois algorithmes génétiques ont été développés. Ensuite, une méthode Tabou à codage direct a été proposée. Une étude expérimentale a été menée afin de fixer les paramètres nécessaires pour chaque méthode et ensuite étudier leurs performances. Ces expérimentations ont montré l'efficacité des méthodes Tabou à codage indirect par rapport aux algorithmes génétiques. Cependant, la méthode Tabou à codage direct n'était pas assez performante par rapport aux méthodes développées utilisant un codage indirect.

Ces travaux ont fait l'objet des communications suivantes : [Dhib *et al.*, 2010], [Dhib *et al.*, 2011c], [Dhib *et al.*, 2011b], [Dhib *et al.*, 2011a].

La deuxième partie de ce chapitre a été consacrée à l'étude du cas dynamique du problème d'ordonnancement *PMSPSP*. Nous avons proposé un modèle bicritère où l'un des objectifs est de minimiser le nombre maximum de changements d'affectation des personnes suite à la prise en compte des aléas. Ensuite, un modèle mathématique linéaire à variables mixtes a été proposé. Intégré à une approche $\epsilon-$contrainte, ce modèle a été utilisé pour l'énumération du front de Pareto pour des instances de petite taille. Un algorithme génétique de type *NSGA-II* a été également proposé pour approximer le front de Pareto. Nous avons comparé les résultats de cette méthode avec les solutions exactes pour des instances de petite taille. Une étude de performance de *NSGA-II* sur d'autres instances a été réalisée en utilisant la métrique de performance *hypervolume*.

Le deuxième modèle d'ordonnancement de projet à contraintes de personnel multi-compétences a été abordé dans le chapitre IV. Pour ce modèle, nous avons proposé des heuristiques afin de résoudre ce problème en cherchant une seule solution minimisant le coût d'affectation global et dont le retard maximum est inférieur à une valeur Q. L'objectif ici était principalement d'analyser la pertinence des règles de séquencement et d'affectation. Bien entendu, ces heuristiques de liste peuvent être appliquées pour énumérer le front de Pareto selon l'approche $\epsilon-$contrainte. Un algorithme génétique de type $NSGA-II$ a été proposé pour l'approximation du front de Pareto. Plusieurs expérimentations ont été menées afin de fixer les différents paramètres de cette méthode. Nos compagnes de tests ont montré l'efficacité de l'algorithme basé sur NSGA-II.
Ces travaux ont fait l'objet des communications suivantes : [Dhib *et al.*, 2012a], [Dhib *et al.*, 2012b].

Ce travail de thèse a été réalisé dans le cadre d'une convention CIFRE. Ainsi, la dernière partie de ce manuscrit est dédiée à la mise en œuvre pratique des résultats de nos recherches pour le problème *PMSPSP*. L'heuristique de liste et une recherche Tabou permettant de calculer un ordonnancement prédictif, sont intégrées au sein de l'ERP *OFBiz* de la société *Néréide*. De même, pour l'élaboration d'un planning réactif l'algorithme $NSGA-II$ a été également implémenté dans le logiciel. L'intégration au logiciel libre des autres travaux réalisés durant cette thèse est en cours de réalisation.

Les perspectives de recherche qui découlent de ce travail sont nombreuses. A court et à moyen terme, nous envisageons de poursuivre nos études relatives aux problèmes d'ordonnancement de projet multi-compétences. A très court terme et d'un point de vue technique, il serait intéressant d'améliorer les bornes inférieures, en particulier, celles dites destructives proposées pour le cas *PMSPSP*. En effet, nous pensons que si le calcul des parties obligatoires et des marges libres prennent en considération le fait que chaque compétence doit être réalisée par la même personne et/ou les personnes affectées à la même tâche doivent être différentes, cela pourrait conduire à une amélioration significative de la qualité de ces bornes. Par la suite, nous envisageons de développer d'autres méthodes exactes de type Procédure par Séparation et Evaluation où ces bornes inférieures peuvent être utilisées. Cela nous permettrait de résoudre optimalement quelques instances réelles dont la taille est de l'ordre d'une trentaine de tâches avec une dizaine d'employés. Un autre avantage de développer une telle procédure, est de pouvoir énumérer le front de Pareto sans avoir recours à une approche de type $\epsilon-$contrainte.

Il serait également intéressant de mener d'autres campagnes de tests pour évaluer les performances de l'algorithme $NSGA-II$ proposé pour la résolution du problème d'ordonnancement de projet à ressources multi-compétences et tâches mono-compétence (MSRMS-T). Cette étude peut se faire en comparant $NSGA-II$ avec l'algorithme *SPEA2* proposé pour la résolution des problèmes d'ordonnancement de projets multi-critères [Ballestín et Blanco, 2011].
De même, il n'a pas échappé au lecteur que le codage utilisé dans les métaheuristiques proposées pour résoudre le premier modèle (chapitre III) est différent de celui utilisé pour résoudre le second problème (Chapitre IV). Par conséquent, nous envisageons de reprendre chaque métaheuristique avec ces différents codages et d'effectuer une nouvelle étude expérimentale. Cette nouvelle étude expérimentale nous permettra d'analyser plus finement l'impact du choix de codage pour la résolution du problème considéré.
Comme méthodes approchées, nous pensons dans un avenir proche développer une méthode de voisinage hybride de type *Matheuristique* faisant ainsi intervenir les programmes linéaires et les métaheuristiques proposées dans ce manuscrit [Della Croce *et al.*, 2011]. De même, une autre méthode hybride faisant appel à la programmation par contrainte et la programmation linéaire pourrait être envisagée [Demassey, 2003].
A moyen terme, d'autres modèles découlant directement de notre travail de thèse mériteraient d'être approfondis. En premier lieu, nous pouvons envisager d'étudier le cas réactif du second modèle, où un critère supplémentaire pourrait être considéré : la minimisation du nombre maximum de changements d'affectation. C'est l'un des critères identifiés conjointement avec les experts de gestion de projet. Ce critère n'a pour l'instant pas été considéré

pour ce modèle.

Dans un second temps, comme seules les approches réactives ont été considérées dans notre étude, il serait donc intéressant d'étudier les approches pro-actives [Billaut *et al.*, 2010]. En effet, dans un environnement réel, des informations sur les types d'aléas et leur probabilité d'occurrence peuvent être identifiées en se basant sur l'historique des activités de l'entreprise. Par conséquent, d'autres méthodes robustes peuvent être développées en s'inspirant de nos approches de résolution.

Un modèle particulièrement intéressant qui pourra faire l'objet d'une étude approfondie est celui de la résolution d'un problème d'ordonnancement dans un environnement multi-projets. Il est à noter que si un seul critère est considéré, les méthodes de résolution proposées dans ce manuscrit sont facilement adaptables. Dans le cas multiobjectif, une attention particulière sera donnée au cas du problème d'ordonnancement de projet multi-acteurs (*Multi-agent scheduling problem*) [Agnetis *et al.*, 2004] et [Soukhal, 2012]. Là encore, il s'agit d'une nouvelle classe de problèmes d'ordonnancement multicritères où chaque chef de projet (chaque projet) définit son propre critère à optimiser. Des solutions de meilleurs compromis doivent être trouvées entre les critères des différents projets et un critère global, appliqués à l'ensemble des tâches à réaliser.

Annexe

Annexe A

Déroulement de l'heuristique *PAPSS* et réduction d'une instance MSPSP à une instance PMSPSP

A.1 Déroulement de l'heuristique *PAPSS*

Dans cette section, on déroule l'algorithme *PAPSS* sur un exemple de 4 tâches, 2 personnes et 2 compétences. On utilise la règle *EDD*. L'horizon de planification est fixé à 8.

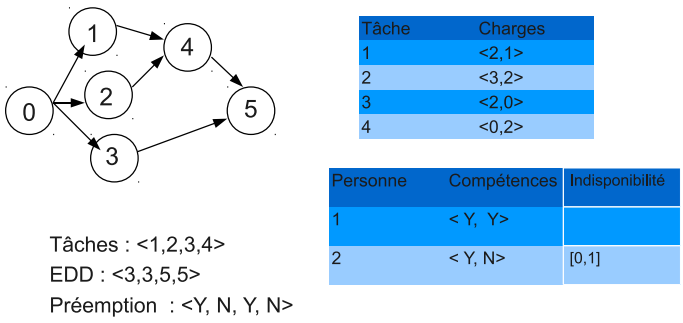


FIGURE A.1 – Instance du problème PMSPSP

A.2 Application de l'algorithme

A l'instant $t = 0$, les tâches A_1, A_2, A_3 peuvent être planifiées car elles ont pas de prédécesseurs. L'ordre selon la règle de priorité est : A_1, A_2, A_3. Mais les deux premières tâches ne peuvent pas démarrer à $t = 0$, car elles ont besoin de deux ressources chacune et une seule ressource est disponible. Ayant besoin d'une seule ressource, la tâche A_3 démarre donc à $t = 0$. A l'instant $t = 1$ la tâche A_1 démarre en préemptant A_3 (A_3 est préemptive

et moins prioritaire que A_1). A l'instant $t = 2$ la tâche A_2 démarre en préemptant les deux tâches A_1 et A_3. La préemption de la tâche A_1 est dû au fait que A_1 et A_2 ont la même priorité, mais A_2 nécessite plus d'une seule compétence. La tâche A_3 est préemptée car elle est moins prioritaire que A_2. A $t = 6$, on place la tâche 4 et l'algorithme s'arrête car toutes les tâches sont planifiées.

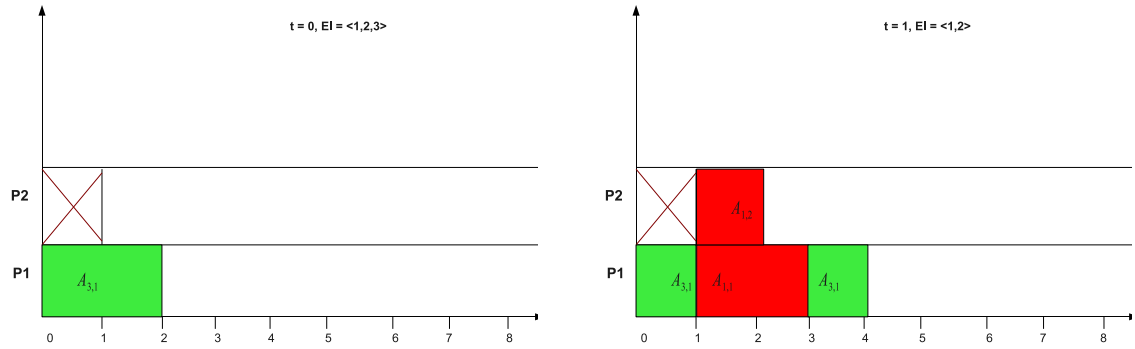


Tableau A.1 – planning partiel construit à t = 0 (à gauche) et à t = 1 (à droite)

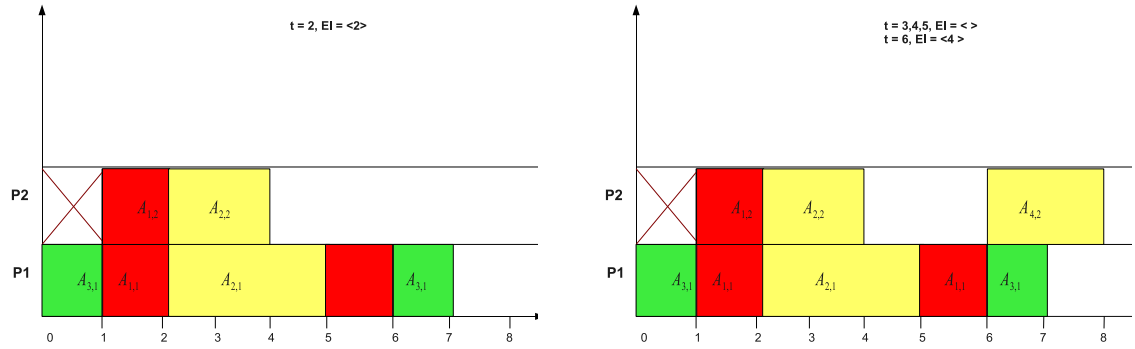


Tableau A.2 – planning partiel construit à t = 2 (à gauche) et planning complet construit à t = 6 (à droite)

A.3 Réduction d'une instance MSPSP à une instance du PMSPSP

Dans cette section nous démontrons que toute instance du problème *MSPSP* [Bellenguez-Morineau, 2006] peut être transformée à une instance du *PMSPSP*.

Preuve 1 *Soit I une instance du problème* MSPSP *[Bellenguez-Morineau, 2006]. Pour chaque compétence S_k on associe un ensemble de compétences S_k^I, tel que $\mid S_k^I \mid = \max_i p_{ik}$. Pour chaque compétence $S_k^{k'} \in S_k^I$ on associe les mêmes ressources que pour la compétence S_k. Pour chaque tâche A_i dans l'instance I dont la durée p_i et nécessite les compétences*

$\{S_{k_1}, S_{k_2}, \ldots, S_{k_m}\}$ *avec respectivement les nombres de personnes* $\{p_{i,1}, p_{i,2}, ...p_{i,m}\}$*, on associe* $A_i{}'$ *de la façon suivante : pour chaque compétence* S_{k_j} *nécessaire pour* A_i *avec* $p_{i,j}$ *personnes,* $A_i{}'$ *nécessite les premières* $p_{i,j}$ *compétences dans l'ensemble* $S^I_{k_j}$ *avec une charge équivalente à la durée* p_i *de la tâche* A_i*.*
De cette façon, on construit une instance I' *du problème* PMSPSP *en gardant les même relations de précédence que dans l'instance* I *du* MSPSP*, les mêmes disponibilités des ressources et compétences de chaque ressource. On remplace chaque tâche* A_i *dans l'instance du* MSPSP *par la tâche* $A_i{}'$ *comme décrit ci-dessus.*
Un example de 2 *tâches,* 2 *compétences et* 3 *personnes montrant cette transformation est présenté dans la figure A.2.*

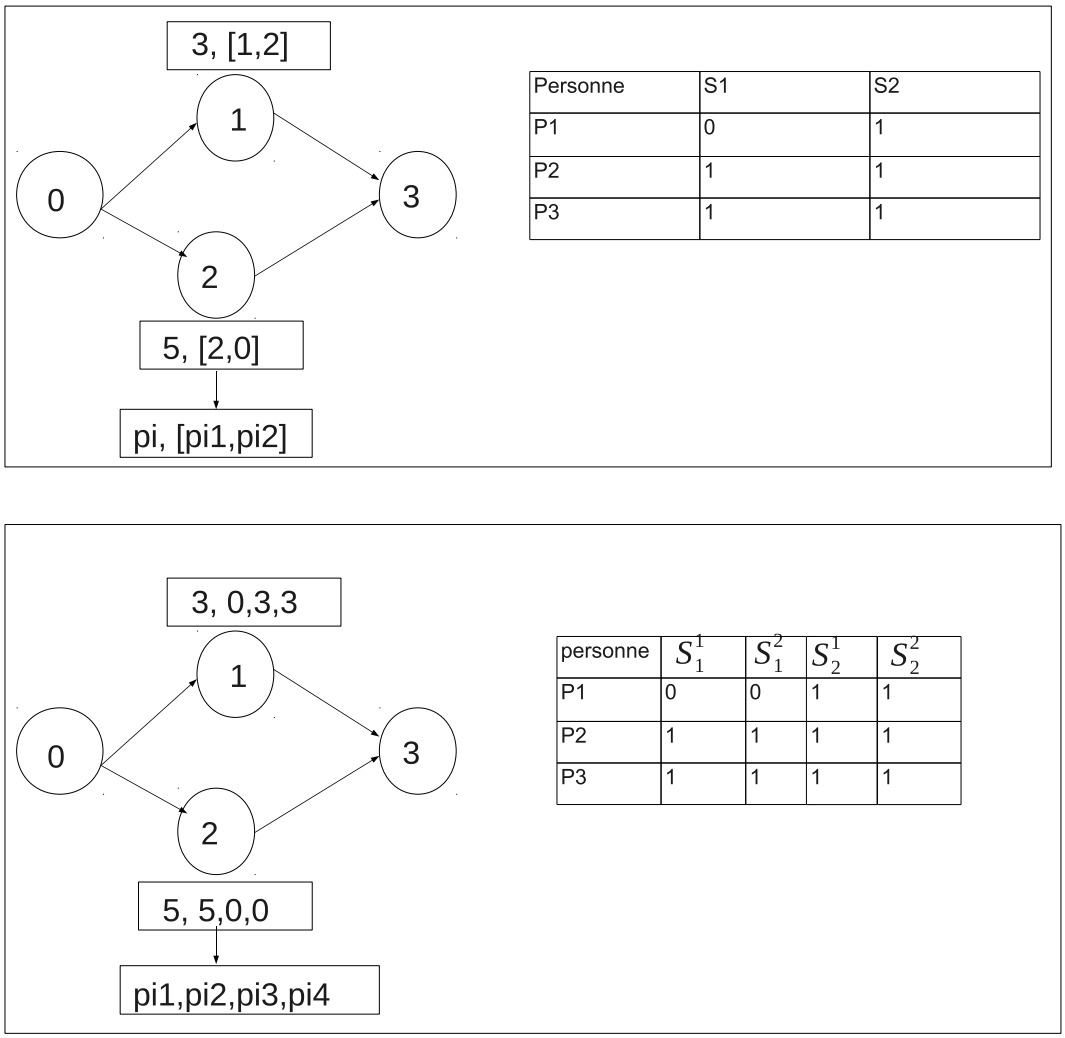


FIGURE A.2 – Réduction d'une instance *MSPSP* (en haut) à une instance *PMSPSP* (en bas)

Références bibliographiques

[Abbasi *et al.*, 2006] Abbasi, B., Shadrokh, S. et Arkat, J. (2006). Bi-objective resource-constrained project scheduling with robustness and makespan criteria. *Applied Mathematics and Computation*, 180(1):146–152.

[Achuthan et Hardjawidjaja, 2001] Achuthan, N. et Hardjawidjaja, A. (2001). Project scheduling under time dependent costs – a branch and bound algorithm. *Annals of Operations Research*, 108(1-4):55–74.

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

[Al-Fawzan et Haouari, 2005] Al-Fawzan, M. et Haouari, M. (2005). A bi-objective model for robust resource-constrained project scheduling. *International Journal of Production Economics*, 96(2):175–187.

[Artigues *et al.*, 2008] Artigues, C., Demassey, S. et Néron, E. (2008). *Resource-consrtrained project scheduling : Models, Algorithms, Extension and Applications*. Control systems, robotics and manufacturing series. Wiley.

[Artigues et Roubellat, 2000] Artigues, C. et Roubellat, F. (2000). A polynomial activity insertion algorithm in a multi-resource schedule with cumulative constraints and multiple modes. *European Journal of Operational Research*, 127(2):294–316.

[Ballestín et Blanco, 2011] Ballestín, F. et Blanco, R. (2011). Theoretical and practical fundamentals for multi-objective optimisation in resource-constrained project scheduling problems. *Computers & Operations Research*, 38(1):51–62.

[Ballestín *et al.*, 2006] Ballestín, F., Valls, V. et Quintanilla, S. (2006). Due dates and RCPSP. *In* J.Józefowska et J.Weglarz, éditeurs : *Perspectives in Modern Project Scheduling*, volume 92 de *International Series in Operations Research & Management Science*, pages 79–104. Springer US.

[Baptiste *et al.*, 1999] Baptiste, P., Pape, C. L. et Nuijten, W. (1999). Satisfiability tests and time-bound adjustments for cumulative scheduling problems. *Annals of Operations Research*, 92(0):305–333.

[Bellenguez-Morineau, 2006] Bellenguez-Morineau, O. (2006). *Méthodes de résolution pour un problème de gestion de projet avec prise en compte de compétences*. Thèse de doctorat, Université de Tours, Tours, France.

[Bellenguez-Morineau et Néron, 2004a] Bellenguez-Morineau, O. et Néron, E. (2004a). Lower bounds for the multi-skill project scheduling problem with hierarchi-

cal levels of skills. *In Practice and Theory of Automated Timetabling (PATAT2004)*, pages 429–432, Pittsburgh, PA, USA.

[Bellenguez-Morineau et Néron, 2004b] Bellenguez-Morineau, O. et Néron, E. (2004b). Methods for solving the multi-skill project scheduling problem. *In 9th International Workshop on Project Management and Scheduling (PMS2004)*, pages 66–69, Nancy, France.

[Bianco *et al.*, 1999] Bianco, L., Caramia, M. et Dellólmo, P. (1999). Solving a preemptive project scheduling problem with coloring techniques. *In* Węglarz, J., éditeur : *Project Scheduling*, volume 14 de *International Series in Operations Research & Management Science*, pages 135–145. Springer US.

[Bianco *et al.*, 1998] Bianco, L., Dell'Olmo, P. et Speranza, M. (1998). Heuristics for multimode scheduling problems with dedicated resources. *European Journal of Operational Research*, 107(2):260–271.

[Billaut *et al.*, 2010] Billaut, J., Moukrim, A. et Sanlaville, E. (2010). *Flexibility and Robustness in Scheduling*. Control systems, robotics and manufacturing series. Wiley.

[Blazewicz *et al.*, 1983] Blazewicz, J., Lenstra, J. et Rinnooy Kan, A. (1983). Scheduling subject to resource constraints : Classification and complexity. *Discrete Applied Mathematics*, 5(1):11–24.

[Bomsdorf et Derigs, 2008] Bomsdorf, F. et Derigs, U. (2008). A model, heuristic procedure and decision support system for solving the movie shoot scheduling problem. *OR Spectrum*, 30(4):751–772.

[Brânzei *et al.*, 2002] Brânzei, R., Ferrari, G., Fragnelli, V. et Tijs, S. (2002). Two approaches to the problem of sharing delay costs in joint projects. *Annals of Operations Research*, 109(1-4):359–374.

[Brooks et White, 1965] Brooks, G. et White, C. (1965). An algorithm for finding optimal or near optimal solutions to the production scheduling problem. *Journal of Industrial Engineering*, 16:34–40.

[Brucker, 1995] Brucker, P. (1995). *Scheduling algorithms*. Springer-Verlag.

[Brucker *et al.*, 1999] Brucker, P., Drexl, A., Möhring, R., Neumann, K. et Pesch, E. (1999). Resource-constrained project scheduling : Notation, classification, models, and methods. *European Journal Journal of Operational Research*, 112(1):3–41.

[Brucker et Knust, 2001] Brucker, P. et Knust, S. (2001). Resource-constrained project scheduling and timetabling. *In* Burke, E. et Erben, W., éditeurs : *Practice and Theory of Automated Timetabling III*, volume 2079 de *Lecture Notes in Computer Science*, pages 277–293. Springer Berlin / Heidelberg.

[Brucker et Schlie,] Brucker, P. et Schlie, R. *Job-shop Scheduling with Multi-purpose Machines*, volume 128 de *P.*

[Buddhakulsomsiria et Kim, 2006] Buddhakulsomsiria, J. et Kim, D. (2006). Properties of multi-mode resource-constrained project scheduling problems with resource vacations and activity splitting. *European Journal of Operational Research*, 175(1):279–295.

[Buddhakulsomsiria et Kim, 2007] Buddhakulsomsiria, J. et Kim, D. (2007). Priority rule-based heuristic for multi-mode resource-constrained project scheduling problems

with resource vacations and activity splitting. *European Journal of Operational Research*, 178(2):374–390.

[Busacker et Gowen, 1961] Busacker, R. et Gowen, P. (1961). Procedure for determining a family of minimal-cost network flow patterns. Rapport technique, Operational Research Office, Johns Hopkins University, Baltimore.

[Calhoun *et al.*, 2002] Calhoun, K., Deckro, R., Moore, J., Chrissis, J. et Hove, J. (2002). Planning and re-planning in project and production scheduling. *Omega*, 30(3):155–170.

[Carlier et Chrétienne, 1988] Carlier, J. et Chrétienne, P. (1988). *Problèmes d'ordonnancement : Modélisation, complexité, algorithmes*. Etudes et recherches en informatique. Masson.

[Carlier et Néron, 2000] Carlier, J. et Néron, E. (2000). A new LP-based lower bound for the cumulative scheduling problem. *European Journal of Operational Research*, 127(2):363–382.

[Chiu et Tsai, 2002] Chiu, H. et Tsai, D. (2002). An efficient search procedure for the resource-constrained multi-project scheduling problem with discounted cash flows. *Construction Management and Economics*, 20(1):55–66.

[Chtourou et Haouari, 2008] Chtourou, H. et Haouari, M. (2008). A two-stage-priority-rule-based algorithm for robust resource-constrained project scheduling. *Computers and Industrial Engineering*, 55(1):183–194.

[Correia *et al.*, 2010] Correia, I., co, L. L. et da Gama, F. S. (2010). Project scheduling with flexible resources : formulation and inequalities. *OR Spectrum*, 33(3):1–29.

[Czyzak et Jaszkiewicz, 1998] Czyzak, P. et Jaszkiewicz, A. (1998). A pareto simulated annealing – a metaheuristic for multiple objective combinatorial optimization. *Journal of Multi-Criteria Decision Analysis*, 7(1):34–47.

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

[de Vonder *et al.*, 2007] de Vonder, S. V., Demeulemeester, E. et Herroelen, W. (2007). A classification of predictive-reactive project scheduling procedures. *Journal of Scheduling*, 10(3):195–207.

[Deb *et al.*, 2000] Deb, K., Agrawal, S., Pratap, A. et Meyarivan, T. (2000). A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization : NSGA-II. Rapport technique, Kanpur Genetic Algorithms Laboratory (KanGAL), Indian Institute of Technology Kanpur.

[Debels et Vanhoucke, 2008] Debels, D. et Vanhoucke, M. (2008). The impact of various activity assumptions on the lead time and resource utilization of resource-constrained projects. *Computers and Industrial Engineering*, 54(1):140–154.

[Della Croce *et al.*, 2011] Della Croce, F., Grosso, A. et Salassa, F. (2011). A matheuristic approach for the two-machine total completion time flow shop problem. *Annals of Operations Research*, pages 1–12.

[Demassey, 2003] Demassey, S. (2003). *Programmation par Contraintes et Programmation Linéaire pour Problème d'Ordonnancement de Projet a Contraintes de Ressources.* Thèse de doctorat, Université d'Avignon et des Pays de Vaucluse, Avignon, France.

[Demeulemeester, 1992] Demeulemeester, E. (1992). *Optimal algorithms for various classes of multiple resource-constrained project scheduling problems.* Thèse de doctorat, Université Catholique, Leuven, Belgique.

[Demeulemeester et Herroelen, 1992] Demeulemeester, E. et Herroelen, W. (1992). A branch-and-bound procedure for the multiple resource-constrained project scheduling problem. *European Journal of Operational Research*, 38(12):1803–1818.

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

[Demeulemeester et Herroelen, 1997] Demeulemeester, E. et Herroelen, W. (1997). New benchmark results for the resource-constrained project scheduling problem. *Management Science*, 43(11):1485–1492.

[Dhib *et al.*, 2011a] Dhib, C., kooli, A., Soukhal, A. et Néron, E. (2011a). Lower bounds for a multi-skill scheduling problem. *In Proceedings of the International Conference on Operations Research (OR'2011)*, pages 471–476, Zurich, Switzerland.

[Dhib *et al.*, 2011b] Dhib, C., Néron, E. et Soukhal, A. (2011b). A multi-skill project scheduling problem in an industrail context. *In Proceedings of the 4th International Conference on Industrial Engineering and Systems Management (IESM'2011)*, pages 316–325, Metz, France.

[Dhib *et al.*, 2011c] Dhib, C., Roumdani, T., Soukhal, A. et Néron, E. (2011c). Bornes inférieures pour un problème d'ordonnancement de projets multi-compétence préemptif. *In ROADEF'2011*, Saint Etienne, France.

[Dhib *et al.*, 2010] Dhib, C., Soukhal, A. et Néron, E. (2010). Un problème de gestion de projet à contrainte de personnel : une approche de résolution par règles. *In ROADEF'2010*, Toulouse, France.

[Dhib *et al.*, 2012a] Dhib, C., Soukhal, A. et Néron, E. (2012a). Multi-skilled resource mono-skill task project scheduling problem (MSRMST). *In Proceedings of the 13th International Workshop on Project Management and Scheduling (PMS'2012)*, pages 137–140, Leuven, Belgium.

[Dhib *et al.*, 2012b] Dhib, C., Soukhal, A. et Néron, E. (2012b). Un problème d'ordonnancement de projet avec ressources multi-compétences et tâches mono-compétence. *In ROADEF'2012*, Angers, France.

[Dodin et Elimam, 2001] Dodin, B. et Elimam, A. (2001). Integrated project scheduling and material planning with variable activity duration and rewards. *IIE Transactions*, 33:1005–1018.

[Dorndorf *et al.*, 1999] Dorndorf, U., Huy, T. P. et Pesch, E. (1999). A survey of interval capacity consistency tests for time- and resource-constrained scheduling. *In* Weglarz, J., éditeur : *Project Scheduling*, volume 14 de *International Series in Operations Research & Management Science*, pages 213–238. Springer US.

[Dorndorf *et al.*, 2000] Dorndorf, U., Pesch, E. et Huy, T. P. (2000). Constraint propagation techniques for the disjunctive scheduling problem. *Artificial Intelligence*, 122(2):189–240.

[Drexl *et al.*, 2000] Drexl, A., Nissen, R., Patterson, J. et Salewski, F. (2000). Progen/px - an instance generator for resource-constrained project scheduling problems with partially renewable resources and further extensions. *European Journal of Operational Research*, 125(1):59–72.

[Drezet, 2005] Drezet, L.-E. (2005). *Résolution d'un problème de gestion de projets sous contraintes de ressources humaines : de l'approche prèdictive à l'approche réactive.* Thèse de doctorat, Université de Tours, Tours, France.

[Drezet, 2008] Drezet, L.-E. (2008). *Resource-consrtrained project scheduling : Models, Algorithms, Extension and Applications*, chapitre RCPSP with Financial Costs, pages 213–226. Wiley.

[Drezet et Billaut, 2008] Drezet, L.-E. et Billaut, J.-C. (2008). A project scheduling problem with labour constraints and time-dependent activities requirements. *European Journal of Operational Research*, 112(1):217–225.

[Elmaghraby, 1977] Elmaghraby, S. (1977). *Activity networks : Project planning and control by network models.* Wiley.

[Ernst *et al.*, 2004] Ernst, A., Jiang, H., Krishamoorthy, M. et Sier, D. (2004). Staff scheduling and rostering : a review of applications, methods and models. *European Journal of Operational Research*, 153(1):3–27.

[Erschler *et al.*, 1991] Erschler, J., Lopez, P. et Thuriot, C. (1991). Raisonnement temporel sous contraintes de ressources et problèmes d'ordonnancement. *Revue d'Intelligence Artificielle*, 5(3):7–32.

[Franck *et al.*, 2001] Franck, B., Neumann, K. et Schwindt, C. (2001). Project scheduling with calendars. *OR Spektrum*, 23:325–334.

[Gürbüz, 2010] Gürbüz, E. (2010). A genetic algorithm for biobjective multi-skill project scheduling problem with hierarchical levels of skills. Mémoire de D.E.A., Middle East Technical University, Ankara, Turkey.

[Hansen, 1997] Hansen, M. (1997). Tabu search for multiobjective optimization : Mots. *In The 13th International Conference on Multiple Criteria Decision Making*, pages 6–10, University of Cape Town.

[Haouari *et al.*, 2012] Haouari, M., Kooli, A. et Néron, E. (2012). Enhanced energetic reasoning-based lower bounds for the resource constrained project scheduling problem. *Computers & Operations Research*, 39(5):1187–1194.

[Hartmann, 1998] Hartmann, S. (1998). A competitive genetic algorithm for resource constrained project scheduling. *Naval Research Logistics*, 45(7):733–750.

[Hartmann, 1999] Hartmann, S. (1999). *Project scheduling under limited resources : Models, methods, and applications*, volume 478 de *Lecture Notes in Economics and Mathematical Systems.* Springer, Berlin, Germany.

[Hartmann, 2001] Hartmann, S. (2001). Project scheduling with multiple modes : A genetic algorithm. *Annals of Operations Research*, 102(1-4):111–135.

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

[Hartmann et Kolisch, 2000] Hartmann, S. et Kolisch, R. (2000). Experimental evaluation of state of the art heuristics for the resource constrained project scheduling problem. *European Journal of Operational Research*, 127(2):394–407.

[Heimerl et Kolisch, 2010] Heimerl, C. et Kolisch, R. (2010). Scheduling and staffing multiple projects with a multi-skilled workforce. *AOR Spectrum*, 32(2):343–368.

[Herroelen et Leus, 2000] Herroelen, W. et Leus, R. (2000). Project scheduling under uncertainty-survey and research potentials. *European Journal of Operational Research*, 165(2):289–306.

[Jarboui *et al.*, 2008] Jarboui, B., Damak, N., Siarry, P. et Rebai, A. (2008). A combinatorial particle swarm optimization for solving multi-mode resource-constrained project scheduling problems. *Applied Mathematics and Computation*, 195(1):299–308.

[Jou, 2005] Jou, C. (2005). A genetic algorithm with sub-indexed partitioning genes and its application to production scheduling of parallel machines. *Computers and Industrial Engineering*, 48(1):39–54.

[Jurisch, 1992] Jurisch, B. (1992). *Scheduling Jobs in Shops with Multi-purpose Machines.* Thèse de doctorat, Fachbereich Mathematik/Informatik, Universitat Osnabruck.

[Kelly, 1963] Kelly, J. (1963). *Industrial scheduling*, chapitre The critical-path method : Resources planning and scheduling, pages 347–365. Prentice-Hall international series in management. Prentice-Hall.

[Kimms, 2001] Kimms, A. (2001). Maximizing the net present value of a project under resource constraints using a lagrangian relaxation based heuristic with tight upper bounds. *Annals of Operations Research*, 102(1-4):221–236.

[Kis, 2005] Kis, T. (2005). A branch-and-cut algorithm for scheduling of projects with variable-intensity activities. *Mathematical Programming*, 103(3):515–539.

[Kis, 2006] Kis, T. (2006). RCPS with variable intensity activities and feeding precedence constraints. *In* Józefowska, J. et Weglarz, J., éditeurs : *Perspectives in Modern Project Scheduling*, volume 92 de *International Series in Operations Research & Management Science*, pages 105–129. Springer US.

[Klein, 2000a] Klein, R. (2000a). Project scheduling with time-varying resource constraints. *International Journal of Production Research*, 38(16):3937–3952.

[Klein, 2000b] Klein, R. (2000b). *Scheduling of Resource-Constrained Projects.* Operations Research/Computer Science Interfaces Series. Kluwer Academic.

[Klein et Scholl, 1999] Klein, R. et Scholl, A. (1999). Computing lower bounds by destructive improvement : An application to resource-constrained project scheduling. *European Journal of Operational Research*, 112(2):322–346.

[Klein et Scholl, 2000a] Klein, R. et Scholl, A. (2000a). Progress : Optimally solving the generalized resource-constrained project scheduling problem. *Mathematical Methods of Operations Research*, 52(3):467–488.

[Klein et Scholl, 2000b] Klein, R. et Scholl, A. (2000b). Scattered branch and bound -an adaptive search strategy applied to resource-constrained project scheduling. *Central European Journal of Operations Research*, 7:77–201.

[Kobylanski et Kuchta, 2007] Kobylanski, P. et Kuchta, D. (2007). A note on the paper by M. A. Al-Fawzan and M. Haouari about a bi-objective problem for robust resource-constrained project scheduling. *International Journal of Production Economics*, 107(2): 496–501.

[Kolisch, 1996] Kolisch, R. (1996). Serial and parallel resource-constrained project scheduling methods revisited : Theory and computation. *European Journal of Operational Research*, 90(2):320–333.

[Kolisch, 2000] Kolisch, R. (2000). Integrated scheduling, assembly area- and part-assignment for large-scale, make-to-order assemblies. *International Journal of Production Economics*, 64(1-3):127–141.

[Kolisch et Hartmann, 2006] Kolisch, R. et Hartmann, S. (2006). Experimental investigation of heuristics for resource-constrained project scheduling : An update. *European Journal of Operational Research*, 174(1):23–37.

[Kolisch et Padman, 2001] Kolisch, R. et Padman, R. (2001). An integrated survey of deterministic project scheduling. *Omega*, 29(3):249–272.

[Kolisch *et al.*, 1996] Kolisch, R., Sprecher, A. et Drexl, A. (1996). PSPLIB-a project scheduling library. *European Journal of Operational Research*, 96(1):205–216.

[Kooli *et al.*, 2012] Kooli, A., Haouari, M., Néron, E. et Carlier, J. (2012). A preemptive bound for the rcpsp. *In 13th International Workshop on Project Management and Scheduling (PMS'2012)*, pages 191–194, Leuven, Belgium.

[Li et Womer, 2009] Li, H. et Womer, K. (2009). Scheduling projects with multi-skilled personnel by a hybrid MILP/CP benders decomposition algorithm. *Journal of Scheduling*, 12(3):281–298.

[Lopez *et al.*, 1992] Lopez, P., Erschler, J. et Esquirol, P. (1992). Ordonnancement de tâches sous contraintes : Une approche énergétique. *Automatique, Productique, Informatique Industrielle*, 26:453–481.

[Mika *et al.*, 2005] Mika, M., Waligóra, G. et Węglarz, J. (2005). Simulated annealing and tabu search for multi-mode resource-constrained project scheduling with positive discounted cash flows and different payment models. *European Journal of Operational Research*, 164(3):639–668.

[Mingozzi *et al.*, 1998] Mingozzi, A., Maniezzo, V., Ricciardelli, S. et Bianco, L. (1998). An exact algorithm for the resource-constrained project scheduling problem based on a new mathematical formulation. *Management Science*, 44(5):714–729.

[Mónica et Tereso, 2010] Mónica, A. S. et Tereso, A. P. (2010). On the multi-mode, multi-skill resource constrained project scheduling problem (mrcpsp-ms). *In 2nd International Conference on Engineering Optimization*, Lisbon, Portugal.

[Montoya *et al.*, 2010] Montoya, C., Bellenguez-Morineau, O. et Rivreau, D. (2010). MIP Models for the Multi-Skill Project Scheduling Problem (MSPSP). Rapport technique, École des Mines de Nantes.

[Neumann *et al.*, 2002] Neumann, K., Schwindt, C. et Zimmermann, J. (2002). Recent results on resource-constrained project scheduling with time windows : Models, solution methods, and applications. *Central European Journal of Operations Research*, 10(2):113–148.

[Nonobe et Ibaraki, 2002] Nonobe, K. et Ibaraki, T. (2002). Formulation and tabu search algorithm for the resource constrained project scheduling problem. *In Essays and Surveys in Metaheuristics*, volume 15 de *Operations Research/Computer Science Interfaces Series*, pages 557–588. Springer US.

[Nonobe et Ibaraki, 2006] Nonobe, K. et Ibaraki, T. (2006). A metaheuristic approach to the resource constrained project scheduling with variable activity durations and convex cost functions. *In* Józefowska, J., Weglarz, J. et Hillier, F., éditeurs : *Perspectives in Modern Project Scheduling*, volume 92 de *International Series in Operations Research & Management Science*, pages 225–248. Springer US.

[Nudtasomboon et Randhawa, 1997] Nudtasomboon, N. et Randhawa, S. (1997). Resource-constrained project scheduling with renewable and non-renewable resources and time-resource tradeoffs. *Computers and Industrial Engineering*, 32(1):227–242.

[Néron, 1999] Néron, E. (1999). *Du flow-shop hybride au problème cumulatif*. Thèse de doctorat, Université de Technologie de Compiègne, Compiègne, France.

[Ozdamar *et al.*, 1998] Ozdamar, L., Ulusoy, G. et Bayyigit, M. (1998). A heuristic treatment of tardiness and net present value criteria in resource constrained project scheduling. *International Journal of Physical Distribution and Logistics Management*, 28(9-10):805–824.

[Padman et Zhu, 2006] Padman, R. et Zhu, D. (2006). Knowledge integration using problem spaces : A study in resource-constrained project scheduling. *Journal of Scheduling*, 9(2):133–152.

[Patterson *et al.*, 1989] Patterson, J., Sowinski, R., Talbot, F. et Weglarz, J. (1989). *Advances in project scheduling*, chapitre An algorithm for a general class of precedence and resource constrained scheduling problems, pages 3–28. Elsevier, Amsterdam.

[Pesch, 1999] Pesch, E. (1999). Lower bounds in different problem classes of project schedules with resource constraints. *In* Węglarz, J., éditeur : *Project Scheduling*, volume 14 de *International Series in Operations Research & Management Science*, pages 53–76. Springer US.

[Péridy et Rivreau, 2009] Péridy, L. et Rivreau, D. (2009). Local adjustments : A general algorithm. *European Journal of Operational Research*, 164(1):24–38.

[Rummel *et al.*, 2005] Rummel, J. L., Walter, Z. D., Dewan, R. M. et Seidmann, A. (2005). Activity consolidation to improve responsiveness. *European Journal of Operational Research*, 161(3):683–703.

[Russell, 1970] Russell, A. (1970). Cash flows in networks. *Management Science*, 16(5): 357–373.

[Schwindt et Trautmann, 2000] Schwindt, C. et Trautmann, N. (2000). Batch scheduling in process industries : An application of resource-constrained project scheduling. *OR Spectrum*, 22(4):501–524.

[Siarry et Michalewicz, 2007] Siarry, P. et Michalewicz, Z. (2007). *Advances in metaheuristics for hard optimization.* Springer.

[Silverston, 2011] Silverston, L. (2011). *The Data Model Resource Book : A Library of Universal Data Models for All Enterprises.* Wiley.

[Soukhal, 2012] Soukhal, A. (2012). *Modèles et Algorithmes pour l'Ordonnancement des Travaux Indépendants et Concurrents.* Rapport d'habilitation à diriger des recherches, LI Tours - Université François Rabelais, Tours, France.

[Sprecher, 1994] Sprecher, A. (1994). *Resource-constrained project scheduling : Exact methods for the multi-mode case.* Lecture Notes in Economics and Mathematical Systems. Springer-Verlag, Berlin, Germany.

[Stinson *et al.*, 1978] Stinson, J., Davis, E. et Khumawala, B. (1978). Multiple resource constrained scheduling using branch and bound. *AIEE Transactions*, 10(3):252–259.

[Talbi, 2009] Talbi, E. (2009). *Metaheuristics : From Design to Implementation.* Wiley Series on Parallel and Distributed Computing. Wiley.

[Valls *et al.*, 2005] Valls, V., Ballestín, F. et Quintanilla, S. (2005). Justification and RCPSP : A technique that pays. *European Journal of Operational Research*, 165(2):375–386.

[Valls *et al.*, 2009] Valls, V., Perez, A. et Quintanilla, S. (2009). Skilled workforce scheduling in service centres. *European Journal of Operational Research*, 193(3):791–804.

[Vanhoucke *et al.*, 2001] Vanhoucke, M., Demeulemeester, E. et Herroelen, W. (2001). On maximizing the net present value of a project under renewable resource constraints. *Management Science*, 47(8):113–1121.

[Veldhuizen et Lamont, 2000] Veldhuizen, D. V. et Lamont, G. (2000). On measuring multi-objective evolutionary algorithm performance. *In 2000 Congress of evolutionary computation*, pages 204–211, Piscataway, New jersey.

[Viana et de Sousa, 2000] Viana, A. et de Sousa, J. (2000). Using metaheuristics in multiobjective resource constrained project scheduling. *European Journal of Operational Research*, 120(2):359–374.

[Xiao *et al.*, 2013] Xiao, J., Ao, X. et Y.Tang (2013). Solving software project scheduling problems with ant colony optimization. *Computers & Operations Research*, 40(1):33–46.

[Zhou *et al.*, 2008] Zhou, D., Fu, Y., Zhong, S. et Zhao, R. (2008). The rete algorithm improvement and implementation. *In Information Management, Innovation Management and Industrial Engineering (ICIII '08)*, volume 1, pages 426–429.

[Zitzler *et al.*, 2003] Zitzler, E., Thiele, L., Laumanns, M., Fonseca, C. et da Fonseca, V. G. (2003). Performance assessment of multiobjective optimizers : an analysis and review. *IEEE Transactions on Evolutionary Computation*, 7(2):117–132.