Proposition et étude d'une extension du RCPSP pour la Mutualisation entre plusieurs sites : définition, formalisation, méthodes exactes et métaheuristiques

Arnaud Laurent

UNIVERSITÉ CLERMONT AUVERGNE

ÉCOLE DOCTORALE

SCIENCES POUR L'INGÉNIEUR DE CLERMONT-FERRAND

THÈSE

présentée par

Arnaud LAURENT

pour obtenir le grade de

Docteur d'université

Spécialité : Informatique - Productique

Proposition et étude d'une extension du RCPSP pour la Mutualisation entre plusieurs sites :

définition, formalisation, méthodes exactes et méta-heuristiques

Soutenue publiquement le 10 octobre 2017 devant le jury :

Monsieur	Laurent DEROUSSI	Co-encadrant
Madame	Maria DI MASCOLO	Rapporteur
Madame	Nathalie Grangeon	Co-encadrante
Monsieur	Aziz MOUKRIM	Examinateur
Madame	Sylvie NORRE	Directrice de thèse
Monsieur	Ameur SOUKHAL	Rapporteur

Résumé

Cette thèse a pour objet l'étude d'une extension du RCPSP, un problème d'ordonnancement de tâches et d'affectation de ressources. Cette extension considère un aspect multi-site au problème. Cette extension a pour but de modéliser des problématiques de mutualisation de ressources entre plusieurs sites, comme c'est notamment le cas dans les communautés hospitalières. Pour résoudre ce problème nous avons proposé des méthodes exactes ainsi que des méthodes approchées. Deux modèles mathématiques ont été proposés et comparés sur de petites instances générées. Des méthodes approchées ont également été proposées. Ces méthodes s'articulent sur un couplage méta-heuritique/algorithme d'ordre strict et permettent d'explorer un espace de recherche restreint. Trois différents codages de solution ont été proposés ainsi que trois algorithmes d'ordre strict correspondant à chacun des trois codages. Ces différentes méthodes ont été comparées et analysées selon les instances et le temps de calcul alloué. **Mots-clés** : RCPSP, Multi-site, ordonnancement, temps de transport, mutualisation de ressources, méta-heuristique

Table des matières

Liste des figures

Liste des tableaux

Liste des algorithmes

Introduction

Depuis plusieurs années, le monde de l'industrie est confronté à des objectifs contradictoires pour faire évoluer son organisation logistique. D'un côté, les acteurs de l'industrie doivent améliorer leur qualité de service et augmenter leur capacité de traitement des demandes. D'un autre côté, ils doivent suivre des objectifs de réduction d'émission de gaz à effet de serre, de flexibilité de la production et de réduction des dépenses, afin de pérenniser leur santé financière. Les acteurs de l'industrie ne sont plus les seuls à affronter ces problématiques. Les services publics sont aussi concernés par la course à la compétitivité, bien que n'ayant pas de but lucratif. Depuis 2004, les établissements hospitaliers ne sont plus financés par une dotation globale, mais payés à l'activité. L'objectif des services publics est toujours de proposer des services de qualité, cependant, le contexte économique fait que de plus en plus, les budgets de ces services ne suivent plus la demande croissante, notamment dans le domaine de la santé. Pour cela, l'état français tente de mettre en place depuis plusieurs années, un ensemble de mesures ayant pour but d'améliorer l'efficience du système de santé public.

En 2009, le ministère de la santé définit la Communauté Hospitalière de Territoire (CHT) comme étant « une innovation de la loi Hôpital, Patients, Santé, Territoires (HPST). Sa finalité est la recherche de la meilleure utilisation des ressources à disposition des établissements et de la complémentarité entre les acteurs ». Cette loi est ensuite reprise par le gouvernement suivant, et la CHT devient le GHT, pour Groupement Hospitalier de Territoire. Son but reste le même, celui de rendre plus efficient les groupes hospitaliers français. Le GHT est basé sur trois fondements :

- un établissement support des mutualisations, avec une personnalité morale à sa tête,
- un comité stratégique, composé des chefs d'établissement,
- un comité territorial, composé des élus locaux.

Ces GHT sont donc des regroupements d'hôpitaux dont le but est de mutualiser une partie de leurs moyens afin d'améliorer l'offre de soins au sein d'un territoire.

C'est dans ce cadre que nous nous sommes plus particulièrement intéressés à l'étude de la mutualisation des ressources entre plusieurs entités du même domaine. Cependant, notre étude se veut plus générale. La mutualisation n'est pas un défi propre au domaine de la santé ou des services publics. Le domaine de l'industrie s'intéresse lui aussi de plus en plus aux problématiques liées à la mutualisation de ressources entre plusieurs sites, chantiers ou usines.

C'est pour cela que nous avons décidé de consacrer nos travaux aux problématiques

de mutualisation de façon générale, sans rentrer dans les détails dus aux champs d'application. A cette fin, nous avons extrait les hypothèses communes à ces problèmes de mutualisation dans un contexte multi-site. Pour ce qui est du niveau de granularité de la modélisation, il en existe trois :

- le niveau stratégique : engager les entités sur des décisions à long terme, de la conception de nouveaux sites ou de leur dimensionnement
- le niveau tactique : prendre des décisions à moyen terme, du recrutement de ressources humaines ou d'achat de matériel
- le niveau opérationnel : programmer des décisions à court terme, prévoir quelles ressources sont utilisées et quelles tâches sont exécutées dans une période de temps donnée

Pour la mutualisation entre plusieurs entités distantes, ces trois niveaux sont à prendre en compte. Cependant, la principale difficulté rencontrée par les entités mutualisées est la mise en place de cette mutualisation au jour le jour. En effet, dans le cas d'entités existantes, l'organisation de la mutualisation n'est pas simple. Il faut que les entités se concertent, fassent des concessions et se mettent d'accord sur un planning commun. La difficulté vient aussi du fait que la décision peut être perçue comme injuste par une ou plusieurs entités. La solution serait donc d'avoir une solution objective, qui réponde aux exigences d'efficience du système mutualisé. C'est pour cette raison que nous nous sommes intéressés principalement au niveau opérationnel dans le cadre d'une mutualisation entre plusieurs sites distants.

Afin de traiter les problématiques les plus générales possibles, nous nous sommes intéressés à un problème théorique d'ordonnancement de projet à contraintes de ressources appelé RCPSP. Ce problème est une généralisation des problèmes d'ordonnancement où des ressources doivent exécuter des tâches. Ce problème dans sa version classique présentée dans la littérature appartient à la classe des problèmes NP-Difficile. Beaucoup de travaux traitent de ce problème et de nombreuses extensions existent prenant en compte de nombreuses caractéristiques supplémentaires. Cependant peu de travaux traitent du RCPSP dans un contexte multi-site. Ces travaux ne permettent pas de modéliser complètement une mutualisation de ressources entre plusieurs sites, où des ressources peuvent se déplacer et dans lesquels on doit déterminer sur quels sites vont être affectées les différentes tâches.

Nous proposons une extension du RCPSP classique permettant de modéliser ces situations de mutualisation entre plusieurs sites distants. Pour cela, les ressources sont séparées en deux catégories, les ressources mobiles qui peuvent se déplacer de sites en sites et les ressources fixes dont l'utilisation est limitée à leur site d'appartenance. Nous nommons ce problème RCPSP Multi-Site. L'objectif de ce problème est de modéliser les situations où un ensemble de tâches doit être effectué par des ressources mutualisées entre plusieurs sites. Le but est de déterminer sur quel site les différentes tâches seront effectuées, en optimisant l'utilisation des ressources. La proposition des méthodes de résolution pour ce problème de RCPSP Multi-Site fait également l'objet de ce manuscrit. Ces méthodes doivent pouvoir être utilisées dans un contexte d'ordonnancement journalier et doivent donc être rapides et performantes.

Ce manuscrit est constitué de cinq chapitres. Le concept de mutualisation de ressources est présenté dans le premier chapitre. Ce chapitre introduit différentes formes de mutualisation existantes dans le milieu industriel ou les services publics. Cette partie a aussi pour but de décrire le problème général de mutualisation de ressources.

Le second chapitre constitue l'état de l'art des problèmes de type RCPSP où des temps de transport sont pris en compte. Ces problèmes sont divisés en deux parties : les extensions du single-mode RCPSP et celles du multi-mode RCPSP. Ces deux problèmes se différencient par le nombre de façons d'exécuter une tâche, unique pour le Single Mode, multiple pour le Multi-Mode. Le Multi-Mode permet par exemple de considérer différents modes d'exécution pour une même tâche. Ces différentes extensions peuvent modéliser des temps de transport sous certaines contraintes. Nous montrons les intérêts mais aussi les limites de ses extensions dans leur capacité à modéliser des temps de transport dans un contexte multi-site et donc la nécessité d'introduire le problème du RCPSP Multi-Site.

Afin de modéliser le problème présenté dans le premier chapitre, nous proposons une nouvelle extension du RCPSP dans le chapitre trois. Cette extension que nous nommons RCPSP Multi-Site permet de modéliser la mutualisation de ressources entre plusieurs sites distants. Chaque tâche doit être assignée à un site pour être exécutée. Ces affectations entraînent des temps de transport de deux types :

— le transport de ressources mobiles
— le transport de produits semi-finis

Ces deux types de transport sont inhérents au contexte multi-site, où des ressources sont mutualisées. Des méthodes de résolution pour le RCPSP Multi-Site sont présentées. Deux modèles mathématiques et des méthodes approchées utilisant trois codages différents sont proposés dans ce chapitre. Les méta-heuristiques, les systèmes de voisinage et les algorithmes d'ordre strict sont présentés dans cette section.

Le chapitre quatre est consacré au problème de RCPSP Multi-Site avec contraintes. Ce problème considère des contraintes plus proches du problème des GHT comme par exemple des incompatibilités entre ressources, entre les ressources et les tâches, des indisponibilités des ressources et des contraintes de non-chevauchement entre les tâches. L'objectif de ce chapitre est de montrer comment le problème classique de RCPSP Multi-Site peut être enrichi par diverses contraintes opérationnelles et comment les méthodes que nous proposons peuvent être adaptées pour y répondre. La démarche est ici prospective, nous donnons des pistes de réflexion pour la résolution de ce problème, cependant nous ne donnons pas de résultats pour ces méthodes.

Les expérimentations sur le RCPSP Multi-Site sont présentées dans le chapitre cinq. Les deux modèles mathématiques sont comparés sur de petites instances générées. Une bibliothèque d'instances est crée en adaptant des instances de la littérature. Les méthodes approchées sont testées sur ces instances adaptées de plus grandes taille. Les différentes méta-heuristiques sont comparées pour chaque codage, puis les codages sont comparés en fonction du temps de calcul.

Enfin ce manuscrit s'achève par une conclusion et des perspectives de recherche.

Chapitre 1

Présentation de la problématique

Sommaire

1.1 Introduction

Depuis de nombreuses années, un des objectifs de l'état français est de garantir une offre de soins égale sur l'ensemble du territoire français, tout en réduisant les dépenses du système de santé. Ce principe s'applique notamment aux hôpitaux publics qui sont la cible de nombreuses réformes vouées à rétablir l'équilibre de leurs comptes. Une idée s'impose depuis 2008, la mutualisation de ressources entre les hôpitaux publics. Le principe est de créer des communautés ou groupements d'hôpitaux publics qui collaborent dans le but d'améliorer l'offre de soins au sein d'un territoire. Ces groupements communautaires sont à l'heure actuelle encore à une étape de création pour la plupart d'entre eux. De plus, ils ne disposent pas d'outils de gestion de ressources partagées, permettant notamment de proposer une gestion objective et efficiente des ressources. Cette gestion doit être objective, car si un des hôpitaux prend seul les décisions, elles doivent être perçues comme justes et équitables par les autres entités.

Nous présentons donc dans une première partie ce que la mutualisation de ressources signifie, implique au niveau des structures et présente comme avantages et désavantages. Ensuite nous nous intéressons au contexte hospitalier qui nous a initialement mené à travailler sur cette problématique de mutualisation de ressources. Ensuite, nous présentons les différentes mesures adoptées à travers les années qui permettent la mutualisation de ressources entre hôpitaux. Puis, nous présentons quelques problématiques d'autres domaines dans lesquelles la mutualisation de ressources existe : un problème de planification d'essais pneumatiques, un problème de mutualisation de moyens de production et la problématique de l'utilisation de ressources partagées dans les grilles de calcul. Nous présentons ensuite la problématique générale liée à la mutualisation de ressources dans un contexte multi-site. Enfin nous présentons l'objectif de cette thèse.

1.2 Cadre général sur la mutualisation de ressources

1.2.1 Définition

La mutualisation est définie comme "la mise en commun à des fins de partage" (Dujardin, 2006). Cette mutualisation est souvent une réponse à un besoin économique. Son but peut être de rationaliser pour économiser de l'argent ou le temps de travail, de bénéficier de compétences accrues, de meilleurs outils ou de meilleures méthodes. La mutualisation peut être entreprise au sein d'une même structure, au niveau régional, national, international. On peut parler de mutualisation pour toutes sortes de " ressources " (humaines, informationnelles, techniques ...). On distingue plusieurs cas de mutualisation (Dujardin, 2006) :

— Acquisition (achat de ressources)
— Traitement (exécution de tâches)
— Expertise (personnes, ressources)
— Produit et services (guichet d'accueil, base de données, ...)

La mutualisation n'est pas sans conséquence sur l'organisation générale d'une structure, elle entraîne forcément une réorganisation du travail, des habitudes à changer, des peurs à lever, la création d'interdépendances.

On identifie aussi 3 cadres de mutualisation (Granata, 2011) :

— **La mutualisation d'opportunité**, qui répond principalement à des logiques de court-terme. Elle peut être liée à des pics d'activité imprévus, des besoins d'acquisition ou de remplacement de ressources dans l'urgence ou contrainte pour des raisons économiques.
— **La mutualisation contractualisée**, qui répond à des logiques de court et moyen termes. Elle est liée à un besoin anticipé et découle d'une concertation entre les parties. Elle entraîne une relation contractuelle généralement entre un nombre restreint de structures.
— **La mutualisation collective**, qui répond à des logiques de moyen et long termes. Elle est liée à un important besoin d'acquisition de ressources et/ou compétences fondamentales qui restent difficiles à acquérir individuellement. Elle engage un nombre plus important de structures dans un projet collectif qui conduit obligatoirement à constituer une structure collective formelle de mutualisation. L'animation de la structure collective est nécessaire pour fédérer divers partenaires qui peuvent parfois être concurrents.

1.2.2 Intérêts et limites de la mutualisation

On peut noter les intérêts de la mutualisation suivants :

— bénéficier de compétences hautement qualifiées uniquement lorsque l'activité le nécessite
— faire face plus simplement aux variations de l'activité
— Échanger de l'expérience au sein d'un réseau
— transformer des contrats sur plusieurs structures en un seul contrat lié à une communauté, limitant la précarité de l'emploi pour les salariés
— améliorer le rendement et la compétitivité des structures

On peut cependant dénoter des inconvénients à la mutualisation. Les ressources humaines partagées doivent notamment être en général plus polyvalentes lorsqu'elles sont partagées étant donné qu'elles doivent intervenir sur plusieurs structures et équipements. Les ressources doivent pouvoir faire preuve d'adaptabilité en fonction des contextes, d'importantes capacités relationnelles et d'une bonne capacité à organiser son travail. Au niveau des structures, les communautés doivent veiller à ce que les partages ne soient pas à sens unique, favorisant une structure plutôt qu'une autre, d'autant plus dans un secteur concurrentiel.

Il existe d'autres cadres de mutualisation de ressources, comme par exemple la mutualisation logistique qui a pour but de mettre en commun une flotte de véhicules de livraison, des entrepôts, etc... Dans cette thèse, nous nous consacrerons uniquement à la mutualisation de ressources dans un contexte multi-site. Nous présentons par la suite un exemple de mutualisation qui a motivé nos travaux, les Groupements Hospitaliers de Territoire. Ces groupements, composés d'hôpitaux publics, ont pour but l'amélioration de l'offre de soins au sein d'un territoire. Ce type de groupement est l'illustration parfaite d'une mutualisation de ressources entre plusieurs entités poursuivant un même but.

1.3 Contexte hospitalier

La notion de service public hospitalier fut introduite en 1970. Son principe est de garantir à chaque citoyen dans des conditions d'égalité, quels que soient son niveau de revenu et son lieu d'habitation, l'accès à l'ensemble des biens et des services jugés fondamentaux. Depuis 2004, les établissements hospitaliers ne sont plus financés par une dotation globale, mais sont payés à l'activité. Ce mode de financement oblige les hôpitaux à adopter des stratégies habituellement réservées au monde industriel. Pour maintenir leur équilibre budgétaire, des structures de coopération inter-site ont été mises en place depuis plusieurs années. Ces mises en œuvre s'appuient sur un cadre législatif que nous présentons briévement. Il formalise la mutualisation en définissant d'une part les structures sur lesquelles cette mutualisation va s'appuyer (CHT puis GHT), puis les objectifs de ces structures par l'intermédiaire du projet médical partagé. Le dernier point abordé portera sur la présentation du problème de mutualisation des imageries et du personnel, qui s'inscrit dans le cadre de ce projet médical partagé, et sur lequel nous nous appuierons dans la suite de ce mémoire.

1.3.1 Cadre législatif

1.3.1.1 Loi HPST

La loi Hôpital Patient Santé et Territoire (HPST) fait suite à un rapport du sénateur Gérard Larcher (Larcher, 2008). Ce rapport fait plusieurs constats négatifs sur l'inadaptation aux besoins de la population et le manque d'efficience, notamment :

— Insuffisances dans l'offre de prise en charge
— Défaut de coordination dans le parcours du patient
— Insuffisances territoriales en raison de disparité entre régions et territoires.

Ce rapport propose notamment de favoriser la coopération entre différents hôpitaux sous forme de communautés et d'encourager ces collaborations par des outils d'objectivation de la prise de décision publique pour permettre des arbitrages rationnels. Ces communautés auront pour but la mutualisation de la gestion des ressources humaines, de la logistique, de la politique d'investissement ainsi que de la recherche.

De ce rapport naît en 2009 la loi HPST promulguée par Roselyne Bachelot. Cette loi a pour but de ramener les hôpitaux publics à l'équilibre budgétaire tout en améliorant l'offre de soins territoriale. Les principales mesures de cette loi sont :

— L'organisation régionale de la santé par les Agences Régionales de la Santé (ARS) qui coordonnent les hôpitaux, la médecine de ville et le secteur médico-social
— La gouvernance des hôpitaux par des conseils de surveillance, directoires, contrats pluriannuels d'objectifs et de moyens. L'objectif est de faciliter la coopération des hôpitaux publics et privés
— L'accès aux soins en repeuplant les déserts médicaux ainsi que la mise en place de sanctions pour les praticiens refusant les titulaires de la Couverture Maladie Universelle (CMU)

La loi introduit aussi la possibilité pour des hôpitaux publics de se regrouper en Communauté Hospitalière de Territoire (CHT). La CHT est une innovation de la loi HPST. Le but de la CHT est de regrouper un ensemble d'établissements publics dans le but d'aboutir à des objectifs communs. On peut citer un exemple de CHT avec celle du grand Clermont

constituée de 6 centres hospitaliers (Paul Ardier, Mont-Dore, Étienne Clémentel, Riom, Billom et le CHU de Clermont-Ferrand).

1.3.1.2 Loi Santé 2016

La loi santé 2016 a été promulguée le 26 janvier 2016 et est dans la continuité de la loi HPST. Cette loi s'articule autour de trois orientations : le renforcement de la prévention, la réorganisation autour des soins de proximité à partir du médecin généraliste et le développement des droits des patients. Prévention, proximité et soutien aux patients passent aussi par la garantie que chacun, quel que soit son revenu, pourra accéder aux professionnels de santé.

Cette loi remplace les CHT par des Groupements Hospitaliers de Territoire (GHT). D'après le vade-mecum publié par le Ministère des Affaires Sociales et de la Santé, nous pouvons résumer le rôle des Groupements Hospitaliers de Territoire ainsi : « Les Groupements Hospitaliers de Territoire sont des regroupements d'établissement qui ont pour objectif de mettre en œuvre une stratégie territoriale de prise en charge des patients afin d'assurer une offre de soins égalitaire et de qualité sur la totalité du territoire. Pour cela, les établissements partenaires sont dans l'obligation d'élaborer un projet médical partagé, qui présente les orientations stratégiques des établissements de santé partenaires. Le projet médical partagé sert de cadre pour définir le projet de soins partagé qui est la traduction, sur le plan soignant, des orientations définies dans le projet médical. Le calendrier prévoit le déploiement de ces projets sur le premier semestre 2017. » Le GHT s'articule autour d'un établissement support, chargé d'assurer certaines fonctions ou activités pour le compte des autres établissements du groupement désignés comme établissements parties au groupement. Trois formes de mutualisation sont définies :

- Les fonctions dévolues à l'établissement support de la GHT. Celui-ci assure pour le compte des établissements parties quatre fonctions : la gestion du système d'information hospitalier (avec en particulier la mise en place coordonnée d'un dossier patient), la gestion d'un département de l'information médicale, les achats et la coordination des instituts et écoles de formation.
- Les fonctions organisées en commun au sein du GHT. Elles sont au nombre de trois : l'imagerie diagnostique et interventionnelle, la biologie médicale et la pharmacie.
- Les fonctions dévolues de manière facultative à l'établissement support. L'établissement support pourra, sur la base du volontariat, gérer pour le compte des établissements parties des fonctions telles que des équipes médicales communes et la mise en place de pôles inter-établissements ou encore des activités administratives, logistiques, techniques ou médico-techniques.

Dans ce contexte, nous nous sommes intéressés à la mise en place d'une fonction d'imagerie mutualisée entre plusieurs sites, avec une équipe médico-technique et médicale elle-aussi mutualisée sur les différents sites. Cette problématique s'inscrit donc pleinement dans cette nouvelle politique de gestion. Nous allons décrire plus précisément ce problème.

1.3.2 Le problème de mutualisation de l'imagerie médicale

Un des enjeux du GHT est de mutualiser ses ressources afin d'améliorer la performance des établissements de santé et d'assurer une meilleure qualité de soins pour les patients. Un des problèmes inhérents des GHT consiste à ordonnancer les examens ou

opérations, à leur affecter les ressources humaines et matérielles nécessaires et définir dans quel hôpital ils vont être pris en charge en optimisant certains critères. Dans cette section, nous nous plaçons dans le cadre d'une convention passée entre plusieurs établissements pour améliorer le planning des examens d'imagerie médicale (IRM, scanner, ...) (Laurent et al., 2016).

1.3.2.1 Les patients et examens

Les patients sont les entités qui doivent passer les examens. Ils sont donc obligatoirement présents dans les hôpitaux pendant toute la durée de leurs examens et sont soumis à des contraintes dans le temps et l'espace. En effet un patient ne peut pas passer deux examens en même temps, et met un certain temps pour aller d'un hôpital à un autre si deux de ses examens n'ont pas lieu dans le même hôpital. Un patient est caractérisé par une liste d'examens, dans laquelle il existe un ordre, complet ou non, pour leur exécution tel que pour un couple d'examens a et b il peut exister une contrainte de précédence.

Chaque examen est défini par une durée et des quantités de types de ressources matérielles et humaines nécessaires à son exécution. Un examen est non-préemptif, il ne peut donc pas être exécuté en plusieurs fois et nécessite la présence du patient. Il ne peut donc pas s'exécuter si le patient est déjà en train de passer un examen, si le patient n'est pas dans l'hôpital où l'examen doit se dérouler ou si le patient se déplace. La durée de l'examen peut prendre en compte plusieurs sous-opérations inhérentes aux examens et dont la durée ne peut être réduite (y compris l'étape d'installation du patient et l'étape post-examen). Un examen peut avoir une date de disponibilité au plus tôt dans le cas où l'examen ne peut être réalisé avant une certaine date. Il peut aussi avoir une date de fin au plus tard, dans ce cas l'examen devra être réalisé avant cette date.

Un examen est soit programmé à l'avance, soit une urgence. Si l'examen est prévu, il est soit interne (un patient déjà hospitalisé) soit externe (un patient qui n'est pas hospitalisé). Un examen interne doit être exécuté dans l'hôpital où le patient est hospitalisé. Un examen en urgence doit être programmé quitte à perturber le planning des examens programmés.

1.3.2.2 Les ressources matérielles

Les ressources matérielles sont des ressources qui ne peuvent pas quitter leur hôpital de référence. C'est une différence majeure avec les ressources humaines du point de vue du problème. Une ressource matérielle est caractérisée par son type (IRM, Scanner) et son hôpital de référence. Une ressource matérielle à des horaires d'ouverture, soit connus, soit à déterminer en prenant en compte des contraintes telles que des maintenances à programmer ou des périodes de nettoyage. Des caractéristiques supplémentaires liées aux ressources matérielles peuvent exister telles que des incompatibilités entre les ressources et certains patients (claustrophobe/obèse et machines exiguës).

1.3.2.3 Les ressources humaines

Les ressources humaines sont soumises aux mêmes contraintes dans le temps et dans l'espace que le patient. Les ressources humaines peuvent se déplacer ou non d'hôpital en hôpital pour effectuer différentes tâches. Si une ressource peut se déplacer, elle peut

avoir une liste d'hôpitaux dans lesquels elle ne peut pas intervenir. Une ressource humaine est caractérisée par son type, par exemple un médecin ou un manipulateur. Cette ressource peut en plus posséder des compétences liées aux patients (enfant) ou liées aux ressources matérielles (IRM, radio). Lorsque les manipulateurs finissent leur formation ils sont considérés comme étant compétents sur toutes sortes de machines, puis ils se spécialisent souvent pour ne se consacrer qu'à un seul type d'examen la plupart du temps. Les ressources humaines ont un planning de travail, cela se traduit par des périodes où la ressource est disponible pour exécuter des examens et des périodes où elle ne l'est pas. Ces plannings sont soit connus et immuables, soit à déterminer en étant soumis à des règles contraignant le temps de travail (loi, contrat).

1.3.2.4 Les hôpitaux

Les hôpitaux représentent les sites où vont être effectués les examens. Tous les hôpitaux sont éloignés d'un temps de trajet connu. Cette durée va être le temps à considérer pour un déplacement de ressource ou de patient. Pour chaque hôpital on dispose d'une liste de ressources matérielles ainsi que des ressources humaines disponibles pour ce site (qui peuvent se déplacer sur le site ou qui sont employées par l'hôpital).

1.3.2.5 Exemple de GHT

Prenons une instance d'un GHT simplifié et de petite taille, avec 3 patients qui doivent passer différents examens sur deux hôpitaux éloignés d'une durée de 4 périodes. Pour les réaliser, nous disposons de 3 manipulateurs (M1, M2, M3), d'un IRM (IRM1) dans l'hôpital 1 et d'un scanner (S1) dans l'hôpital 2. Le tableau 1.1 précise les examens que doit passer chaque patient ainsi que leur durée et les ressources nécessaires. Notons que le nombre de manipulateurs requis pour un même examen peut différer d'un patient à l'autre en fonction de son état de santé ou de son âge.

	Examen 1	**Examen 2**
Patient 1	IRM - Durée : 3 2 manipulateurs	Scanner - Durée : 4 2 manipulateurs
Patient 2	Scanner - Durée : 2 1 manipulateur	IRM - Durée : 2 1 manipulateur
Patient 3	IRM - Durée : 2 1 manipulateur	

Tableau 1.1 – Exemple d'instance de GHT

Une solution de cette instance est donnée sur la figure 1.1. On peut voir que le manipulateur M2 et que les patients 1 et 2 doivent se déplacer entre les deux hôpitaux. Ces déplacements influent sur les dates de début des seconds examens que doivent passer les patients 1 et 2.

Nous venons de donner un exemple concret d'un problème de mutualisation dans lequel les ressources sont partagées entre plusieurs sites. Dans la section suivante, nous présentons d'autres contextes, a priori très éloignés du domaine hospitalier, qui font également appel à cette notion de mutualisation de ressources dans un contexte multi-site.

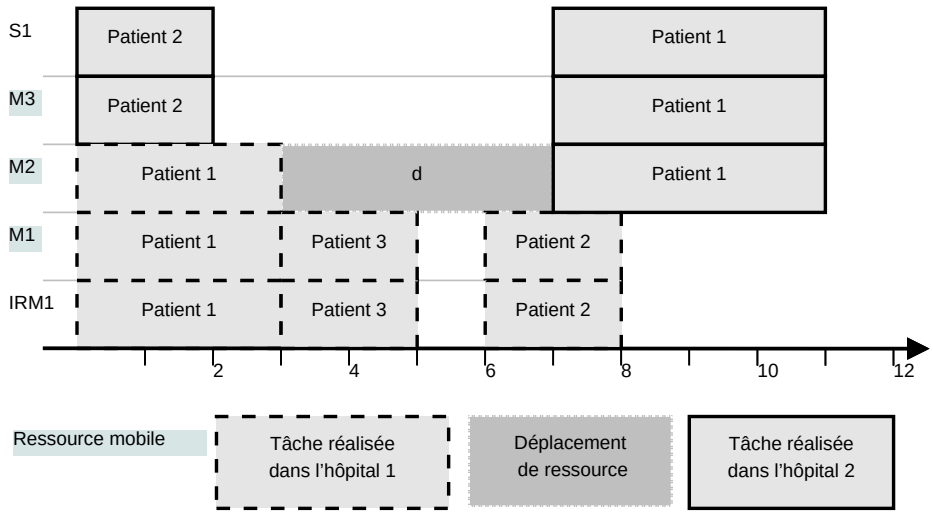

FIGURE 1.1 – Planification des examens avec l'affectation des ressources pour l'exemple de GHT

1.3.2.6 Illustration de l'intérêt de la mutualisation dans un GHT

L'aspect économique est souvent un argument majeur pour la mutualisation. Nous présentons une illustration de l'intérêt économique de la mutualisation des ressources par un petit exemple de GHT pour l'imagerie médicale.

Soit un groupement hospitalier avec 2 sites où 13 examens sont à programmer. Le site 1 contient un scanner (S1) et deux manipulateurs (M3 et M4). Le site 2 contient une radio (R1) et deux manipulateurs (M1 et M2). Les examens à programmer sont :

— 9 radios, d'une durée de 1 période
— 4 scanners, d'une durée de 2 périodes

On considère donc 13 patients devant passer 1 examen chacun. Les patients de 1 à 9 doivent passer une radio et les patients de 10 à 13 doivent passer un scanner. Un scanner et un examen radiologique nécessitent tous les deux un manipulateur et une machine pour être effectué. Les patients 2, 4, 7, 11 et 13 sont à mobilité réduite et nécessite un deuxième manipulateur pour effectuer leur examens.

Une programmation possible pour ces examens, sous forme de diagramme de Gantt, est représentée par la figure 1.2.

La programmation proposée fait en sorte de répartir la charge de travail sur les ressources de façon uniforme ainsi que de minimiser la date de fin de tous les examens (makespan). Le makespan est de 9 périodes. Sur le site 1 les ressources sont utilisées sur 8 périodes. Le scanner est utilisé pendant 8 périodes (100%) et les deux manipulateurs pendant 6 périodes (75%). Sur le site 2 les ressources sont utilisées sur 9 périodes. La radio est utilisée sur 9 périodes (100%) et les deux manipulateurs sur 6 périodes (66.6%).

Dans le cas d'une mutualisation des ressources humaines sur cet exemple de GHT, on constate que l'on peut économiser une ressource si on mutualise les manipulateurs. On considère alors seulement 3 manipulateurs (au lieu de 4). Ces manipulateurs sont mutualisés et peuvent maintenant se déplacer de site en site. La durée de déplacement d'un site à l'autre est de 1 période. La programmation des examens présentée sur la figure 1.3 optimise la répartition de la charge ainsi que le makespan.

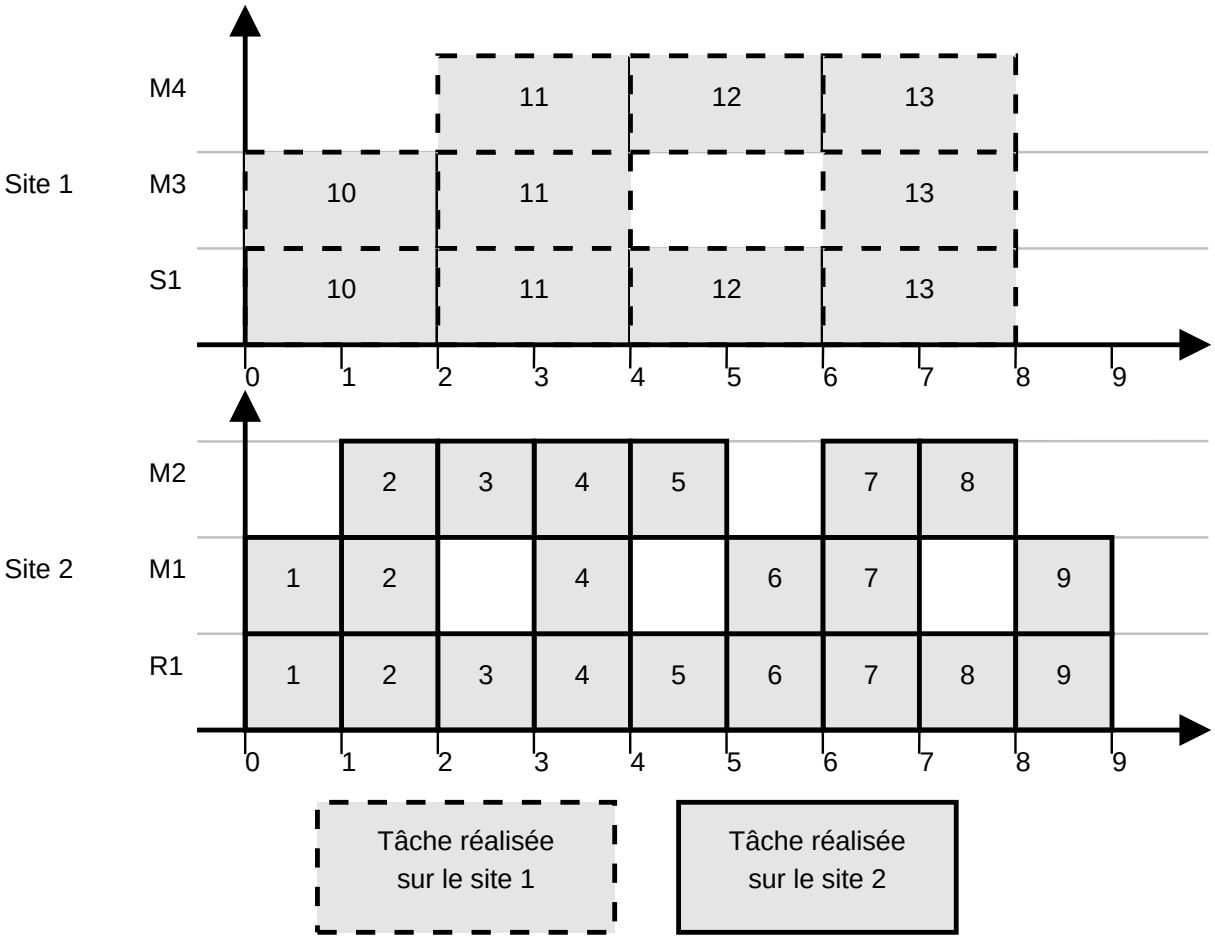

FIGURE 1.2 – Programmation des examens sans mutualisation

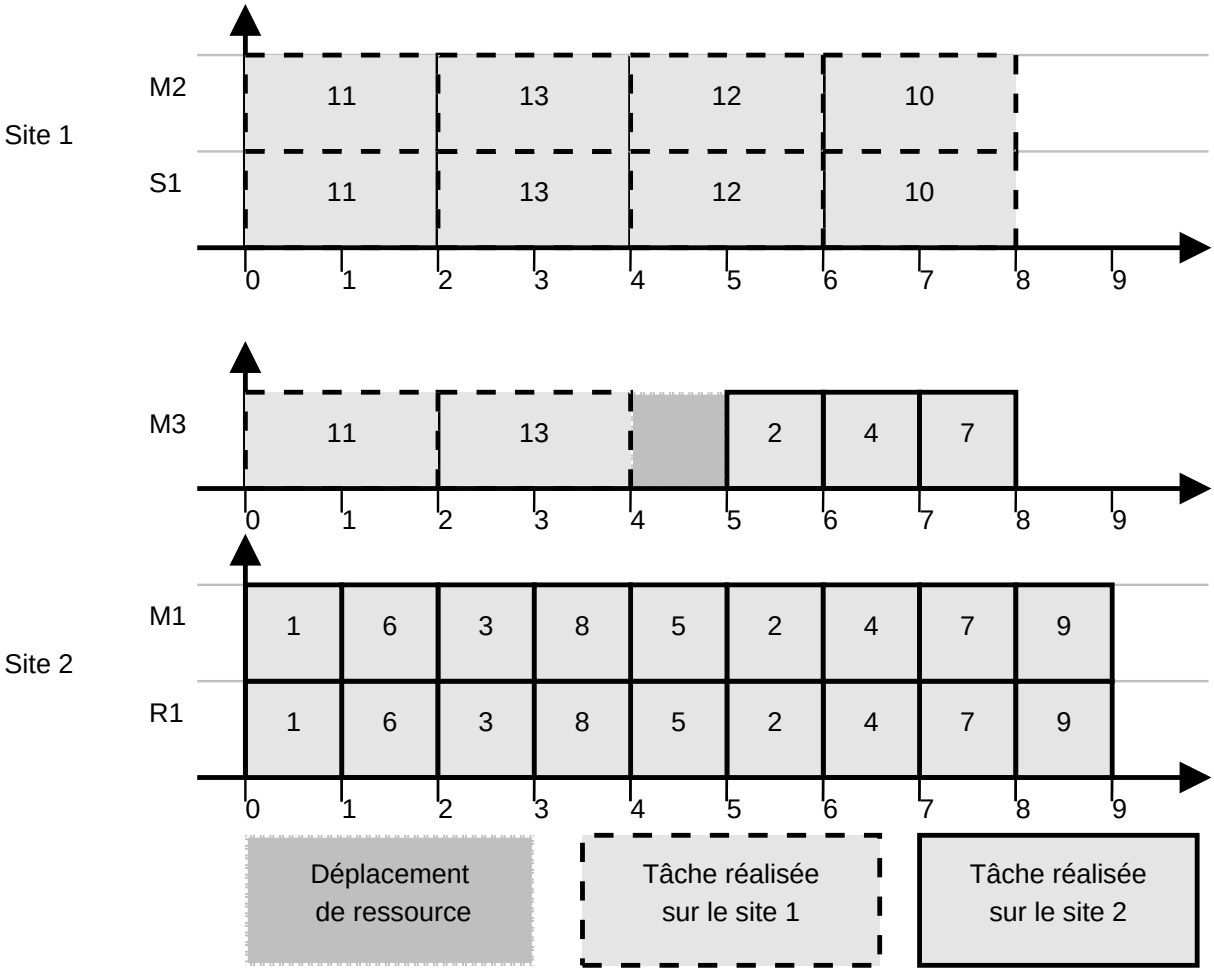

FIGURE 1.3 – Programmation des examens avec mutualisation

Dans cette solution, le manipulateur 3 effectue 2 examens de scanner sur le site 1 puis se déplace sur le site 2 pour effectuer 3 examens radiologiques. Cette solution a un makespan de 9 périodes, comme celle sans mutualisation. Au niveau du taux d'occupation, les manipulateurs 1, 2 et 3 sont occupés respectivement 8, 8 et 9 périodes (88.8%, 88.8%, 100%). On constate donc que l'on peut programmer ces examens avec seulement 3 ressources mutualisées, en gardant la même qualité de solution selon le critère du makespan.

1.4 Autres contextes

Le contexte hospitalier n'est évidement pas le seul contexte ou des coopérations entre structures amènent à mutualiser des ressources. Ces mutualisations de ressources peuvent aller du partage d'employés (d'entretien, de secrétariat ou d'aide d'éducation) entre plusieurs communes, d'un partage de livres entre plusieurs bibliothèques, à la collaboration et le partage entre plusieurs pays de scientifiques travaillant sur un même programme spatial. Un exemple concret concerne la mutualisation d'ouvriers qualifiés entre plusieurs chantiers. Le partage d'enseignants et de salles d'enseignement entre plusieurs établissements est également un exemple de mutualisation de ressources sur plusieurs sites distants. Nous présentons par la suite d'autres formes de mutualisation de ressources, dans d'autres contextes que celui du domaine médical.

1.4.1 Planification des essais pneumatiques

Un autre exemple de mutualisation de ressources entre plusieurs sites est celui de la planification des essais pneumatiques. Ce projet a été étudié au LIMOS entre 2004 et 2005 par (Cierge, 2005). La problématique est la suivante : des pilotes doivent réaliser un ensemble de tests d'enveloppes (adhérence, freinage, usure, ...) sur des circuits situés dans des régions différentes. Certains tests sont plus destructeurs que d'autres, et par conséquent les tests réalisés sur un même jeu d'enveloppes doivent être réalisés dans un ordre défini, du moins destructeur vers le plus destructeur.

Pour réaliser un test, il faut une piste (une partie d'un circuit), un pilote et un véhicule. De nombreuses contraintes sont définies :

— compatibilité pilotes / véhicules
— certification des pilotes pour réaliser certains tests
— contraintes concernant la piste (trempage, séchage, temps d'attente, ...)

Le nombre de pilotes d'essai habilités étant limité, le but est de trouver une affectation des pilotes aux essais. Pour cela on doit considérer les transports entre les circuits des ressources de mesures ainsi que des pilotes. L'entreprise pour laquelle s'est déroulée cette étude disposait de deux circuits. Le premier dans le centre de la France, le deuxième dans le sud. Une navette entre les deux circuits permettait de déplacer des ressources entre les sites. La gestion dans cette navette n'avait pas été intégrée dans l'outil d'optimisation qui avait été conçu. Les quantités de ressources disponibles sur chaque site à chaque période étaient considérées comme connues.

1.4.2 Mutualisation de moyen de production

Lors d'une conjoncture économique défavorable, les entreprises ont du mal à améliorer leurs performances de production. La baisse des investissements s'explique par la raréfaction du financement bancaire et la crainte des entrepreneurs pour l'endettement. Les conséquences sont donc doubles, puisque les industriels se retrouvent avec des outils de production sous-exploités et ont du mal à renouveler leur parc industriel. La question est donc : « peut-on trouver de nouvelles sources de financement sans emprunter ? »

L'idée est de profiter de la sous-exploitation des ressources de production en les mutualisant. Cela peut prendre la forme de locations de ressources de stockage, de machines ou bien encore d'énergie.

Ce principe a deux avantages principaux. Le premier est évidement le flux monétaire supplémentaire. Le second est d'améliorer la flexibilité de la production, en permettant à l'entreprise d'agrandir son parc industriel sans risque de perte de rentabilité. Les problèmes qui se posent, dans ce cas, sont les suivants :

— Combien de ressources doit-on considérer comme "louables" (niveau stratégique)
— Pour une période de temps donnée, combien de ressources allons-nous affecter à la location et à la production ? (niveau tactique)
— Comment planifier la production en considérant le partage de ressources avec d'autres entités ? (niveau opérationnel)

1.4.3 Grille de calcul

Une grille de calcul est un ensemble de ressources informatiques mises en commun afin de proposer une grande puissance de calcul. Son origine vient du constat initial que la plupart du temps les ressources informatiques sont sous-utilisées et de nombreux cycles de processeurs sont perdus. L'intérêt des grilles est donc de partager ces cycles non utilisés. Ce concept est aujourd'hui développé à travers le monde et constitue virtuellement certains des plus puissants ordinateurs au monde. On peut citer comme exemple d'utilisation des projets scientifiques biomédicaux (neuGrid) ou astronomiques (SETI@home).

Les ressources sont :

— Partagées : elles sont à disposition des différents utilisateurs de la grille
— Distribuées : leur localisation géographique sont différentes
— Hétérogènes : leur nature peuvent être différente, ainsi que leur système d'exploitation ou leur composition matérielle.
— Coordonnées : les ressources sont organisées par un ou plusieurs agents, centralisés ou non
— Autonomes : les ressources ne sont pas contrôlées par une entité centralisée

La mutualisation de ces ressources se fait via un middleware, un "intergiciel" qui gère les échanges entre les différents acteurs de la grille. Cet intergiciel permet la mutualisation de ressources hétérogènes (différents systèmes, différents matériels) en toute transparence pour les utilisateurs. Le problème est donc pour le middleware de distribuer les travaux à exécuter entre les différentes ressources en optimisant leur utilisation.

1.5 Le problème de mutualisation de ressources

Cette section a pour but de rassembler les caractéristiques communes aux problèmes de mutualisation entre plusieurs sites distants. Nous nous concentrons ici sur le niveau opérationnel du problème de mutualisation.

1.5.1 Le traitement de tâches

Si la mutualisation de plusieurs entités existe, c'est toujours dans le but d'améliorer le traitement d'un ensemble de tâches. Ces tâches peuvent être de plusieurs types, par exemple : des tâches de production, de transport, des cours (enseignement), des processus ou bien des examens médicaux. La mutualisation permet d'améliorer le traitement de ces tâches selon un ou plusieurs critères. Ces critères peuvent être :

— des coûts de production,
— une meilleure flexibilité,
— une meilleure robustesse aux pannes et absences de ressources,
— ...

On constate aussi dans plusieurs cas de mutualisation, des contraintes de précédence entre les tâches. Ces contraintes de précédence entre les tâches permettent de modéliser des activités découpées en plusieurs tâches. Ces contraintes de précédence peuvent représenter le transfert d'un produit semi-fini d'une tâche à l'autre. Ce produit semi-fini peut représenter un produit en cours de production, dans une phase intermédiaire. Ce produit semi-fini peut aussi représenter un patient devant passer plusieurs examens dans un GHT ou bien un processus découpé en morceaux dans des grilles de calcul.

Ce travail est dans la continuité de (Gourgand et al., 2015). Les auteurs ont montré l'intérêt du partage de ressources au niveau tactique pour ce problème d'imagerie médicale. Leur problème consiste à planifier des examens médicaux avec une granularité journalière, sans considérer l'ordonnancement des tâches. Ils considèrent un exemple avec 100 examens médicaux à planifier sur une semaine. Sans le partage de ressource, seulement 88 examens pouvait être exécutés. Quand les mêmes ressources sont partagées entre les sites, tous les examens peuvent être planifiés dans l'horizon temporel.

1.5.2 Les sites

La notion de sites distants est la principale caractéristique commune aux problématiques de mutualisation de ressources étudiées dans ce manuscrit. Les sites peuvent être des hôpitaux, des circuits, des entrepôts, des chantiers ou encore des ordinateurs dans le cas d'une grille de calcul. Ces sites sont éloignés d'une distance connue pour un couple de sites. On peut donc considérer une matrice asymétrique de distances entre tous ces sites.

1.5.3 Les ressources

Le troisième point commun entre tous ces problèmes est la mutualisation de ressources. Ces ressources peuvent prendre plusieurs formes, telles que des ressources humaines, matérielles, de stockage ou de transport. Cependant, une partie des ressources considérées pour l'ordonnancement des tâches continuera d'être associée directement à

l'entité à lesquelles elles appartiennent. C'est notamment le cas pour les machines d'imagerie médicale dans le cas des GHT, ou le cas des machines de production dans le problème de mutualisation de plusieurs usines. Ces ressources vont influencer l'affectation des tâches aux sites. Si une tâche nécessite une ressource présente uniquement sur deux sites, alors cette tâche devra être effectuée sur l'un de ces sites. On différencie alors deux classes de ressources :

— Les ressources fixes, qui restent associées à leur site initiale.
— Les ressources mobiles, qui sont vouées à se déplacer entre les sites pour effectuer des tâches. Ces ressources mobiles sont disponibles sur le site où elles réalisent leur première tâche.

On ne peut cependant pas classer les ressources en fonction de leur type dans une classe ou l'autre. Des ressources humaines peuvent par exemple être mobiles ou fixes. Cela peut notamment être le cas pour les GHT. On peut considérer un problème où les manipulateurs sont considérés comme mobiles et les autres ressources humaines (médecins, brancardiers, ...) ne quitte pas leur hôpital d'appartenance.

1.5.4 Récapitulatif des caractéristiques communes

En résumé, un problème d'ordonnancement lié à une mutualisation entre plusieurs entités distantes est caractérisé par plusieurs points :

— des tâches à exécuter,
— des contraintes de précédence entre ces tâches,
— des sites distants,
— des ressources fixes,
— des ressources mobiles.

Ces termes génériques seront utilisés dans la suite de ce mémoire. Nous présentons dans le tableau 1.2 la correspondance entre les caractéristiques présentées et les problèmes concrets. Nous utiliserons les notations suivantes :

— PE : le problème des Pilotes d'Essai
— GHT : le problème des Groupements Hospitaliers de Territoire
— MP : le problème de la Mutualisation des moyens de Production
— GC : le problème des Grilles de Calcul

1.6 Conclusion

Nous avons présenté dans cette partie différents contextes où des concepts de mutualisation de ressources existent ou sont proposés. Dans un premier temps nous avons présenté le problème initial des Groupements Hospitaliers de Territoire (GHT) qui nous a amené à considérer cette problématique. Ce concept naît de l'idée de rétablir l'équilibre des comptes des hôpitaux français dans un contexte économique de plus en plus difficile. Le principe de ces GHT est, entre autres, de mutualiser les différentes ressources de ces hôpitaux. Nous avons ensuite présenté d'autre cas de mutualisation de ressources entre différentes structures :

— le problème de planification des essais pneumatiques. Le but est de trouver une planification des essais sur des circuits distants en considérant l'affectation des pilotes,

Problème	PE	GHT	MP	GC
Tâche	essais	examens, opérations	tâches de production	processus
Précédence	transfert du pilote	transfert du patient	transfert de produits semi-finis	transfert de résultats entre processus
Sites	Circuits	hôpitaux	Usines	Ordinateurs
Distance	entre les circuits	entres les hôpitaux	entres les usines	temps de transfert dans la grille
Ressources fixes	pistes, matériel à tester	machines, une part des ressources humaines	machines, outils	processeurs, mémoire
Ressources mobiles	pilotes	ressources humaines	ressources humaines	-

TABLEAU 1.2 – Relation entre le problème général de mutualisation et les problèmes théoriques

— la mutualisation des moyens de production où l'idée est de louer des ressources industrielles sous-utilisées, notamment en temps de récession économique,
— les grilles de calculs qui sont une mise en commun de ressources informatiques dans le but de créer des super-calculateurs.

Nos travaux ont été initialement motivés par les Communautés Hospitalières de territoire (CHT) qui sont par la suite devenues des Groupements Hospitaliers de Territoire (GHT). Ces structures étant nouvelles, elles sont actuellement gérées de façon empirique et sont souvent sous-exploitées. Le but de nos travaux est de proposer des méthodes de gestion de ressources dans un contexte multi-site. Dans la suite de ce manuscrit, nous proposons un problème théorique proche du problème général de mutualisation présenté dans ce chapitre.

Parmi les problèmes d'ordonnancement rencontrés dans la littérature, le RCPSP semble être le problème qui se rapproche le plus de notre problème de mutualisation de ressources. Nous consacrons donc notre second chapitre à ce problème et à ses extensions.

Chapitre 2

État de l'art

Sommaire

2.1 Introduction

Le RCPSP (Resource Constrained Project Scheduling Problem) consiste à ordonnancer un ensemble de tâches, soumises à des contraintes de précédence et qui nécessitent des ressources dans une certaine quantité pour être exécutées, dans le but d'optimiser un ou plusieurs critères, comme par exemple la durée totale de réalisation du projet (makespan).

Plusieurs modèles du RCPSP sont répertoriés dans la littérature. Nous nous sommes intéressés principalement à deux d'entre eux : le Single-Mode RCPSP(SM-RCPSP) et le Multi-Mode RCPSP(MRCPSP). Toutes les données (durée des activités, consommation et capacité des ressources) sont supposées entières. Pour ces deux modèles, dans leur version classique, les hypothèses communes considérées sont les suivantes :

— Les ressources sont renouvelables : les ressources sont restituées à la fin de l'exécution d'une tâche et redeviennent donc disponibles pour réaliser d'autres tâches (exemple : machines, opérateurs).
— Les ressources peuvent être de plusieurs types, elles sont disponibles dans une quantité donnée sur tout l'horizon temporel.
— Les tâches sont non-préemptives, une fois commencées elles ne peuvent être interrompues.
— Les tâches sont soumises à des contraintes de précédence.
— Les tâches nécessitent la même quantité de ressources sur toute leur durée d'exécution.

La distinction entre les deux modèles se fait au niveau des modes d'exécution. Un mode d'exécution pour une tâche correspond à :

— une durée d'exécution,
— une quantité de ressources nécessaire pour chaque type de ressource.

Pour le SM-RCPSP, un seul mode est possible pour chaque tâche, on connaît donc la durée d'une tâche et ses besoins en ressources. Pour le MRCPSP, le mode d'exécution est à déterminer. Le MRCPSP peut modéliser des problèmes dont les tâches nécessitent un nombre variable de ressources, mais dont le temps d'exécution dépend du nombre de ressources affectées.

L'objet de cette section n'est pas de dresser un état de l'art exhaustif du problème de RCPSP mais plutôt de situer notre problème par rapport aux différents modèles existants et extensions du RCPSP. Le lecteur intéressé par des états de l'art sur le RCPSP pourra se reporter à (Icmeli et al., 1993) (Kolisch and Hartmann, 1999), (Yang et al., 2001), (Kolisch and Hartmann, 2006), (Hartmann and Briskorn, 2010) et (Artigues et al., 2013).

Nous présentons dans la première partie les deux modèles classiques du RCPSP qui sont le SM-RCPSP et le MRCPSP. Dans les seconde et troisième parties, nous étudierons respectivement les extensions du SM-RCPSP et du MRCPSP pouvant modéliser des temps de transport. La quatrième partie est consacrée aux extensions du RCPSP intégrant des contraintes de compétence ou d'incompatibilité sur les ressources. Enfin nous conclurons sur la revue de la littérature sur le RCPSP avec temps de transport et l'intérêt de notre extension, le RCPSP multi-site.

2.2 Les modèles du RCPSP

Nous présentons dans cette section les deux modèles du RCPSP classique, le Single-Mode RCPSP et le Multi-Mode RCPSP.

2.2.1 Single-Mode RCPSP

Le problème classique de gestion de projet à contraintes de ressources, appelé RCPSP pour Resource Constrained Project Scheduling Problem est noté $PS|prec|C_{max}$ par (Kan, 1976). Ce problème est NP-difficile, ce qui a été démontré par (Blazewicz et al., 1983). L'objectif est de réaliser un ensemble de N tâches en utilisant un ensemble de ressources renouvelables de K types différents. Les tâches $j = 1, N$ sont non-préemptives et ont une durée p_j (en nombre de périodes). Chaque tâche $j = 1, N$ possède un ensemble P_j de tâches qui doivent la précéder. Chaque tâche $j = 1, N$ nécessite une affectation de $r_{j,k}$ ressources de type $k = 1, K$. On a une quantité R_k de ressources de type $k = 1, K$ disponibles. Les tâches 1 et N sont les tâches fictives de début et fin de projet qui doivent respectivement précéder et succéder à toutes les autres tâches.

Une classification des modèles du RCPSP a été proposée par (Yang et al., 2001) que nous reprenons dans le tableau 2.1.

Afin de préciser les termes utilisés dans ce tableau nous rappelons quelques définitions générales en ordonnancement.

Définition 1. *(T'Kindt and Billaut, 2000) soit Ω l'ensemble des solutions à un problème d'ordonnancement et $C_j(x)$ la date de sortie de la tâche j selon l'ordonnancement x. Un critère Z est un* ***critère régulier*** *si et seulement si Z est une fonction croissante des dates de fin de traitement des tâches. Autrement dit : $\forall x, y \in \Omega, C_j(x) \leq C_j(y), \forall j = 1, N, \Rightarrow Z(C_1(x), ..., C_n(x)) \leq Z(C_1(y), ..., C_n(y))$.*

Définition 2. *Les contraintes de précédence sont dites* ***simples*** *si pour un couple de tâches (j, j') avec $j \in P_{j'}$ le début de la tâche j' doit être strictement supérieur à la date de fin de j sans d'autres contraintes de délais.*

Définition 3. *Un ordonnancement est* ***semi-actif*** *si aucune tâche ne peut être exécutée une période plus tôt, sans repousser la date d'exécution d'une autre tâche ou violer une contrainte (Sprecher et al., 1995).*

Définition 4. *Un ordonnancement est* ***actif*** *si aucune tâche ne peut être exécutée plus tôt, sans repousser la date d'exécution d'une autre tâche ou violer une contrainte (Sprecher et al., 1995).*

Définition 5. *Les contraintes de précédence sont dites* ***généralisées*** *si un délai minimum $\delta_{j,j'}$ est ajouté entre la date de fin de la tâche j et la date de début de la tâche j'. Ce délai pouvant être négatif, il permet de modéliser un délai minimum ou maximum entre l'exécution de deux tâches.*

Par la suite lorsque nous parlerons de RCPSP, nous ferons référence au SM-RCPSP de base.

Afin d'illustrer ce problème, nous considérons une instance du RCPSP présentée dans (Artigues et al., 2013). Le but est d'ordonnancer N=12 tâches comme indiqué sur le graphe

	SM-RCPSP classique	SM-RCPSP avec préemption	SM-RCPSP avec contraintes de précédence généralisées	SM-RCPSP général
Critère	Makespan	Makespan	Makespan	Régulier
Contraintes de précédence	Simples	Simples	Généralisées	Généralisées
Disponibilité des ressources par période	Constante	Constante	Constante ou variable	Constante ou variable
Consommation de ressources par période	Constante	Constante	Constante ou variable	Constante ou variable
Préemption	Non	Oui	Non	Non

Tableau 2.1 – Classification des modèles du SM-RCPSP (Yang et al., 2001)

de précédence de la figure 2.1. Chaque tâche est représentée par un cercle avec au-dessus sa durée et en-dessous les ressources nécessaires à son exécution sous la forme "Rtype (quantité)". On dispose de k=2 types de ressources disponibles en quantité $R1 = 7$ et $R2 = 4$. Les tâches 1 et 12 sont les deux tâches fictives de début et fin de projet.

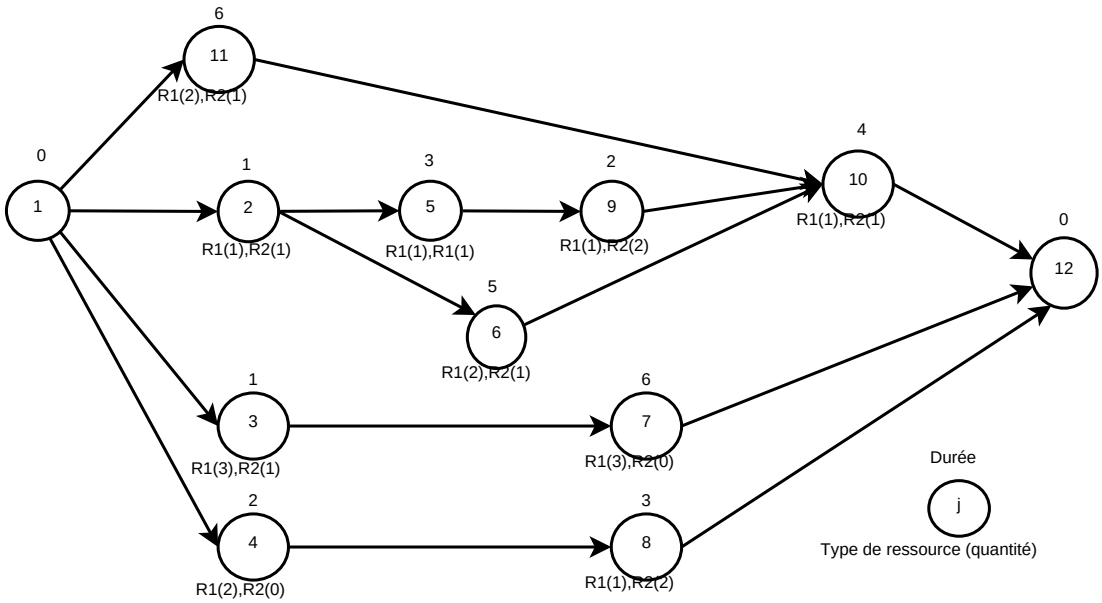

FIGURE 2.1 – Graphe de précédence

Une solution est donnée sur la figure 2.2. La solution est représentée sous forme d'un diagramme de Gantt. L'axe des abscisses représente le temps et l'axe des ordonnées représente le nombre de ressources disponibles. Cette solution est optimale avec un makespan de 12 périodes.

Une formalisation mathématique de ce problème a été proposée par (Oğuz and Bala, 1994).

— **Données**

J Ensemble de N tâches

N Nombre de tâches (tâches fictives incluses)

dd_j Date de fin au plus tard de la tâche $j \in J$ calculée à partir des contraintes de précédence

rd_j Date de fin au plus tôt de la tâche $j \in J$ calculée à partir des contraintes de précédence

P_j Ensemble de tâches qui doivent précéder $j \in J$

p_j Durée de la tâche $j \in J$

R Ensemble des ressources

K Nombre de types de ressource

T Nombre de périodes maximum

R_k Nombre de ressources de type $k = 1, .., K$ disponibles

$r_{j,k}$ Nombre de ressources de type k nécessaires pour la tâche $j \in J$;

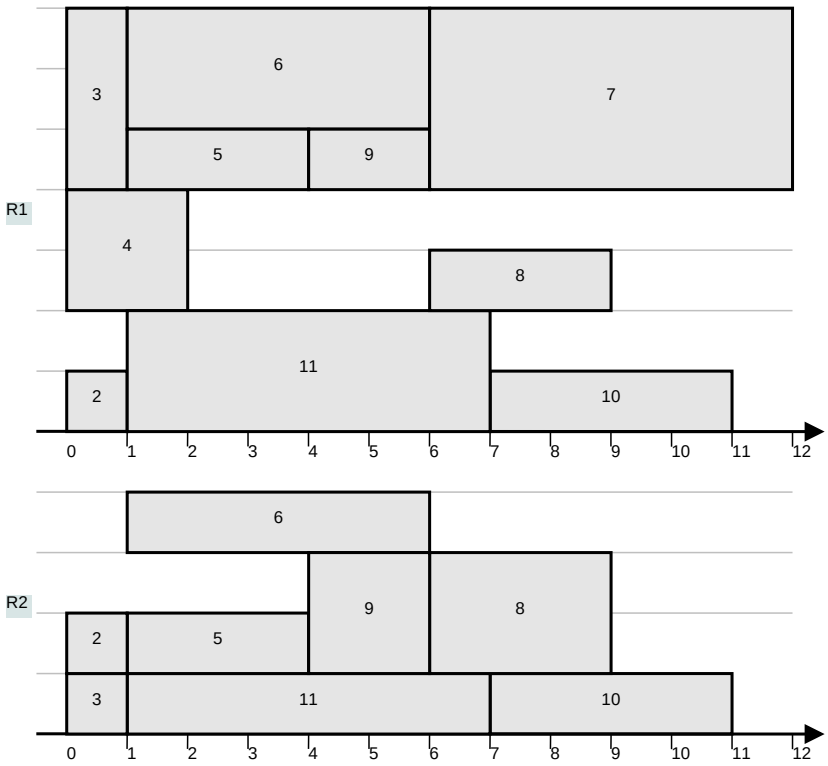

FIGURE 2.2 – Diagramme de Gantt

— **Variables**

$X_{j,t}$ =1 si la tâche $j \in J$ se termine à la période $t = 1, .., T$, 0 sinon

$$minimiser \quad \sum_{t=rd_N}^{dd_N} t * X_{N,t} \tag{2.1}$$

$$\sum_{t=rd_j}^{dd_j} X_{j,t} = 1 \quad \forall j \in J; \tag{2.2}$$

$$\sum_{t=rd_j}^{dd_j} t * X_{j,t} \geq \sum_{t=rd_{j'}}^{dd_{j'}} t * X_{j',t} + p_j \quad \forall j \in J; \quad \forall j' \in P_j; \tag{2.3}$$

$$\sum_{j=1}^{N} (r_{j,k} * \sum_{q=t}^{t+p_j-1} X_{j,q}) \leq R_k \quad \forall k \in R; \forall t = 1, .., T; \tag{2.4}$$

$$X_{j,t} \in \{0, 1\} \quad \forall j \in J; \forall t = 1, .., T; \tag{2.5}$$

Le but est de minimiser la durée du projet, c'est à dire la date de fin de la tâche fictive de fin de projet, appelée makespan (2.1). Les contraintes du problème sont : les contraintes de non préemption et de réalisation (2.2) ; les contraintes de précédence (2.3) ; les contraintes de respect des quantités de ressources disponibles (2.4) ; les contraintes de bivalence (2.5).

2.2.2 Multi-Mode RCPSP

Le Multi-Mode Resource Constrained Project Scheduling Problem permet de spécifier plusieurs modes d'exécution pour chaque tâche. Un mode correspond à une durée d'exécution et une quantité requise de ressources pour exécuter une tâche. Ce problème a été introduit par (Elmaghraby, 1977). Un exemple d'application est le problème des chantiers polyvalents (Boutevin, 2003). Le principe est d'ordonnancer des convois dans un atelier. Un convoi est un ensemble de composants. Un convoi a besoin de zones au sol modélisé par des ressources spatiales, et ces besoins en ressources spatiales dépendent de la disposition du convoi. À chaque disposition correspond donc un mode d'exécution.

Des états de l'art ont été réalisés sur ce problème par (Yang et al., 2001), (Hartmann and Briskorn, 2010).

Une classification des modèles du MRCPSP a été proposée par (Yang et al., 2001) que nous reprenons dans le tableau 2.2.

Les trade offs correspondent aux choix liés aux modes d'exécution, comme par exemple diminuer le temps d'exécution d'une tâche en augmentant le nombre de ressources qui lui sont affectées (Trade off : Temps / Ressources).

	MRCPSP avec ressources non renouvelables	MRCPSP avec ressources renouvelables	General MRCPSP
Critère	Makespan	Makespan	Régulier
Type de ressources	Non renouvelables	Renouvelables	Non renouvelables et/ou renouvelables
Contraintes de précédence	Simples	Simples	Généralisées
Disponibilité des ressources par période	Constante à l'intérieur d'un mode	Constante à l'intérieur d'un mode	Constante à l'intérieur d'un mode
Consommation de ressources par période	Constante à l'intérieur d'un mode	Constante à l'intérieur d'un mode	Constante à l'intérieur d'un mode
Préemption	Non	Non	Non
Trade Offs	Temps / Ressources	Temps / Ressources	Temps / Ressources Temps / Coût

Tableau 2.2 – Classification des modèles du MRCPSP (Yang et al., 2001)

Par la suite lorsque nous parlerons de MRCPSP nous ferons référence au MRCPSP avec ressources renouvelables.

Le modèle mathématique suivant du MRCPSP avec ressources non-renouvelables est proposé par (Kolisch and Sprecher, 1997).

— **Données**

J Ensemble de N tâches

N Nombre de tâches (tâches fictives incluses)

dd_j Date de fin au plus tard de la tâche $j \in J$ calculée à partir des contraintes de précédence

rd_j Date de fin au plus tôt de la tâche $j \in J$ calculée à partir des contraintes de précédence

p_j Durée de la tâche $j \in J$

M_j Nombre de mode d'exécution possible pour la tâche $j \in J$

$p_{j,m}$ Durée de la tâche $j \in J$ dans le mode m

R^p Ensemble des ressources renouvelables

R^v Ensemble des ressources non-renouvelables

K^p Nombre de types de ressource renouvelables

K^v Nombre de types de ressource non-renouvelables

R^p_k Nombre de ressources renouvelables de type $k = 1, .., K^p$ disponibles

R^v_k Nombre de ressources non-renouvelables de type $k = 1, .., K^v$ disponibles

$r^p_{j,k,m}$ Nombre de ressources renouvelables de type k nécessaires pour la tâche $j \in J$ dans le mode m

$r^v_{j,k,m}$ Nombre de ressources non-renouvelables de type k nécessaires pour la tâche $j \in J$ dans le mode m

— **Variables**

$X_{j,t,m}$ =1 si la tâche $j \in J$ se termine à la période $t = 1, .., T$ dans le mode m, 0 sinon

$$minimiser \quad \sum_{t=rd_N}^{dd_N} \sum_{m=1}^{M_n} t * X_{N,t,m} \tag{2.6}$$

$$\sum_{t=rd_j}^{dd_j} \sum_{m=1}^{M_j} X_{j,t,m} = 1 \quad \forall j \in J; \tag{2.7}$$

$$\sum_{t=rd_j}^{dd_j} \sum_{m=1}^{M_j} t * X_{j,t} \geq \sum_{t=rd_j}^{dd_j} \sum_{m=1}^{M_{j'}} t * X_{j',t} + p_j \quad \forall j \in J; \quad \forall j' \in P_j; \tag{2.8}$$

$$\sum_{j=1}^{N} (\sum_{q=\max(t,rd_j)}^{\min(t+p_j-1,dd_j)} \sum_{m=1}^{M_j} r^p_{j,k,m} * X_{j,q,m}) =) \leq R^p_k \quad \forall k \in R^p; \forall t = 1, .., T; \tag{2.9}$$

$$\sum_{j=1}^{N} (\sum_{q=rd_j}^{dd_j} \sum_{m=1}^{M_j} (r^v_{j,k,m} * X_{j,q,m})) \leq R^v_k \quad \forall k \in R^v; \tag{2.10}$$

$$X_{j,t,m} \in \{0,1\} \quad \forall j \in J; \forall m = 1,..,M_j; \forall t = 1,..,T; \tag{2.11}$$

Le but est de minimiser la durée du projet, appelée makespan (2.6). Les contraintes du problème sont : les contraintes de non préemption et de réalisation (2.7) ; les contraintes de précédence (2.8) ; les contraintes de respect des quantités de ressources renouvelables disponibles (2.9) et des ressources non-renouvelables disponibles (2.10) ; les contraintes de bivalence (2.11).

2.3 Des extensions du SM-RCPSP pour la prise en compte de délais inter-tâches

Dans cette partie, nous regroupons les extensions du RCPSP qui considèrent des délais minimum entre deux ou plusieurs tâches. L'intérêt de ces délais dans le cas de notre problème est qu'ils peuvent modéliser des temps de transport. Nous ferons donc une présentation de ces extensions ainsi qu'une analyse de leur capacité à répondre à notre problématique.

2.3.1 RCPSP avec temps de montage (setup time)

Cette extension du RCPSP où des temps de montage (setup time) sont ajoutés est proposée par (Kolisch, 1995). Une tâche peut nécessiter au plus une ressource de montage (setup-resource). L'ensemble de ces ressources est appelé R^s. Chacune de ces ressources est disponible en un seul exemplaire ($R^s_k| = 1 \ \forall k = 1,..,K^s$). Chaque ressource de montage ne peut exécuter qu'une tâche à la fois. Les tâches nécessitant la ressource $r \in R^s$ pour être exécutées, peuvent nécessiter cette ressource dans $u = 1,..,U_r$ différents états de montage. Si la tâche j a besoin de la ressource $r \in R^s$ dans l'état u pour être exécutée alors $k_{jru} = 1$ sinon $k_{jru} = 0$.

Si la ressource $r \in R^s$ n'est pas dans l'état u au moment d'exécuter la tâche j et que $k_{jru} = 1$, alors un temps de montage doit être pris en compte. Ce temps de montage ajoute un temps supplémentaire pour l'exécution d'une tâche. Ceci est traduit par deux modes d'exécution pour une tâche :

— Un mode $m = 1$ où le temps de montage n'est pas considéré
— Un mode $m = 2$ où le temps de montage est considéré

Un temps de montage est considéré si la ressource de montage n'est pas dans l'état nécessaire pour l'exécution de la tâche.

(Kolisch, 1995) propose une modélisation de ce problème en ajoutant aux notations classiques du RCPSP, les notations suivantes :

— **Données**

R^s Ensemble de ressources de montage

U_r Nombre d'états de montage possible pour la ressource r

K^s Nombre de type de ressources de montage s

$k_{j,r,u}$ =1 si la tâche $j \in J$ doit être exécutée par la ressource $r \in R_s$ dans l'état $u = 1,..,U_r$, 0 sinon

— **Variables**

$Y_{r,u,t}$ =1 si la ressource $r \in R_s$ est dans l'état $u = 1, .., U_r$ à la periode $t = 1, .., T$, 0 sinon

$X_{j,t,m}$ =1 si la tâche $j \in J$ débute à la période $t = 1, .., dd_j$ dans le mode $m = \{1, 2\}$, 0 sinon

$$minimiser \quad \sum_{t=rd_N}^{dd_N} \sum_{m=1}^{2} t * X_{N,t,m} \tag{2.12}$$

$$\sum_{t=rd_j}^{dd_j} \sum_{m=1}^{2} X_{j,t,m} = 1; \quad \forall j \in J; \tag{2.13}$$

$$\sum_{t=rd_j}^{dd_j} \sum_{m=1}^{2} t * X_{j,t,m} \geq \sum_{t=rd_h}^{dd_h} \sum_{m=1}^{2} t * X_{h,t,m} + p_{j,m}; \quad \forall j \in J; \quad \forall h \in P_j; \tag{2.14}$$

$$\sum_{j=1}^{N} (r_{j,k} * \sum_{q=\max(t, rd_j)}^{\min(t+p_{j,m}-1, dd_j)} \sum_{m=1}^{2} X_{j,q,m}) \leq R_k; \quad \forall k \in K \cup K^s; \forall t = 1, .., T; \tag{2.15}$$

$$k_{j,r,u} * X_{j,(t+d_{j,1}),1} \leq Y_{r,u,t}; \quad \forall j \in J; \forall u = 1, .., U_r; \forall t = 1, .., T; \forall r \in R^s \tag{2.16}$$

$$Y_{r,u,t} \leq Y_{r,u,(t-1)} + \sum_{j=1}^{N} k_{j,r,u} * X_{j,(t+d_{j,2}),2}; \quad \forall u = 1, .., U_r; \forall t = 1, .., T; \forall r \in R^s \tag{2.17}$$

$$\sum_{s=1; s \neq u}^{U_r} Y_{r,s,t} + \sum_{j=1}^{N} k_{j,r,u} * X_{j,(t+d_{j,2}),2} \leq 1; \quad \forall u = 1, .., U_r; \forall t = 1, .., T; \forall r \in R^s \tag{2.18}$$

$$X_{j,t,m} \in \{0, 1\}; \quad \forall j \in J; \forall m = 1, 2; \forall t = 1, .., T; \tag{2.19}$$

$$Y_{r,u,t} \in \{0, 1\}; \quad \forall r \in R^s; \forall u = 1, .., U_r; \forall t = 1, .., T; \tag{2.20}$$

La fonction objectif (2.12), les contraintes de non-préemption (2.13), les contraintes de précédence (2.14) et les contraintes de disponibilité des ressources (2.15) sont les mêmes que pour le MRCPSP. Les contraintes (2.16) assurent qu'une activité peut être commencée sans temps de montage uniquement si la ressource de montage est dans l'état adéquat. Autrement, la contrainte (2.17) couplée à la contrainte (2.18) provoque l'utilisation du mode (m=2) où un temps de montage est considéré. Les contraintes de bivalence sont les contraintes (2.19) et (2.20).

(Kolisch, 1995) propose aussi une résolution heuristique.

2.3.2 RCPSP avec time lag minimum

Le principe des time lag est de rajouter des délais minimaux à respecter entre la fin d'une tâche et le début d'une autre. Ces time lag minimaux peuvent servir à modéliser un temps de transfert d'une tâche à l'autre ou bien un temps d'attente obligatoire. Un délai s'applique dans le cas où une tâche j doit succéder à une autre tâche j' avec un time lag minimal $lagmin(j', j)$. (Neumann et al., 2012) et (Schwindt, 2006) généralisent les time lag en modélisant le problème sous forme de graphe. Chaque tâche $j = 1, .., N$ est associée au sommet $j' = 1, .., N$ du graphe. Chaque time lag minimum $lagmin(j', j)$ induit un arc $< j', j >$ entre les deux sommets de poids $lagmin(j', j)$

Ainsi pour une tâche i et deux tâches j, k qui doivent la précéder en utilisant la même ressource, le délai entre les tâches ne sera pas le même si l'ordre est (j, k, i) ou (k, j, i). Ces délais entre les tâches sont donc dit « **dépendant de la séquence** » ou « **setup time sequence dependent** ». Ils peuvent notamment modéliser des temps de nettoyage d'une machine, ou bien des temps d'adaptation d'une ressource humaine pour passer d'une tâche à une autre avec un temps d'apprentissage dépendant de la différence entre les deux tâches.

Pour la modélisation de ce problème, l'équation (2.21) remplace l'équation (2.3) dans le modèle du RCPSP classique :

$$\sum_{t=1}^{T} t * X_{j,t} \geq \sum_{t=1}^{T} t * X_{j',t} + p_j + lagmin(j', j); \quad \forall j \in J; \quad \forall j' \in P_j; \tag{2.21}$$

De nombreux travaux se sont intéressés à la prise en compte de ces time lag minimaux. (Chassiakos and Sakellaropoulos, 2005), (Klein and Scholl, 2000), (Vanhoucke, 2004) et (Lombardi and Milano, 2009) proposent des méthodes exactes pour résoudre ce problème. (Kolisch, 1998) propose une méthode pour déterminer une borne inférieure et (Chassiakos and Sakellaropoulos, 2005), (Klein, 2000), (Kolisch, 1998) proposent des méthodes heuristiques pour trouver une solution approchée au problème.

2.3.3 Multi project RCPSP avec setup time

Le Resource Constrained Multi-Project Scheduling Problem (RCMPSP) est une extension du RCPSP où plusieurs projets sont considérés. Un ensemble de projets M doit être exécuté avant leur date limite $DD_m; \forall m \in M$. Les projets disposent du même ensemble de ressources pour être exécutés un ensemble de tâches. Chaque projet est totalement indépendant des autres, il n'existe donc aucune contrainte de précédence entre les projets ou entre deux tâches n'appartenant pas au même projet.

La gestion des ressources se fait par projet, c'est-à-dire qu'une ressource doit forcément être affectée à un projet pour pouvoir exécuter une tâche de ce même projet. Un projet m possède donc un nombre $r_{m,k,t}$ de ressources de type k à la période t. Les ressources sont donc soit associées à un projet, soit disponibles dans le "pool" commun des ressources (projet fictif m=0) et peuvent être transférées d'un projet à un autre. Au début du problème chaque ressource est disponible dans le "pool" commun. Pour transférer des ressources de type k d'un projet m à un projet m', un coût CT_k est engendré , ainsi qu'un temps de transfert $TT_{m,m',k}$.

Pour chaque transfert on connaît le prix de ce transfert ainsi que le temps pendant lequel les ressources transférées ne seront pas disponibles. Le but est donc de connaître

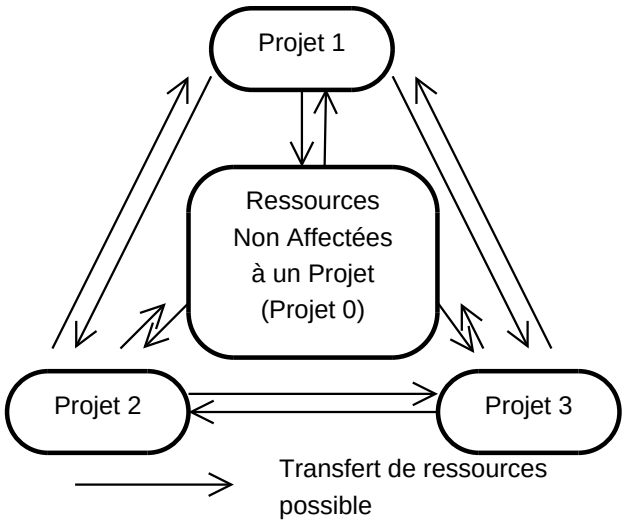

FIGURE 2.3 – Transfert de ressources possible entre les projets (Mittal and Kanda, 2009)

le début de chaque tâche pour chaque projet ainsi que de déterminer quelle quantité de ressources $g_{m,m',k,t}$ de type k doit être transférée du projet m au projet m' à la période t.

Les temps de transfert sont ainsi pris en compte par la contrainte (2.22) qui détermine les ressources disponibles pour un projet à une période donnée. On note $b_{m,k,t}$ la quantité de ressources de type k disponibles pour le projet m à la période t.

$$b_{m,k,t} = b_{m,k,t-1} - \sum_{m' \in M} g_{m,m',k,t} + \sum_{m' \in M} g_{m',m,k,t-TT_{m',m,k}};\ \forall m \in M;\ \forall k = 1,..,K;\ \forall t = 1,..,T; \tag{2.22}$$

Ce problème a été proposé par (A. Alan B. Pritsker, 1969). Depuis de nombreux travaux se sont intéressés à ce problème. Des heuristiques de construction ont été proposées par (Yang and Sum, 1993), (Krüger and Scholl, 2009), (Yang and Sum, 1997), des résolutions par méta-heuristique par (Cai and Li, 2012), (Kumar, 2014) et enfin des résolutions de programmes linéaires par (Krüger and Scholl, 2010) et (Mittal and Kanda, 2009).

2.3.4 RCPSP avec time lag conditionnels

Le RCPSP avec time lag conditionnels a été proposé par (Toussaint, 2010). C'est une extension au problème de RCPSP avec time lag minimums. Un time lag est donné entre chaque couple de tâches mais son application est conditionnelle. Le principe repose sur le fait de prendre en compte le time lag si au moins une des conditions suivantes est vérifiée :

— Les deux tâches partagent au moins une ressource,

— Il existe une contrainte de précédence entre les deux tâches.

Le but est de prendre en compte des événements qui pourraient retarder l'exécution d'une tâche. Ces retards peuvent provenir de plusieurs contraintes physiques, telles que des temps de transfert qui doivent s'appliquer entre deux tâches consécutives (qui modélisent par exemple le transport d'un produit) ou des temps d'acheminement de ressources sur le lieu d'exécution d'une tâche. L'équation (2.21) s'applique donc sous condition qu'un

des deux cas précédents soit avéré, ce qui n'a jamais été à notre connaissance modélisé mathématiquement.

Un modèle a été proposé par (Toussaint, 2010) pour résoudre le RCPSP. Ce modèle est basé sur une modélisation du RCPSP classique sous la forme de multi-flots. Le but est de déterminer les transferts de ressources entre les activités. On prend un ensemble $J = \{1, ..., N\}$ de tâches à planifier et un ensemble $R = \{1, ..., m\}$ de ressources renouvelables. Chaque type $k = 1, K$ de ressources est disponible en quantité M_k. Chaque tâche $j \in J$ a besoin d'une quantité $r_{j,k}$ de ressources de type k pour être exécutée. Une tâche j dure d_i unités de temps. Il peut exister une contrainte de précédence entre deux tâches j et j' telle que la date de fin de j doit précéder la date de début de j'. Le but est de déterminer la date de début Δ_j de chaque tâche j. On cherche à minimiser la date de fin de projet (makespan) appelée C_{max}. On note $f_{j,j',k}$ la quantité de ressources de type k transféré de la tâche j à la tâche j'.

— **Données**

J Ensemble de n tâches

R Ensemble de m ressources

E Ensemble des couples (i, j) tels que la tâche $i \in A$ précède la tâche $j \in A$

d_i Durée de la tâche $i \in A$

M_k Nombre de ressources de type $k \in R$ disponibles

$r_{i,k}$ Nombre de ressources de type $k \in R$ nécessaires pour la tâche $i \in A$

N Un grand nombre

— **Variables**

Δ_i La date de début de la tâche $i \in J$

$x_{i,j}$ La tâche $i \in J$ précède la tâche $j \in J$ dans son exécution si $x_{i,j} = 1$, 0 sinon

C_{max} La date de fin du projet

$f_{j,j',k}$ La quantité de ressources de type $k = 1, K$ transférée de la tâche $i \in J$ vers la tâche $j \in J$

On ajoute deux tâches fictives s et p à J telles que :

$$r_{s,k} = r_{p,k} = M_k; \quad \forall k = 1, K; \tag{2.23}$$

$$d_s = d_p = 0 \tag{2.24}$$

Le modèle mathématique linéaire correspondant est le suivant :

$$Minimiser \;\; C_{max} \tag{2.25}$$

$$x_{j,j'} = 1; \quad \forall (j, j') \in E; \tag{2.26}$$

$$\Delta_{j'} - \Delta_j - N * x_{j,j'} \geq d_j - N; \quad \forall j \in J \cup \{s\}; \forall j' \in J \cup \{p\}; \tag{2.27}$$

$$f_{j,j',k} - N * x_{j,j'} \leq 0; \quad \forall j \in J \cup \{s\}; \forall j' \in J \cup \{p\}; \forall k = 1, K; \tag{2.28}$$

$$\sum_{j' \in J \cup \{p\}} f_{j,j',k} = r_{j,k} \quad \forall j \in J \cup \{s\}; \forall k = 1, K; \tag{2.29}$$

$$\sum_{j \in J \cup \{s\}} f_{j,j',k} = r_{j,k} \quad \forall j' \in J \cup \{p\}; \forall k = 1, K; \tag{2.30}$$

$$C_{max} \geq \Delta_j + d_j \quad \forall j \in J; \tag{2.31}$$

$$f_{j,j',k} \in \mathbb{N}; \quad \forall j \in J \cup \{s\}; \forall j' \in J \cup \{p\}; \forall k = 1, K; \tag{2.32}$$

$$x_{j,j'} \in \{0, 1\}; \quad \forall j \in J \cup \{s\}; \forall j' \in J \cup \{p\}; \tag{2.33}$$

On cherche à minimiser le makespan (2.25). On respecte les contraintes de précédence (2.26). Les contraintes (2.27) sont celles qui vérifient la valeur de $x_{j,j'}$. Les contraintes (2.28) assurent qu'il n'existe aucun flot entre deux tâches si l'une ne précède pas l'autre. Le fait que toutes les tâches reçoivent une quantité nécessaire de ressources est assuré par les contraintes (2.29). Les contraintes (2.30) représentent la libération par toutes les tâches de toutes les ressources qui leur sont affectées. Le calcul du makespan se fait par les contraintes (2.31). Enfin les contraintes(2.32) sont les contraintes d'intégrité et les contraintes (2.33) celles de bivalence.

La prise en compte des time lag nécessite de connaître par avance le lieu où les tâches vont être exécutées dans le cas où ils représentent des temps de transfert.

— **Données**

$lag_{i,j}$ Time lag conditionnel entre la tâche $i \in J$ et la tâche $j \in J$

— **Variables**

Lag_t L'ensemble des couples d'activités (i, j) tels que i transfère une quantité de ressources non-nulle à j à l'instant t.

En se basant sur le modèle mathématique présenté précédement, un modèle a été proposé pour le RCPSP avec time-lags conditionnels par (Toussaint, 2010).

— Le calcul du makespan initial (2.31) est redéfini en prenant en compte les time lag (2.34) :

$$C_{max} = \max_{\substack{i \in J \\ (i,j) \in E}} (\Delta_i + d_i + lag_{i,j}) \tag{2.34}$$

— Les contraintes de précédence (2.26) deviennent (2.35) avec la prise en compte des time lag :

$$\Delta_i + d_i + lag_{i,j} \leq \Delta_j; \quad \forall i \in J \cup \{s, p\}; \forall j \in J \cup \{s, p\}; (i, j) \in E; \tag{2.35}$$

— On ajoute la contrainte qui oblige la somme des ressources utilisées par les activités en traitement et des ressources en cours de transfert à ne pas dépasser la quantité maximale de ressources disponibles (2.36) :

$$\sum_{\substack{i \in J \cup \{s,p\} \\ \Delta_i \leq t < \Delta_i + d_i}} r_{i,k} + \sum_{(i,j) \in Lag_t} f_{i,j,k} \leq M_k; \quad \forall k = 1, K; \forall t \in [0, C_{max}]; \tag{2.36}$$

Pour cette extension (Toussaint, 2010) utilise des méthodes de résolution de problème de flots. Le travail utilise des méthodes approchées. Les instances sont générées à partir d'une solution parfaite de 15 à 29 tâches avec un seul type de ressource. Une solution parfaite consiste à partir d'une planification des tâches sans temps mort sur un diagramme de Gantt. Les time lag et les relations de précédence sont générés de façon à ne pas modifier cette solution parfaite. Les valeurs des time lag sont inférieures ou égales à la différence de temps entre deux tâches dans la solution parfaite.
Les résultats obtenus grâce aux méthodes de résolution de problèmes de flots varient entre 0 et 13% par rapport à la valeur optimale du makespan sur 100 réplications (entre 0 et 10% sur 500 réplications).

2.4 Des extensions du MRCPSP pour la prise en compte de délais inter-tâches

Nous présentons dans cette section les extensions du MRCPSP pouvant modéliser des temps de transport. L'intérêt du Multi-Mode est la possibilité de considérer plusieurs modes d'exécution pour une tâche, et donc potentiellement plusieurs modes de transport.

2.4.1 MRCPSP avec contraintes de précédence généralisées

Le Multi-Mode Resource-Constrained Project Scheduling Problem with Generalized Precedence Relations (MRCPSP-GPR) proposé par (De Reyck and Herroelen, 1999) ajoute au problème du MRCPSP des contraintes de précédence généralisées. Les contraintes de précédence sont dites généralisées si un délai minimum $\delta_{j,j'}$ est ajouté entre la fin de la tâche j et la date de début de la tâche j'. Ce délai pouvant être négatif, il permet de modéliser un délai minimum ou maximum entre l'exécution de deux tâches. Une contrainte généralisée se modélise par la contrainte (2.37).

$$\sum_{t=1}^{T} t * X_{j,t} \geq \sum_{t=1}^{T} t * X_{j',t} + p_j + \delta_{j,j'} \quad \forall j' \in J; \quad \forall j \in P_{j'}; \tag{2.37}$$

De nombreux travaux se sont intéressés à cette problématique, notamment (Bagherinejad and Majd, 2014), (Ponz-Tienda et al., 2015), (Heilmann, 2001), (Jedrzejowicz and Skakovski, 2010), (Barrios et al., 2011), (Ballestín et al., 2013) proposent des heuristiques, (Jedrzejowicz and Ratajczak-Ropel, 2011a), (Jedrzejowicz and Ratajczak-Ropel, 2011b) proposent une résolution multi-agents, (Sabzehparvar and Seyed-Hosseini, 2008) proposent une résolution d'un modèle mathématique et (Heilmann, 2003), (Schnell and Hartl, 2016) une résolution par branch and bound.

2.4.2 MRCPSP avec setup time schedule dependent

Le Multi-Mode Resource Constrained Project Scheduling Problem avec contraintes de précedence généralisées (MRCPSP-GPR) est une extension du MRCPSP classique. À ce problème, (Mika et al., 2004) ajoutent des temps de montage, dépendant de l'ordonnancement (Multi-mode Resource Constrained Project Scheduling Problem with Schedule-dependent Setup Time MRCPSP-SST). Contrairement aux time lag, les temps de montage

dépendant de l'ordonnancement ne sont pas seulement liés aux tâches, mais aussi aux ressources utilisées. Ainsi des tâches liées par une contrainte de précédence vont entraîner un temps de montage qui va dépendre de la localisation des ressources utilisées. On parle dans ce cas de transfert de ressources nécessaires au montage (setup required resources). Bien que l'on parle de transfert de ressources, ces temps de transfert ne s'appliquent qu'entre des tâches liées par des contraintes de précédence. Aucun temps de transfert n'est appliqué à des ressources. Un exemple d'application est le cas d'une grille de calcul. On dispose d'un ensemble de nœuds pour exécuter des tâches en parallèle. Une tâche peut nécessiter des données générées par d'autres tâches pour être exécutée. Ainsi cette tâche ne peut commencer que lorsque que toutes les données nécessaires à son exécution ont été transférées sur le nœud où elle sera exécutée. Ce temps de transfert dépend ainsi de la localisation des nœuds où les tâches sont affectées. Un autre exemple pouvant être modélisé par ce problème est une usine avec plusieurs sites de production. À chaque étape, les produits semi-finis sont les ressources nécessaires au montage et entraîne des temps de transfert. Ces temps de transfert dépendrons donc des usines utilisées où seront transférés les produits semi-finis. (Mika et al., 2008) proposent une méthode de résolution de ce problème en utilisant une recherche tabou. Ils représentent leurs solutions sous forme de trois vecteurs :

- AL pour Activity List. Une liste ordonnée de tâches (dont l'ordre respecte les contraintes de précédence)
- MA pour Mode Assignment. Une liste d'affectations des modes aux tâches. Le j-ème élément du vecteur détermine le mode utilisé pour exécuter la tâche j.
- LA pour Location Assignment. Une liste d'affectations des tâches aux sites.

Pour transformer ce codage en solution, les auteurs utilisent un SGS (Schedule Generation Scheme) inspiré des travaux de (Kelley et al., 1963) en prenant en plus en compte les temps de montage dépendant de l'ordonnancement. (Mika et al., 2011) séparent la phase d'allocation des ressources aux sites en différenciant les ressources servant à exécuter les tâches et les ressources servant au transfert.

2.5 Des extensions du RCPSP pour la prise en compte des compétences des ressources

Dans cette section nous présentons les extensions du RCPSP prenant en compte les compétences des ressources.

2.5.1 Le Multi-Skilled Project scheduling problem

Ce problème est une extension du RCPSP décrit dans la partie 2.2.1, il est nommé MSPSP pour Multi-Skilled Project Scheduling Problem (Bellenguez and Néron, 2005). Contrairement au RCPSP de base, les tâches ne nécessitent pas des types de ressource mais des compétences. Chaque ressource i peut fournir plusieurs compétences KI_i. Chaque ressource est donc individualisée, puisque chaque ressource est caractérisée par un ensemble de compétences potentiellement unique. Une ressource peut être affectée aux tâches j qui nécessitent des compétences KJ_j correspondantes. Pour exécuter une tâche j on doit lui affecter un nombre $n_{j,k}$ de ressources possédant la compétence k. Chaque ressource ne

peut être affectée qu'à une seule tâche et une seule compétence sur une même période de temps. Ce problème peut être vu comme un cas particulier du MRCPSP où chaque mode représente un sous-ensemble de ressources pouvant exécuter la tâche. Nous présentons un exemple avec les tâches présentes dans le tableau 2.3 et les ressources présentes dans le tableau 2.4.

Tâche	Besoin en compétence A	Besoin en compétence B
1	2	0
2	1	1

Tableau 2.3 – Listes des tâches

Ressource	Compétence A	Compétence B
1	Oui	Non
2	Oui	Oui
3	Oui	Oui
4	Non	Oui

Tableau 2.4 – Listes des ressources

Les modes possibles pour exécuter la tâche 1 seront donc :
— Ressource 1, Ressource 2
— Ressource 1, Ressource 3
— Ressource 2, Ressource 3
pour la tâche 2 :
— Ressource 1, Ressource 2
— Ressource 1, Ressource 3
— Ressource 1, Ressource 4
— Ressource 2, Ressource 3
— Ressource 2, Ressource 4
— Ressource 3, Ressource 4

On peut donc modéliser ce problème par un MRCPSP, cependant le nombre de modes peut être très grand pour une tâche en fonction du nombre de ressources et de leurs compétences comme le montre l'exemple présenté. Le modèle du MRCPSP n'est ainsi pas forcement adapté pour un grand nombre de modes par tâche, c'est pour cela que (Bellenguez and Néron, 2005) propose le MSPSP.

Nous présentons le modèle mathématique proposé par (Correia et al., 2012a).

— **Données**

K Ensemble des compétences

R Ensemble des ressources

KJ_j Ensemble de compétences $\in K$ nécessaires pour la tâche $j \in J$;

KI_i Ensemble de compétences $\in K$ que la ressource $i \in R$ possède

$n_{j,k}$ Nombre de ressources avec la compétence $k \in K$ nécessaires à la tâche $j \in J$

M Un grand nombre

En se basant sur les notations précédentes on en déduit les ensembles suivants :

J_i Ensemble de tâches où la ressource $i \in R$ peut contribuer $J_i = \{j = 1,..,N | KI_i \cup KJ_j \neq \emptyset\}$

R_k Ensemble de ressources possédant la compétence $k \in K$ $R_k = \{i \in R; k \in KI_i\}$

IJ_j Ensemble de ressources qui peuvent être affectées à la tâche $j = 1,..,N$ $IJ_j = \{i \in R | KI_i \cup KJ_j \neq \emptyset\}$

E Ensemble de couples de tâches $j = 1,..,N$ et $j' = 1,..N$ tel qu'il n'existe pas de relation de précédence, directe ou indirecte, entre ces deux tâches

— **Variables**

X_j Date de début de la tâche $j = 1,..,N$

$\omega_{j,j'}$ La tâche $j = 1,..,N$ débute avant la tâche $j' = 1,..,N$ si $\omega_{j,j'} = 1$, 0 sinon ; $(j,j') \in E$

$Y_{i,j,k}$ La ressource $i \in R$ est affectée à la tâche $j \in J_i$ pour la compétence $k \in K$ si $Y_{i,j,k} = 1$, 0 sinon

Une formalisation de ce problème a été proposée par (Correia et al., 2012a).

$$Minimiser \quad X_N \tag{2.38}$$

$$X_l \geq X_j + p_j \quad (j,l \in J) \; avec \; ((j,l) \in E); \tag{2.39}$$

$$X_l \geq X_j + p_j - M * (1 - \omega_{j,j'}) \quad (j,j' = 1,..,N) \; avec \; ((j,j') \in E); \tag{2.40}$$

$$\omega_{j,j'} + \omega_{j',j} \leq 1; \quad (j,j' = 1,..,N) \; avec \; ((j,j') \in E); \tag{2.41}$$

$$\sum_{i \in I_k} Y_{i,j,k} = n_{j,k} \quad j = 1,..,N; \quad k \in KJ_j; \tag{2.42}$$

$$\sum_{k \in KI_i \cap KJ_j} Y_{i,j,k} \leq 1 \quad i \in R, j \in J_i; \tag{2.43}$$

$$\sum_{k \in KI_i \cap KJ_j} Y_{i,j,k} + \sum_{k \in KI_i \cap KJ'_j} Y_{i,j',k} \leq \omega_{j,j'} + \omega_{j',j} + 1;$$
$$i \in I, (j,j' = 1,..,N) \; avec \; ((j,j') \in E); \tag{2.44}$$

$$X_j \in \mathbb{N} \quad j \in J; \tag{2.45}$$
$$Y_{i,j,k} \in \{0,1\} \quad i \in I, j \in J_i, k \in KI_i \cap KJ_j; \tag{2.46}$$
$$\omega_{j,l} \in \{0;1\} \quad (j,l \in J) \; avec \; ((j,l) \in E); \tag{2.47}$$

La fonction objectif est le makespan (2.38). On respecte les contraintes de précédence (2.39). Les contraintes (2.40) interdisent à la tâche j' de débuter avant la fin de la tâche j si

$w_{j,j'} = 1$. Les contraintes (2.41) interdisent à deux tâches de se précéder mutuellement. La contrainte (2.42) assure que la quantité de ressources nécessaire pour chaque compétence soit respectée. Les contraintes (2.43) interdisent à une ressource de contribuer à plus d'une compétence pour une même tâche. La contrainte (2.44) assure le non-chevauchement des tâches qui partagent au moins une ressource. La contrainte (2.45) est la contrainte de positivité. Les contraintes (2.46) et (2.47) sont les contraintes de bivalence.

Un modèle pour résoudre ce problème est proposé par (Correia et al., 2012a), des bornes inférieures par (Bellenguez and Néron, 2005), des heuristiques et une résolution par branch and bound par (Bellenguez-Morineau, 2008). (Kazemipoor et al., 2002) s'intéressent à ce problème en ajoutant des maintenances à programmer sur des machines.

2.5.2 Le problème de gestion de projet à contraintes de ressources individualisées

Dans le modèle classique du RCPSP, les ressources ne sont pas spécifiquement affectées à une tâche. On considère la solution réalisable si une quantité de ressources suffisante est disponible pendant la durée d'exécution de la tâche. On peut être amené à individualiser les ressources si l'on souhaite traiter de cas particulier. Ainsi on peut définir quatre nouveaux types de contraintes sur les ressources (Norre, 2005) :

- **Les contraintes d'incompatibilité entre tâches et ressources :** une tâche nécessite un certain nombre de ressources par type, cependant il se peut que certaines des ressources de ce type ne soient pas compatibles avec cette tâche pour d'autres raisons.
- **Les contraintes d'incompatibilité entre ressources :** certaines ressources peuvent posséder des incompatibilités envers d'autres ressources. Si c'est le cas, chacune d'elles peut être associée à la tâche, mais pas si une autre ressource incompatible lui est associée. Ces incompatibilités peuvent aussi être vues comme des compétences. Par exemple, dans le cas de ressources humaines par rapport à des ressources matérielles.
- **Les contraintes de proximité :** des ressources peuvent posséder une liste exhaustive des ressources qui peuvent collaborer avec elles. Ce cas se présente notamment lorsque les ressources représentent des zones, où on doit utiliser des ressources contiguës pour exécuter les tâches.
- **Les contraintes de disponibilité des ressources :** les ressources étant individualisées, on peut définir leurs périodes de disponibilité. On peut donc considérer un calendrier de non-disponibilité pour chaque ressource. Si une ressource est affectée à une tâche, alors la ressource doit être disponible durant toute sa durée d'exécution.

On appellera cette nouvelle extension du RCPSP, le RCPSP-IND. Pour exécuter une tâche, on dispose de R_k ressources de type k, mais certaines de ces ressources ne peuvent pas lui être affectées car elles possèdent une incompatibilité $\lambda^2_{j,k,u}$ envers la tâche. Certaines ressources possèdent aussi des incompatibilités $\lambda^1_{k_1,u_1,k_2,u_2}$ envers d'autres ressources, et ne peuvent pas être affectées ensemble à une même tâche. Pour qu'une ressource soit affectée à une tâche, il faut que sa disponibilité $\sigma_{k,u,t}$ soit effective pendant toute la durée de la tâche.

Nous présentons le modèle mathématique proposé par (Norre, 2005).

— **Données**

J Ensemble de tâches, les deux tâches fictives sont contenues dans cet ensemble

N Nombre de tâches, $N = card(J)$

rd_j Date de fin au plus tôt de la tache $j \in J$

dd_j Date de fin au plus tard de la tâche $j \in J$

P_j Ensemble de tâches qui doivent précéder $j \in J$

p_j Durée de la tâche $j \in J$

K Nombre de types de ressources

R_k Nombre de ressources de type $k \in K$

$r_{j,k}$ Nombre de ressources de type $k \in K$ nécessaire pour la tâche $j \in J$;

T Nombre de périodes maximum

$\sigma_{k,u,t}$ $= 1$ si la ressource u de type $k \in K$ est disponible à la période $t = 1, .., T$, 0 sinon

$\lambda^1_{k_1,u_1,k_2,u_2}$ $=1$ si la ressource u_1 de type $k_1 \in K$ est compatible avec la ressource u_2 de type $k_2 \in K$, 0 sinon

$\lambda^2_{j,k,u}$ $=1$ si la ressource u de type $k \in K$ est compatible avec la tâche j, 0 sinon

— **Variables**

$X_{j,t}$ $=1$ Si la tâche $j \in J$ se finie à la période $t = 1, .., T$, 0 sinon

$\omega_{j,k,u}$ $=1$ Si la ressource u de type $k \in K$ est affectée à la tâche $j \in J$, 0 sinon

Une formalisation mathématique a été proposée par (Norre, 2005).

$$Minimiser \quad \sum_{t=rd_N}^{dd_N} t * X_{N,t} \tag{2.48}$$

$$\sum_{t=rd_j}^{dd_j} t * X_{j,t} = 1; \quad j \in J \tag{2.49}$$

$$\sum_{t=rd_h}^{dd_h} t * X_{h,t} + p_j \leq \sum_{t=rd_j}^{dd_j} t * X_{j,t} \quad j \in J; h \in P_j \tag{2.50}$$

$$\sum_{u=1}^{R_k} \omega_{j,k,u} = r_{j,k} \quad j \in J; k = 1, K \tag{2.51}$$

$$\omega_{j_1,k,u} + \omega_{j_2,k,u} + \sum_{t_1=t}^{t+p_{j_1}-1} X_{j_1,t_1} + \sum_{t_2=t}^{t+p_{j_2}-1} X_{j_2,t_2} \leq 3$$
$$(j_1, j_2 \in J) \; avec \; (j_1 < j_2); k = 1, K; u = 1, R_k; t = 1, T \tag{2.52}$$

$$\lambda^2_{j,k,u} = 0; \quad \omega_{j,k,u} = 0; j \in J; k = 1, K; u = 1, R_k \tag{2.53}$$

$\lambda^1_{k_1,u_1,k_2,u_2} = 0;$

$$\omega_{j,k_1,u_1} + \omega_{j,k_2,u_2} \leq 1$$
$$j \in J; (k_1, k_2 = 1, K) \; avec \; (k_1 \leq k_2); u_1 = 1, R_{k_1}; u_2 = 1, R_{k_2} \quad (2.54)$$

$$\sum_{t_1=t}^{t+p_j-1} t * X_{j,t_1} + \omega_{j,k,u} \leq 1$$
$$j \in J; k = 1, K; u = 1, R_k; t = 1, T \sigma_{k,u,t} = 0; \quad (2.55)$$

$$X_{j,t} \in \{0; 1\} \quad j \in J; t = 1, T \quad (2.56)$$
$$\omega_{j,k,u} \in \{0; 1\} \quad j \in J; k = 1, K; u = 1, R_k \quad (2.57)$$

Le but est de minimiser le makespan (2.48). On respecte les contraintes (2.49) de non-préemption. On respecte les contraintes (2.50) de précédence. Le respect de tous les besoins en ressources pour les tâches est assuré par la contrainte (2.51). Le respect de la non-duplication des ressources est assuré par la contrainte (2.52). Les contraintes (2.53) interdisent l'affectation d'une tâche à une ressource en cas d'incompatibilité. Les contraintes (2.54) interdisent l'affectation de deux ressources à une tâche en cas d'incompatibilité entre les ressources. Les contraintes (2.55) vérifient la disponibilité des ressources. Les contraintes (2.57) sont les contraintes de bivalence.

(Norre, 2005) propose cette extension dans le but de modéliser des problématiques industrielles avec contraintes de ressource. Pour la résolution, l'auteur propose plusieurs heuristiques adaptées de la littérature ainsi qu'une résolution par méta-heuristique.

2.6 Conclusion

Nous avons concentré cet état de l'art sur les problèmes de RCPSP pouvant modéliser des temps de transport dans la première partie, puis sur les modèles du RCPSP prenant en compte des contraintes additionnelles sur les ressources. Plusieurs extensions du RCPSP peuvent prendre en compte des temps de transport, cependant ces extensions considèrent les mêmes contraintes sur ces temps de transport :

— Les temps de transport ne dépendent que de la tâche et d'une seule ressource lui étant associée : le RCPSP avec temps de montage. Pour cette extension un temps de montage peut être considéré si une ressource de montage est utilisée par une tâche et cette ressource n'est pas dans l'état nécessité par la tâche. Dans le cas d'un temps de transport, ce temps ne peut dépendre que de l'état nécessaire pour exécuter la tâche (par exemple un lieu). Cela implique que le temps de transport ne dépende pas de l'état précédent (donc du lieu où se trouvait la tâche). Cette extension est donc essentiellement destinée à modéliser des étapes préliminaires à l'exécution d'une tâche, indépendantes de la séquence.

- Le temps de transport dépend uniquement de la séquence de tâches : RCPSP avec contraintes de précédence généralisées, MRCPSP avec contraintes de précédence et RCMPSP avec setup time. Dans ces trois extensions, le temps minimum entre l'exécution de deux tâches dépend uniquement de ces deux tâches (indirectement de leur projet pour le RCMPSP). Dans le cas d'une modélisation d'un temps de transport, cela signifie que le temps de transport est connu entre deux tâches, et que ce temps n'affecte que les produit semi-finis transmis entre les tâches. On ne peut donc pas modéliser les temps de transport de ressources entre deux tâches qui ne se transmettent pas de produits semi-finis.
- Le temps de transport dépend de la séquence des tâches et des ressources affectées aux tâches : RCPSP avec time lag conditionnels. Dans ce cas le time lag ne s'applique pas seulement entre deux tâches qui se transmettent des produits semi finis (contraintes de précédence). Les time lag s'appliquent aussi pour les tâches qui partagent une même ressource. Cependant les temps de transport doivent être connus entre tous les couples de tâches possibles, et donc connaître leur lieu d'exécution.
- Le temps de transport est déterminé en fonction du lieu d'exécution et l'utilisation des ressources est limitée par leur lieu : MRCPSP avec Schedule Setup Time. Cette extension est la seule qui permet de déterminer le lieu d'exécution d'une tâche, et donc d'obtenir des temps de transport dépendant de ce lieu. Cependant les ressources ne sont pas considérées comme mobiles. En effet chaque lieu possède une quantité de ressources disponibles pour exécuter une tâche. On ne peut donc pas modéliser de transport de ressources.

Les hypothèses considérées par les différentes extensions sur les temps de transport, ainsi que les attentes correspondant au problème de mutualisation de ressources, sont présentées dans le tableau 2.5.

Le but de l'extension que nous proposons est de pouvoir modéliser les transports de ressources et de déterminer le lieu d'exécution d'une tâche. Le chapitre suivant présente l'extension proposée. Un modèle mathématique est proposé ainsi que des méthodes de résolution approchées.

Problème	RCPSP avec temps de montage	RCPSP Time lag Min	MRCPSP setup time	RCPSP avec time lag conditionnels	MRCPSP GPR	MRCPSP setup time schedule dependant	Problème de mutualisation
Temps de transport connu pour chaque couple de tâches	non	oui	non	oui	non	non	non
Temps de transport considérés entre les tâches liées par une contrainte de précédence	oui	oui	oui	oui	oui	oui	oui
Temps de transport considérés pour les ressources mobiles	non	non	non	oui	non	non	oui
Lieu d'exécution des tâches à déterminer	non	non	non	non	non	oui	oui

Tableau 2.5 – Hypothèses sur les temps de transport

Chapitre 3

Le Resource-Constrained Project Scheduling Problem Multi-Site, formalisation et proposition de méthodes de résolution

Sommaire

3.1 Introduction

Les extensions du RCPSP pouvant prendre en compte des temps de transport sont limitées dans leur capacité à modéliser un environnement multi-site. Nous proposons donc une extension intitulée RCPSP Multi-Site. Le principe de cette extension est de considérer un ensemble de sites sur lesquels des tâches seront exécutées. Les ressources sont différenciées en deux catégories :

— les ressources fixes, qui sont assignées à un site et qui peuvent être affectées uniquement à des tâches assignées au même site qu'elles.
— les ressources mobiles, qui peuvent être assignées à n'importe quelle tâche sur n'importe quel site. Cependant l'affectation d'une ressource mobile sur deux tâches affectées à deux sites différents entraîne un temps de transport de la ressource.

L'objectif est de déterminer sur quel site chaque tâche sera exécutée et à quelle date. Pour cela on peut transférer des ressources mobiles de site en site. Des temps de déplacements vont ainsi s'appliquer dans deux cas :

— en cas de transfert de ressources entre deux sites (partage de ressources entre deux tâches),
— en cas de réalisation de deux tâches reliées par une contrainte de précédence sur deux sites différents (contrainte de précédence entre deux tâches).

Dans ce chapitre nous présentons dans un premier temps le problème du RCPSP Multi-Site en étendant le problème classique du Single-Mode RCPSP. Nous proposons ensuite deux modélisations mathématiques de ce problème : un modèle reprenant les contraintes cumulatives du RCPSP et un modèle individualisant l'affectation des ressources. Nous proposons dans la partie suivante des méthodes à base de méta-heuristiques pour résoudre ce problème. Après avoir présenté chacune des méta-heuristiques utilisées, nous proposons trois différents codages de solution. À chaque codage correspond un algorithme d'ordre strict permettant de construire une solution. Des systèmes de voisinage sont présentés afin de modifier les différentes parties des codages. Nous discuterons dans une dernière partie de l'accessibilité à une solution optimale pour chacun des codages, puis nous conclurons sur les méthodes proposées.

3.2 Proposition du modèle du RCPSP Multi-Site

Dans cette partie, nous présentons notre extension du SM-RCPSP, le RCPSP Multi-Site.

Le SM-RCPSP a pour objectif de réaliser un ensemble de N tâches en utilisant un ensemble de ressources renouvelables de K types différents. Ces ressources sont renouvelables. On dispose de T périodes pour exécuter toutes les tâches. Les tâches sont non-préemptives, une fois commencées elles ne peuvent pas être stoppées pendant toute la durée de leur exécution. Chaque tâche j de durée p_j (en nombre de périodes) possède un ensemble P_j de tâches qui doivent la précéder. De ces contraintes de précédence, on peut déduire la date d'exécution au plus tôt rd_j de chaque tâche j. Chaque tâche j nécessite une affectation de $r_{j,k}$ ressources de type k. On a une quantité R_k de ressources de type k disponibles. Les tâches 1 et N sont les tâches fictives de début et de fin de projet qui doivent respectivement précéder et succéder toutes les autres tâches.

3.2.1 Proposition du RCPSP Multi-Site

L'extension que nous proposons consiste à ajouter au SM-RCPSP un contexte multi-site avec le choix du lieu d'exécution pour les tâches. Pour cela nous introduisons la notion de sites et de ressources fixes ou mobiles. On dispose de S sites et chaque couple de sites (s, s') est séparé par un temps de déplacement $\delta_{s,s'}$. Le temps de déplacement entre deux sites n'est pas forcément symétrique. Le temps $\delta_{s,s}$ de déplacement entre un site et lui-même est de 0. Une tâche a obligatoirement besoin d'un site pour être exécutée. Une ressource r de type k est soit mobile $M_{k,r} = 1$, soit fixe $M_{k,r} = 0$. La particularité d'une ressource fixe est qu'elle peut être affectée uniquement aux tâches exécutées sur son site de référence $loc_{k,r}$. Les ressources mobiles sont sujettes à des temps de déplacement dans le cas où elles doivent exécuter successivement deux tâches sur deux sites différents. Ce temps de déplacement se traduit par un délai minimum entre la fin de la première tâche et le début de la seconde, égal au temps de déplacement entre les deux sites. Ce cas se produit lorsque deux tâches 1 et 2 de durées respectives $p1$ et $p2$ se partagent une même ressources mobile $R1$ et s'exécutent sur deux sites différents. On constate sur la figure 3.1 que l'exécution de la tâche 2 se déroule après la fin de la tâche 1 plus un délai t égal au temps de transport entre les sites 1 et 2. Le deuxième cas où ce délai s'applique est s'il existe une contrainte de précédence entre deux tâches $j = 2, .., N-1; j' \in P_j$ qui ne sont pas réalisées sur le même site. On constate sur la figure 3.2 qu'un délai s'applique, entre la fin de la tâche 1 et le début de la tâche 2, égal au temps de transport entre les sites 1 et 2. Ici ce temps de transport représente non pas le transfert d'une ressource mais le transfert d'un « produit semi-fini » transmis par la tâche 1 à la tâche 2.

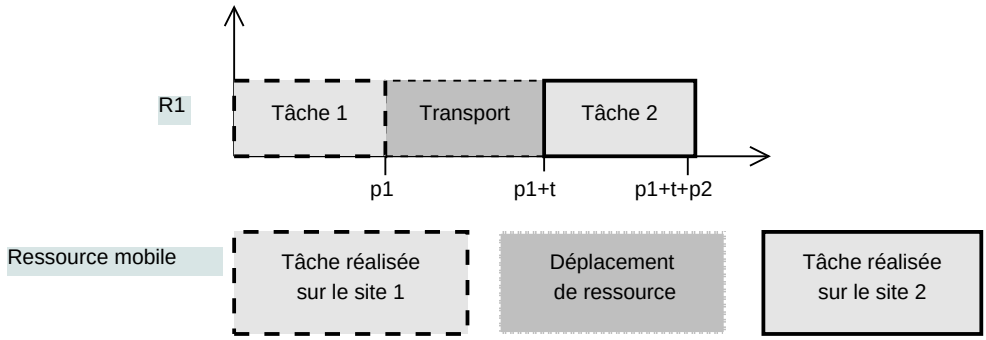

FIGURE 3.1 – Exécution de deux tâches par une ressource mobile avec temps de transport

Dans la littérature des délais similaires sont appliqués sur des instances du RCPSP. Ces délais sont appelés time lags et sont définis pour un couple de tâches données (voir section 2.3.2). Ici les time lags ne sont donc plus donnés pour deux tâches puisqu'ils dépendent des sites où vont être exécutées les tâches et ces sites sont à déterminer pour construire une solution au problème. De plus les time lags classiques de la littérature ne s'appliquent qu'en cas de contraintes de précédence. (Toussaint, 2010) introduit alors les time lags conditionnels qui s'appliquent en cas de contrainte de précédence ou de partage de ressources (voir section 2.3.4). Le problème du RCPSP avec time lags conditionnels consiste à trouver un ordonnancement des tâches et une affectation des ressources pour effectuer l'ensemble des tâches, qui minimisent un critère donné, par exemple le makespan. Pour le RCPSP Multi-Site, l'objectif est le même mais il faut en plus déterminer l'affectation des tâches aux sites, ce qui a une incidence sur les temps de transport.

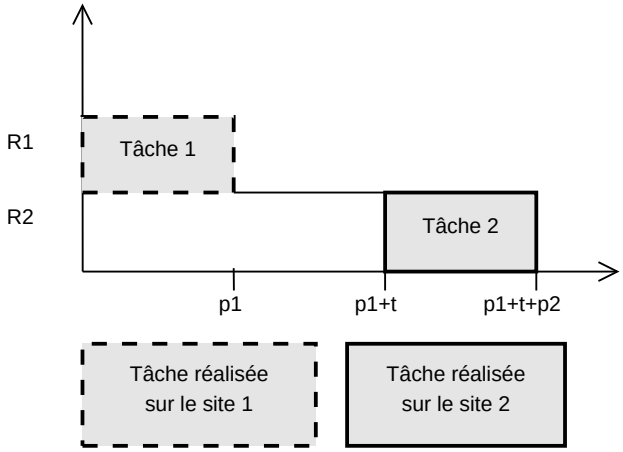

FIGURE 3.2 – Exécution de deux tâches liées par une contrainte de précédence sur deux sites différents

3.2.2 Exemple

Afin d'illustrer notre problème, nous considérons une instance du RCPSP Multi-Site comportant S=2 sites, K=4 types de ressources et N=9 tâches. Les caractéristiques des ressources sont données dans le tableau 3.1. Le temps de transport d'un site à l'autre est de deux périodes. Les 9 tâches sont reliées par des contraintes de précédence données sur la figure 3.3. Les tâches 1 et 9 sont les deux tâches fictives de début et fin de projet. Chaque tâche est représentée par un cercle avec au-dessus sa durée et en-dessous les ressources nécessaires à son exécution sous la forme "Rtype (quantité)".

	Type	Statut	Site d'appartenance
R1,1	1	Fixe	1
R1,2	1	Mobile	-
R2	2	Fixe	2
R3	3	Mobile	-
R4	4	Mobile	-

TABLEAU 3.1 – Liste des ressources disponibles

Une solution est donnée sur la figure 3.4. On peut voir que sur cette solution deux ressources mobiles R3 et R1,2 se déplacent. R3 exécute les tâches 2 et 4 sur le site 1 puis se déplace sur le site 2 pour effectuer la tâche 5. La ressource R1,2 exécute les tâches 2 et 4 sur le site 1 avant de se déplacer sur le site 2 pour réaliser la tâche 6. Ainsi des temps de déplacement s'appliquent pour chacun de ces déplacements de ressources. Par contre, R4 qui est une ressource mobile reste sur le site 2 pendant tout l'horizon temporel. Des temps de déplacement s'appliquent aussi entre les tâches 2 et 3, ainsi qu'entre les tâches 4 et 5 puisqu'elles ne sont pas effectuées sur le même site et sont reliées par des contraintes de précédence (de même entre les tâches 2 et 5 et entre les tâches 7 et 8). On obtient donc une solution avec un makespan de 12 périodes.

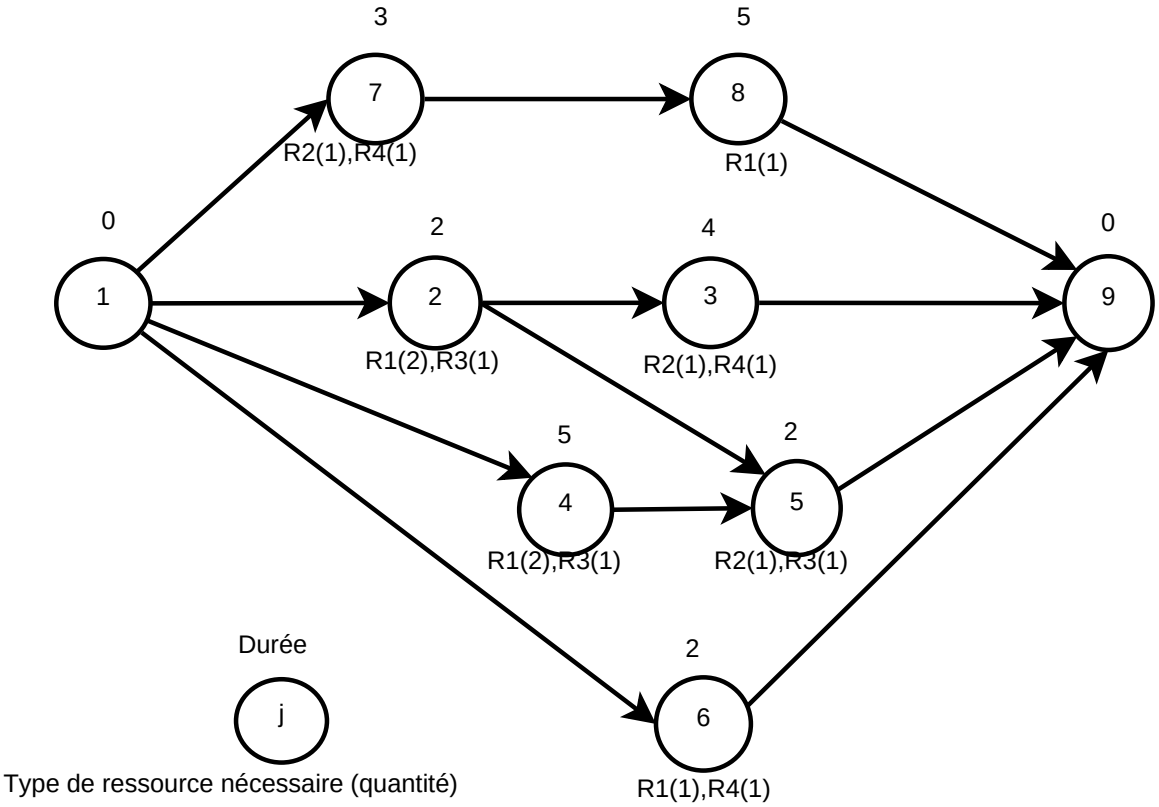

FIGURE 3.3 – Graphe de précédence du problème

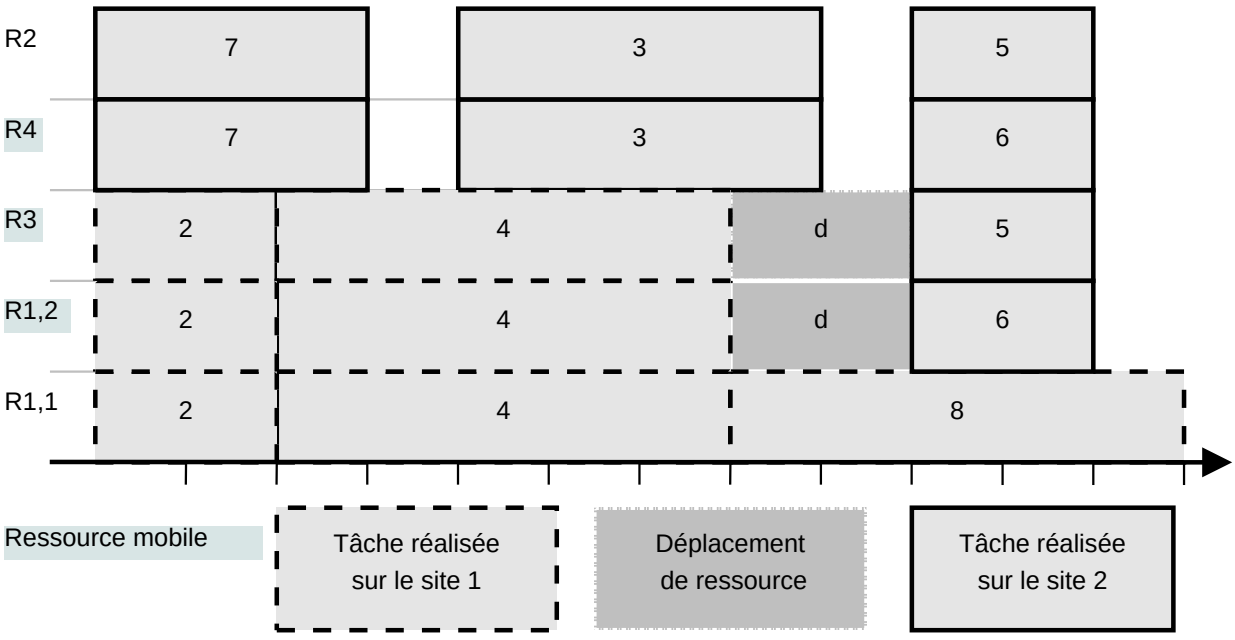

FIGURE 3.4 – Solution au problème

3.2.3 Complexité et positionnement du problème par rapport à la littérature

L'objectif de cette section est de positionner le problème du RCPSP Multi-Site par rapport aux problèmes connus de la littérature. Nous présentons les extensions pouvant être modélisées sous forme de RCPSP Multi-Site.

Le problème du RCPSP Multi-Site est NP-difficile au sens fort. Cela est prouvé par le fait que ce problème est une extension du RCPSP qui est NP-difficile au sens fort (Blazewicz et al., 1983). En effet chaque instance du RCPSP est équivalente à une instance du RCPSP Multi-Site avec un seul site et les mêmes caractéristiques pour les tâches et les ressources.

On peut également modéliser les instances du RCPSP avec time lag conditionnels par des instances du RCPSP Multi-Site. La différence des deux problèmes repose sur le fait que les time lags ne sont pas connus à l'avance pour le RCPSP Multi-Site. On peut cependant forcer la valeur d'un tel time lag en obligeant deux tâches à être exécutées sur deux sites spécifiques ayant pour distance les séparant, la valeur du time lag conditionnel entre les deux tâches (figure 3.5). Pour forcer l'affectation d'une tâche sur un site, il suffit de rajouter un besoin pour chaque tâche, d'un type de ressource utilisée uniquement par cette tâche. On fixe une ressource de ce type uniquement sur le site voulu avec une capacité de 1. La tâche ne peut donc s'exécuter que sur ce site. On généralise ce principe pour toutes les tâches. On obtient en un temps polynomial une instance du RCPSP Multi-Site avec $N-2$ sites et $K+N-2$ types de ressources, avec K le nombre de types de ressources présentes dans l'instance de RCPSP avec time lags conditionnels et N le nombre de tâches. Toutes les ressources présentes dans l'instance du RCPSP avec time lags conditionnels sont présentes et sont toutes mobiles. Résoudre cette instance du RCPSP Multi-Site revient à résoudre l'instance du RCPSP avec time lags conditionnels. Les valeurs des time lags conditionnels sont données par les distances entre les sites comme le montre la figure 3.5.

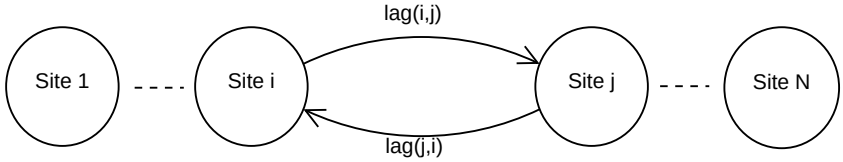

FIGURE 3.5 – Valeur des time lags conditionnels pour une instance du RCPSP Multi-Site

Le problème est donc une extension du RCPSP avec time lag conditionnels ainsi que du RCPSP avec time lag minimum et du RCPSP Single-Mode comme présenté sur les figures 3.6 et 3.7.

3.3 Modélisation mathématique du RCPSP multi-site

Nous proposons deux modèles mathématiques pour résoudre le problème du RCPSP Multi-Site. Les deux modèles se différencient par l'individualisation ou non des ressources. Le premier modèle reprend le principe des contraintes cumulatives du RCPSP qui consiste à vérifier pour chaque période de temps que le nombre de ressources nécessaires pour

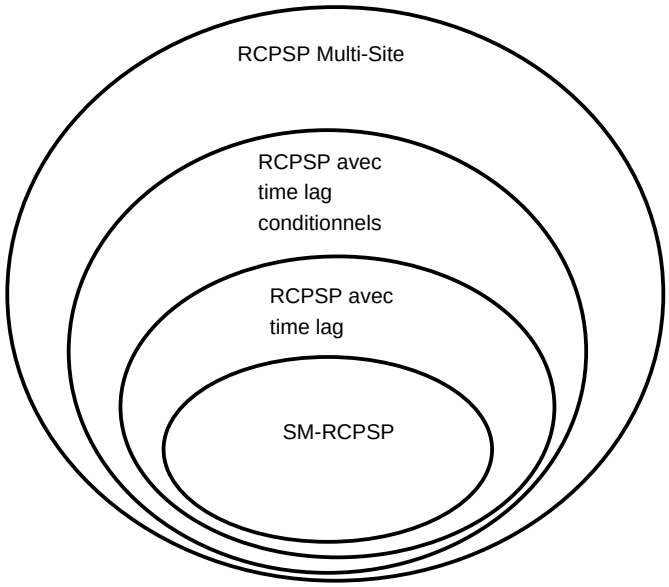

FIGURE 3.6 – Positionnement du RCPSP Multi-Site par rapport à la littérature

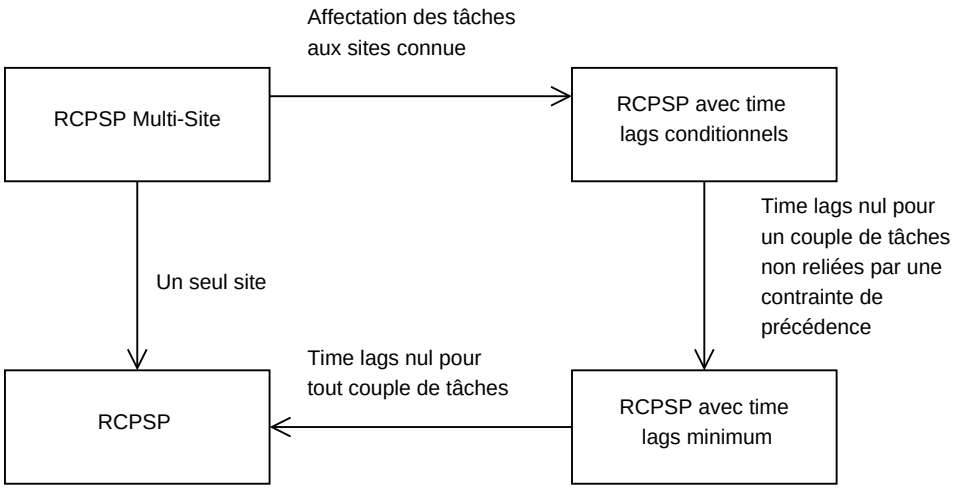

FIGURE 3.7 – Relation du RCPSP Multi-Site avec les autres extensions du RCPSP

l'exécution des tâches en cours ne dépasse pas la quantité de ressources disponibles à cette période. Le deuxième modèle individualise les ressources et gère leurs affectations une à une en précisant que deux tâches partageant une même ressource sont soumises à un délai minimum entre leur exécution, correspondant au temps de transport de la ressource.

3.3.1 Modèle mathématique non-individualisé

Pour modéliser ce problème sans individualiser les ressources on peut considérer un nombre $Rf_{k,s}$ de ressources fixes de type k disponibles sur le site s. On transfère donc une quantité $g_{s,s',k,t}$ de ressources mobiles de type k à la période t comme modélisé pour le problème de Multi Project RCPSP avec setup time (partie 2.3.3) par la contrainte 2.22. La quantité disponible $Rm_{k,s,t}$ des ressources mobiles doit être déterminée à chaque période et pour chaque site. Cette contrainte correspond aux contraintes cumulatives du RCPSP 2.4 présentées dans la section 2.2.1.

Les notations présentées sont celles fréquemment utilisées pour la modélisation de problèmes de type RCPSP et ses extensions. Elles sont complétées par de nouvelles notations propres au caractère multi-site.

— **Données**

N Nombre de tâches, la tâche 1 et N représentent les tâches fictives de début et de fin de projet.

p_j Durée de la tâche $j = 1, .., N$

P_j Ensemble de tâches qui doivent précéder $j = 1, .., N$

K Nombre de types de ressources

R_k Nombre de ressources de type $k = 1, .., K$

$Rf_{k,s}$ Nombre de ressources fixes de type $k = 1, .., K$ disponibles sur le site $s = 1, .., S$

Bm_k Nombre de ressources mobiles de type $k = 1, .., K$ disponibles

$r_{j,k}$ Nombre de ressources de type $k = 1, .., K$ nécessaires pour la tâche $j = 1, .., N$

T Nombre maximum de périodes

S Nombre de sites

$\delta_{s,s'}$ Temps de déplacement entre le site $s = 1, .., S$ et le site $s' = 1, .., S$

H Un grand nombre

— **Variables**

$X_{j,t}$ =1 si la tâche $j = 1, .., N$ se termine à la période $t = 1, .., T$, 0 sinon

$Z_{j,s}$ = 1 si la tâche $j = 1, .., N$ se déroule sur le site $s = 1, .., S$, 0 sinon

$Rm_{k,s,t}$ Nombre de ressources mobiles de type $k = 1, .., K$ disponibles sur le site $s = 1, .., S$ à la période $t = 1, .., T$

$g_{s,s',k,t}$ Nombre de ressources de type $k = 1, .., K$ transférées entre le site $s = 1, .., S$ et le site $s' = 1, .., S$ à la période $t = 1, .., T$

$XZ_{j,s,t}$ = 1 si la tâche $j = 1, .., N$ se déroule sur le site $s = 1, .., S$ à la période $t = 1, .., T$, 0 sinon

Notre problème est modélisé sous la forme du programme linéaire en nombres entiers :

$$Minimiser \quad \sum_{t=1}^{T} t * X_{N,t} \tag{3.1}$$

$$\sum_{t=1}^{T} X_{j,t} = 1; \forall j = 1, .., N; \tag{3.2}$$

$$\sum_{s=1}^{S} Rm_{k,s,1} = Bm_k; \forall k = 1, .., K; \tag{3.3}$$

$$Rm_{k,s,t} = Rm_{k,s,t-1} - \sum_{s'=1}^{S} g_{s,s',k,t} + \sum_{s'=1}^{S} g_{s',s,k,(t-\delta_{s',s})}; \quad \forall s = 1, .., S; \quad \forall k = 1, .., K; \forall t = 2, .., T; \tag{3.4}$$

$$g_{s,s',k,1} = 0; \forall k = 1, .., K; \forall s, s' = 1, .., S; \tag{3.5}$$

$$\sum_{j=1}^{N} \sum_{l=t}^{max(t+p_j-1,T)} r_{j,k} * XZ_{j,s,l} \leq Rf_{k,s} + Rm_{k,s,t}; \quad \forall s = 1, .., S; \quad \forall k = 1, .., K; \forall t = 1, .., T; \tag{3.6}$$

$$\sum_{t=1}^{T} t * X_{j,t} \geq \sum_{t=1}^{T} t * X_{h,t} + p_j + (Z_{j,s} + Z_{h,s'} - 1) * \delta(s, s'); \quad \forall h = 2, .., N; \forall j \in P_h; \forall s, s' = 1, .., S; \tag{3.7}$$

$$XZ_{j,s,t} \geq Z_{j,s} + X_{j,t} - 1; \forall j = 1, .., N; \forall s = 1, .., S; \forall t = 1, .., T; \tag{3.8}$$

$$\sum_{s=1}^{S} Z_{j,s} = 1; \forall j = 1, .., N; \tag{3.9}$$

$$X_{j,t} \in \{0; 1\}; \forall j = 1, .., N; \forall t = 1, .., T; \tag{3.10}$$

$$Z_{j,s} \in \{0; 1\}; \forall j = 1, .., N; \forall s = 1, .., S; \tag{3.11}$$

$$g_{s,s',k,t} \in [\![0; Bm_k]\!]; \forall k = 1, .., K; \forall s, s' = 1, .., S; \forall t = 1, .., T; \tag{3.12}$$

$$Rm_{k,s,t} \in [\![0; Bm_k]\!]; \forall k = 1, .., K; \forall s = 1, .., S; \forall t = 1, .., T; \tag{3.13}$$

Le modèle proposé s'appuie et étend les modèles de la littérature de (Oğuz and Bala, 1994) et (Correia et al., 2012b). Le but est de minimiser le makespan (3.1). La non préemption et la réalisation des tâches sont assurées par la contrainte (3.2).

La contrainte (3.3) garantit que l'on dispose exactement de Bm_k ressources mobiles de type k réparties sur les sites pour la première période. La contrainte (3.4) garantit la conservation des ressources, c'est-à-dire que le nombre de ressources mobiles disponibles correspond aux ressources déjà présentes sur le site, moins celles qui ont quitté le site, plus celles qui ont rejoint le site. Cette contrainte assure aussi que les ressources mobiles ne sont disponibles sur leur site d'arrivée, qu'une fois le temps de transfert appliqué. Pour que cette contrainte fonctionne, on ajoute la contrainte (3.5) qui interdit le transfert de ressources pendant la première période.

L'autre cas où des temps de transfert s'appliquent est lorsque deux tâches sont liées par une contrainte de précédence.

La contrainte (3.6) correspond à la contrainte de capacité du RCPSP. L'ensemble des tâches s'exécutant sur une même période et un même site, ne doit pas consommer plus de ressources que disponibles pour chaque type. Le nombre de ressources disponibles correspond à la somme des ressources mobiles et fixes d'un même type. La contrainte (3.8) assure que la variable $XZ_{j,s,t}$ soit égale à 1 si la tâche j s'exécute à la période t sur le site s. Chaque tâche doit être associée à un site pour être effectuée. Ceci est garantit par la contrainte (3.9). Les variables $X_{j,t}$ et $Z_{j,s}$ sont binaires (3.10) et (3.11). Les variables entières $Rm_{k,s,t}$ et $g_{s,s',k,t}$ sont positives et inférieures ou égales au nombre total de ressources mobiles (3.12) et (3.13).

3.3.2 Modèle mathématique à ressources individualisées

Dans cette formulation nous faisons le choix d'individualiser les ressources qui vont effectuer les tâches. Ce modèle pourra ensuite être étendu dans l'objectif d'intégrer des contraintes sur les ressources. Ce modèle a été publié dans (Laurent et al., 2014). Les notations utilisées pour ce modèle sont les suivantes :

— **Données**

N Nombre de tâches, les tâches 1 et N représentent les tâches fictives de début et de fin de projet.

p_j Durée de la tâche $j = 1, .., N$

P_j Ensemble de tâches qui doivent précéder $j = 1, .., N$

K Nombre de types de ressources

R_k Nombre de ressources de type $k = 1, .., K$

$M_{k,r}$ $= 1$ si la ressource $r = 1, .., R_k$ de type $k = 1, .., K$ est mobile, 0 si elle est fixe

$loc_{k,r}$ Site d'appartenance de la ressource fixe $r = 1, .., R_k$ de type $k = 1, .., K$

$r_{j,k}$ Nombre de ressources de type $k = 1, .., K$ nécessaires pour la tâche $j = 1, .., N$

T Nombre maximum de périodes

S Nombre de sites

$\delta_{s,s'}$ Temps de déplacement entre le site $s = 1, .., S$ et le site $s' = 1, .., S$

H Un grand nombre

— **Variables**

$X_{j,t}$ =1 si la tâche $j = 1, .., N$ se termine à la période $t = 1, .., T$, 0 sinon

$Y_{j,k,r}$ = 1 si la ressource $r = 1, .., R_k$ de type $k = 1, .., K$ est affectée à la tâche $j = 1, .., N$, 0 sinon

$Z_{j,s}$ = 1 si la tâche $j = 1, .., N$ se déroule sur le site $s = 1, .., S$, 0 sinon

$\omega_{j,h}$ = 1 si un temps de déplacement doit être pris en compte entre la fin de la tâche $j = 1, .., N$ et le début de la tâche $h = 1, .., N$, 0 sinon. C'est le cas lorsque ces deux tâches partagent une ressource ou lorsqu'elles sont reliées par une contrainte de précédence. Toutefois, si les tâches j et h sont exécutées sur le même site, le temps de déplacement est nul.

Notre problème est modélisé sous la forme du programme linéaire en nombres entiers suivant :

$$Minimiser \quad \sum_{t=1}^{T} t * X_{N,t} \tag{3.14}$$

$$\sum_{t=1}^{T} X_{j,t} = 1; \quad \forall j = 1, .., N; \tag{3.15}$$

$$\omega_{j,h} = 1; \omega_{h,j} = 0; \quad \forall h = 1, .., N; \forall j \in P_h; \tag{3.16}$$

$$Y_{j,k,r} + Y_{h,k,r} \leq \omega_{j,h} + \omega_{h,j} + 1; \\ (\forall j, h = 1, .., N) \; avec \; (j < h); \forall k = 1, .., K; \forall r = 1, .., R_k; \tag{3.17}$$

$$\sum_{t=1}^{T} t * X_{j,t} \geq \sum_{t=1}^{T} t * X_{h,t} + p_j + (Z_{h,s} + Z_{j,s'} - 1) * \delta(s, s') - H * (1 - \omega_{h,j}); \\ \forall j, h = 2, .., N - 1; \forall s, s' = 1, .., S; \tag{3.18}$$

$$\sum_{t=1}^{T} t * X_{j,t} \geq \sum_{t=1}^{T} t * X_{h,t} + p_h; \\ ((j = N) \; avec \; (\forall h = 2, .., N - 1)) \cup ((h = 1) \; avec \; (\forall j = 2, .., N - 1)); \tag{3.19}$$

$$\sum_{r=1}^{R_k} Y_{j,k,r} = r_{j,k}; \quad \forall j = 1, .., N; \forall k = 1, .., K; \tag{3.20}$$

$$Y_{j,k,r} \leq Z_{j,loc_{k,r}}; \quad \forall j = 1, .., N; (\forall k = 1, .., K; \forall r = 1, .., R_k;) \; avec \; (M_{k,r} = 0); \tag{3.21}$$

$$\sum_{s=1}^{S} Z_{j,s} = 1; \quad \forall j = 1, .., N; \tag{3.22}$$

$$X_{j,t} \in \{0; 1\}; \quad \forall j = 1, .., N; \forall t = 1, .., T; \tag{3.23}$$

$$Y_{j,k,r} \in \{0; 1\}; \quad \forall j = 1, .., N; \forall k = 1, .., K; \forall r = 1, .., R_k; \tag{3.24}$$

$$Z_{j,s} \in \{0; 1\}; \quad \forall j = 1, .., N; \forall s = 1, .., S; \tag{3.25}$$

$$\omega_{j,h} \in \{0; 1\}; \quad \forall j, h = 1, .., N; \tag{3.26}$$

Tout comme le modèle précédant, le modèle à ressources individualisées s'appuie sur les modèles de la littérature de (Oğuz and Bala, 1994) et (Correia et al., 2012b). Le but est de minimiser le makespan (3.14). La non préemption et la réalisation des tâches sont assurées par la contrainte (3.15).

Les contraintes (3.16), (3.17) et (3.18) permettent de modéliser qu'un temps de déplacement est à prendre en compte dans les deux cas suivants :

— Deux tâches sont reliées par une contrainte de précédence (contrainte (3.16)).
— Une même ressource est affectée à deux tâches. Dans ce cas, les deux tâches ne peuvent pas avoir lieu en même temps (contrainte (3.17)).

Dans les deux cas, les temps de déplacement sont nuls si les deux tâches sont réalisées sur le même site. La contrainte (3.18) exprime le calcul des dates de fin des tâches en intégrant les temps de déplacement qui sont fonction des sites sur lesquels les tâches sont affectées. Le temps de transfert appliqué par les contraintes (3.18) et (3.16) du problème avec ressources individualisées correspond à la contrainte (3.7) du problème avec contraintes cumulatives.

Dans cette modélisation, la prise en compte de l'affectation des ressources aux sites impose l'identification des ressources. Les contraintes de respect des quantités de ressources disponibles (données par les contraintes (2.4) dans le modèle de base du RCPSP) ne figurent donc plus en tant que telles. Elles sont assurées par les contraintes (3.17) et (3.18) dans le modèle du RCPSP Multi-Site qui indiquent qu'à un instant donné une ressource effectue au plus une action (réalisation d'une tâche ou d'un déplacement).

On note que la contrainte (3.18) modélise les délais liés aux temps de transport entre deux tâches. Ce délai s'applique en cas de contrainte de précédence (3.16) ou en cas de partage de ressources (3.17). Cette modélisation peut donc s'adapter très facilement au cas des time lag conditionnels dans lequel la notion de sites n'apparaît pas. Il convient alors de remplacer le terme $(Z_{j,s} + Z_{h,s'} - 1) * \delta(s, s')$ par la valeur du time lag entre les tâches j et h comme présenté dans l'équation (3.27).

$$\sum_{t=1}^{T} t * X_{j,t} \geq \sum_{t=1}^{T} t * X_{h,t} + p_j + lag(j, h) - H * (1 - \omega_{h,j}) \quad j, h = 2, .., N - 1; s, s' = 1, .., S; \tag{3.27}$$

Le calcul des dates de fin de tâche est assuré par la contrainte (3.19). La contrainte (3.20) assure que les quantités nécessaires de ressources sont affectées. Chaque tâche doit

être exécutée sur le site d'appartenance des ressources fixes affectées (si des ressources fixes sont affectées) (3.21). Chaque tâche est exécutée sur un seul site (3.22). Les contraintes de bivalence sur les variables sont les contraintes (3.23)(3.24)(3.25) et (3.26).

3.4 Proposition de méthodes approchées

Ce problème faisant partie des problèmes NP-difficiles au sens fort (voir partie 3.2.3), des méthodes de résolution approchées semblent être plus adaptées pour des instances de taille moyenne et grande. Nous proposons de concevoir des méta-heuristiques basées individu. Nous présentons donc dans un premier temps les méta-heuristiques utilisées. Nous présentons ensuite les différents codages utilisés pour représenter une solution. Pour chacun de ces codages, nous proposons un algorithme d'ordre strict qui construit une solution. Puis nous proposons des systèmes de voisinage composés de un à trois mouvements. Enfin nous discutons de l'accessibilité des codages à une solution optimale.

3.4.1 Méta-heuristiques

Les méta-heuristiques sont des méthodes génériques pour résoudre des problèmes sans considérer leur nature. L'objectif de ces méthodes est de modifier des solutions au travers de systèmes de voisinage, afin de progresser vers un minimum global. Pour résoudre notre problème nous nous basons sur des méta-heuristiques basée individu, c'est à dire qui ne considèrent qu'une seule solution courante au cours du déroulement de l'algorithme.

3.4.1.1 La recherche locale

Le principe de la recherche locale (algorithme 1) est d'explorer le voisinage d'une solution et d'accepter une solution voisine si elle est de meilleure ou d'égale qualité. On réitère un certain nombre de fois (en fonction du test d'arrêt) l'opération pour obtenir un minimum local. Dans nos algorithmes, nous utilisons une recherche locale stochastique, qui choisit à chaque itération un seul voisin aléatoirement jusqu'à ce qu'on estime avoir atteint un minimum local. On ne peut pas être sûr d'être dans un minimum local, pour cela on détermine un critère d'arrêt qui donne une certaine probabilité d'être dans un minimum local. Un exemple de critère d'arrêt est de stopper la recherche locale lorsque que l'on a parcouru un certain nombre de solutions sans trouver de solution améliorante.

3.4.1.2 Le recuit simulé inhomogène

Le principe du recuit simulé (algorithme 2) proposé par (Kirkpatrick et al., 1983) permet d'éviter le problème majeur d'une recherche locale, qui est de converger vers un minimum local. Le recuit simulé permet d'accepter des transitions dégradantes avec une certaine probabilité. La probabilité $p(X', i)$ d'accepter une solution X' voisine d'une solution courante X, va dépendre d'une valeur décroissante appelée température T_i, avec i l'itération courante. La probabilité d'accepter une solution X' telle que $H(X) - H(X') < 0$ à l'itération i, est :

$$p(X', i) = exp\left(\frac{H(X) - H(X')}{T_i}\right) \tag{3.28}$$

Algorithme 1 : Algorithme de principe de la recherche locale stochastique

Entrées : X_0 : Solution initiale aléatoire et réalisable ;
Variables : X : Solution courante ;
X' : Solution candidate ;
Initialisation : $X := X_0$;
1 **Début**
2 **Tant que** *Test d'arrêt est faux* **faire**
3 X' := choisir aléatoirement et uniformément un voisin dans le système de voisinage de X ;
Si *H(X')≤ H(X)* **alors**
4 X :=X' ;
5 **Finsi**
6 **Fintq**
7 **Retourner** X
8 **Fin**

Nous présenterons dans cette partie un recuit simulé inhomogène dans lequel la température diminue à chaque itération suivant une progression géométrique. Ainsi à chaque itération i, la température sera mise à jour de la façon suivante :

$$T_i = T_{i-1} * \alpha; \quad \forall i = 1, ..., iterMax \tag{3.29}$$

On fixe la température de départ T_0 selon l'algorithme de (Kirkpatrick et al., 1983). Pour que notre algorithme converge efficacement vers une solution optimale, on fixe par expérimentations T_a la température de fin souhaitée. Le nombre total d'itérations, et donc d'applications de l'équation (3.29), est noté $iterMax$. On en déduit α selon l'équation suivante :

$$\alpha = \sqrt[iterMax]{\frac{T_a}{T_0}} \tag{3.30}$$

3.4.1.3 La recherche locale itérée

Le principe de la recherche locale itérée (Lourenço et al., 2003) est d'effectuer une suite de recherches locales, en effectuant une perturbation du minimum local entre chacune d'entre elles. L'algorithme de principe de la recherche locale itérée, que nous allons utiliser pour résoudre notre problème, est donné par l'algorithme 3.

Pour parcourir notre espace de recherche restreint aux minimums locaux, nous devons déterminer les critères d'acceptation entre la solution courante et la nouvelle solution obtenue après perturbation et recherche locale. Nous en utiliserons deux différents (Correia et al., 2012b) :

— celui de la recherche locale ($ILS|\,LS$) qui consiste à remplacer la solution courante $X*$ par la solution candidate X' si sa qualité est équivalente ou meilleure :
Si $H(X') \leq H(X^*)$ alors retourner X', sinon retourner X^*
— celui du recuit simulé inhomogène ($ILS|\,SA$) (Metropolis et al., 1953) qui consiste à accepter toutes les solutions de qualité équivalente et meilleure ainsi que des solutions moins bonnes selon une probabilité décroissante dans le temps. Ce critère

Algorithme 2 : Algorithme de principe du recuit simulé

Entrées : X_0 : Solution initiale aléatoire et réalisable ;
T_0 : Température initiale ;
Variables : X^* : Meilleure solution trouvée ;
X : Solution courante ;
X' : Solution voisine ;
T : Température ;
Initialisation : $X := X_0$;$X^* := X_0$;$T := T_0$

```
1  Début
2      Tant que T ≥ T_a faire
3          X' := choisir aléatoirement et uniformément un voisin dans le système de
            voisinage de X ;
           Si exp((H(X)−H(X'))/T_i) > random[0,1[ alors
4              X := X'
               Si H(X) < H(X*) alors
5                  X* := X
6              Finsi
7          Finsi
8          T_i := α * T_i ;
9      Fintq
10     Retourner X*
11 Fin
```

Algorithme 3 : Algorithme de principe de la recherche locale itérée

Entrées : X_0 : Solution initiale ;
Variables : X^* : Meilleure solution trouvée ;
X' : Solution voisine ;
Initialisation : $X^* := X'^* :=$ Recherche locale sur X_0 ;

```
1  Début
2      Tant que Test d'arrêt est faux faire
3          X' := Perturbation de X'* ;
           X' := Recherche locale sur X' ;
           X'* := Critère d'acceptation de X' par rapport à X'* en prenant en
           compte l'historique ; Si H(X) < H(X*) alors
4              X* := X
5          Finsi
6      Fintq
7      Retourner X*
8  Fin
```

est celui provenant du recuit simulé inhomogène (algorithme 2) :
Si $exp(\frac{H(X^*)-H(X')}{T}) > random[0,1[$ alors retourner X', sinon retourner X^*

Dans le cas ou la recherche locale est stochastique, un autre paramètre à déterminer est le critère d'arrêt des recherches locales. L'intérêt de ces recherches locales est de trouver un minimum local. L'objectif est donc d'arrêter une recherche locale lorsque qu'elle a convergé dans un minimum local. Cependant on ne peut pas être certain de se trouver dans un tel minimum. On utilise donc une règle qui donne une bonne probabilité d'être dans un minimum local. Cette règle est la suivante : Si au bout d'un certain nombre d'itérations on n'améliore pas strictement la qualité de la solution courante, on stoppe la recherche locale. On nomme ce nombre maximal d'itérations sans amélioration : le palier.

3.4.2 Principe de couplage

Nous proposons de coupler les méta-heuristiques précédemment présentées avec des algorithmes d'ordre strict (ou Schedule Generation Scheme) (Laurent et al., 2015). Le but est de représenter une solution par un codage ne décrivant que partiellement la solution. A chaque codage correspond alors un algorithme d'ordre strict, se basant sur une liste ordonnée, construisant une solution correspondante. Le principe de ce couplage est schématisé par la figure 3.8.

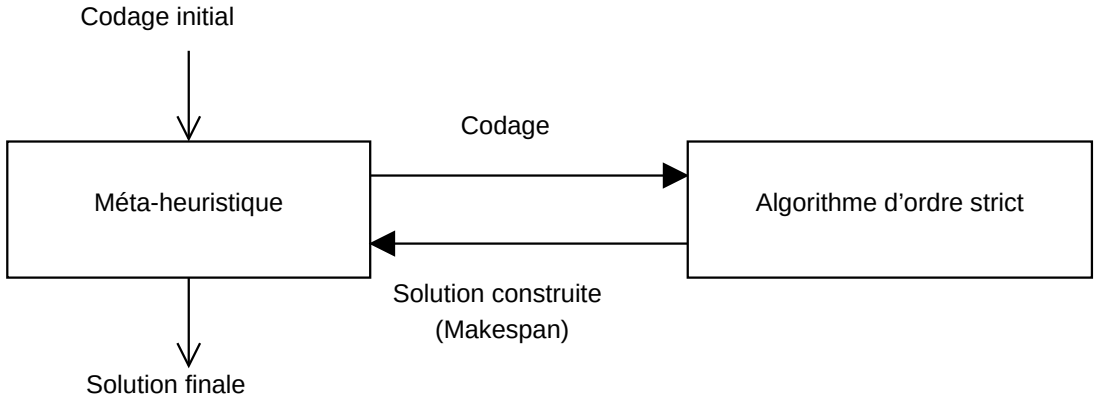

FIGURE 3.8 – Principe du couplage proposé

A chaque itération de la méta-heuristique, une solution partielle est obtenue en modifiant le codage d'une solution à l'aide d'un système de voisinage. La solution est alors construite et évaluée à l'aide d'un algorithme d'ordre strict. Puis la méta-heuristique accepte ou non la nouvelle solution et passe à l'itération suivante.

Pour utiliser ce principe de couplage nous définissons trois codages ainsi que les algorithmes d'ordre strict correspondants. Ces codages (σ, σ, l et σ, l, a) sont composés d'un ou de plusieurs composants décrivant la solution. Ces composants sont une liste ordonnée de tâche σ, une liste d'affectation de tâches aux sites l et une matrice d'affectation a des ressources aux tâches. Nous définissons aussi des systèmes de voisinage pour explorer l'espace des solutions.

3.4.3 Le codage σ

3.4.3.1 Composition du codage σ

Le premier codage proposé (Laurent et al., 2017) est celui utilisé couramment dans la résolution du problème de RCPSP et plus généralement de problèmes d'ordonnancement. Une solution X est simplement représentée par un vecteur :

$$X = (\sigma) \tag{3.31}$$

Le vecteur σ correspond à une séquence des tâches telle que :

— $\sigma = (\sigma_1, \sigma_2, ..., \sigma_N)$ est une séquence des N tâches qui désigne une liste topologique respectant les contraintes de précédence. σ_1 et σ_N sont les tâches fictives de début et de fin de projet. Chaque ressource exécutera les tâches dans l'ordre de cette séquence, c'est-à-dire que si une tâche j précède une tâche i dans σ, alors aucune ressource ne pourra exécuter i avant j.

Un ordonnancement est également défini par une date de début pour toutes les tâches, une affectation des ressources aux tâches ainsi qu'une affectation des tâches aux sites. Ce codage seul ne représente pas un ordonnancement, mais potentiellement plusieurs ordonnancements qui respectent la séquence σ. Pour déterminer un ordonnancement, nous appliquerons à ce codage un algorithme d'ordre strict, en nous inspirant des travaux de (Carlier, 1984).

3.4.3.2 Algorithme d'ordre strict du codage σ

Un algorithme d'ordre strict a pour principe *d'ordonnancer les tâches le plus tôt possible en respectant les contraintes de précédence et les disponibilités des ressources, dans l'ordre donné par une liste topologique* σ (Carlier, 1984).

Pour le codage σ, nous disposons seulement d'une liste topologique σ. Dans l'ordre donné par la liste de tâches σ, l'algorithme va déterminer la date de fin d_j des tâches $\forall j = 1, .., N$. Pour cela l'algorithme doit déterminer l'ensemble $a_j = a_{j,1}, .., a_{j,K}$ des ressources qui vont exécuter la tâche σ_j et sur quel site l_{σ_j}. Le principe d'affectation est le suivant :

Pour chaque tâche $j = 2, .., N-1$ nécessitant $r_{j,k}$ ressources de type $k = 1, .., K$ l'algorithme détermine **pour chaque site** les $r_{j,k}$ ressources disponibles au plus tôt pour exécuter la tâche. A ce stade $j-1$ tâches ont été ordonnancées, on connaît donc les dates de fin des $j-1$ premières tâches ainsi que les dates de disponibilité des ressources à l'étape j.

Pour une ressource $r = 1, ..., R_k$ de type $k = 1, ..., K$, nous distinguons trois cas pour calculer sa date de disponibilité $av^s_{k,r}$ pour réaliser la tâche σ_j sur le site s :

— Soit cette ressource est mobile et sa date de disponibilité au plus tôt pour commencer la tâche est égale à :
  — $av^s_{k,r} = 0$ si elle n'a encore aucune tâche assignée
  — $av^s_{k,r} = d_h + \delta(l_h, s)$ si sa dernière tâche assignée est h
— Soit cette ressource est fixe telle que : $s \neq loc_{k,r}$ et la ressource ne peut pas être affectée à j : $av^s_{k,r} = +\infty$
— Soit cette ressource est fixe telle que : $s = loc_{k,r}$ et sa date de disponibilité au plus tôt pour commencer la tâche $\sigma_j \in \{1, N\}$ est égale à $av^s_{k,r} = d_h$ avec h la dernière tâche assignée à la ressource.

On considère $Y^s_{\sigma_j,k,r} = 1$ pour les $r_{\sigma_j,k}$ ressources disponibles au plus tôt sur le site s. Pour le site s, la date de fin d'exécution $d^s_{\sigma_j}$ de la tâche σ_j est donnée par l'équation (3.32).

$$d^s_{\sigma_j} = \max(A^s, B^s) + p_{\sigma_j} \tag{3.32}$$
$$A^s = \max(d_{\sigma_h} + \delta(l_h, s); \forall h \in P_{\sigma_j}) \tag{3.33}$$
$$B^s = \max(av^s_{k,r}; \forall k, r | Y^s_{\sigma_j,k,r} = 1) \tag{3.34}$$

A^s représente la plus grande date de fin (temps de déplacement inclus sur le site s) des tâches qui doivent précéder σ_j. B^s représente la plus grande date de disponibilité sur le site s des ressources que l'on a assignées à σ_j.

On connaît alors la date d'exécution au plus tôt pour tous les sites pour la tâche j. L'algorithme affecte donc la tâche au site qui peut l'exécuter au plus tôt ($\min_{s=1,..,S}(d^s_{\sigma_j})$). Il affecte la tâche σ_j au site s ($l_j = s$) pouvant exécuter la tâche le plus tôt. Les $r_{\sigma_j,k}$ ressources de type k qui ont les dates de disponibilité $av^s_{k,r}$ les plus petites (a_{σ_j}), sont ensuite affectées à la tâche σ_j. Les disponibilités ($av^s_{k,r}$) sont alors mises à jour pour les $r_{\sigma_j,k}$ ressources affectées et on en déduit ainsi la date de fin d'exécution de la tâche σ_j.

L'algorithme 4 permet de définir les dates de fin d'exécution de chaque tâche et donc de calculer différents critères d'évaluation basés sur les dates, dont le makespan.

Algorithme 4 : Algorithme d'ordre strict pour le codage σ (H(X))

Entrées : $X = (\sigma)$
Variables : d : Vecteur des dates de fin des tâches ;
av : Matrice des dates de disponibilité des ressources sur les sites ;
Initialisation : $d := \{0, ..., 0\}$;
1 **Début**
2 **Pour** $j = 2$ *à* N **faire**
3 **Pour** $s = 1$ *à* S **faire**
4 Calculer les valeurs de av^s pour σ_j en prenant en compte d;
5 Calculer la date de fin $d^s_{\sigma_j}$ (3.32);
6 **Finpour**
7 $d_j = \min_{s=1,..S}(d^s_{\sigma_j})$;
8 Affecter la tâche au site et aux ressources qui peuvent l'exécuter au plus tôt;
9 Mettre à jour des dates de disponibilité des ressources av;
10 **Finpour**
11 **Retourner** d_N
12 **Fin**

3.4.3.3 Exemple

Nous présentons dans cette partie un exemple de représentation du codage σ avec la solution obtenue par application de l'algorithme d'ordre strict. L'instance du RCPSP Multi-Site présentée comporte 2 sites, 7 ressources de 2 types différents et 9 tâches. Les caractéristiques des 7 ressources sont données dans le tableau 3.2. Le temps de transport

d'un site à l'autre est de deux périodes. Les 9 tâches sont reliées par des contraintes de précédence données sur la figure 3.9. Les tâches 1 et 9 sont les deux tâches fictives de début et fin de projet. Chaque tâche est représentée par un cercle avec au-dessus sa durée et en-dessous les ressources nécessaires à son exécution sous la forme "type (quantité)".

	Type	Statut	Site d'appartenance
R1,1	1	Fixe	1
R1,2	1	Mobile	-
R1,3	1	Mobile	-
R1,4	1	Fixe	2
R2,1	2	Fixe	1
R2,2	2	Mobile	-
R2,3	2	Fixe	2

Tableau 3.2 – Liste des ressources disponibles

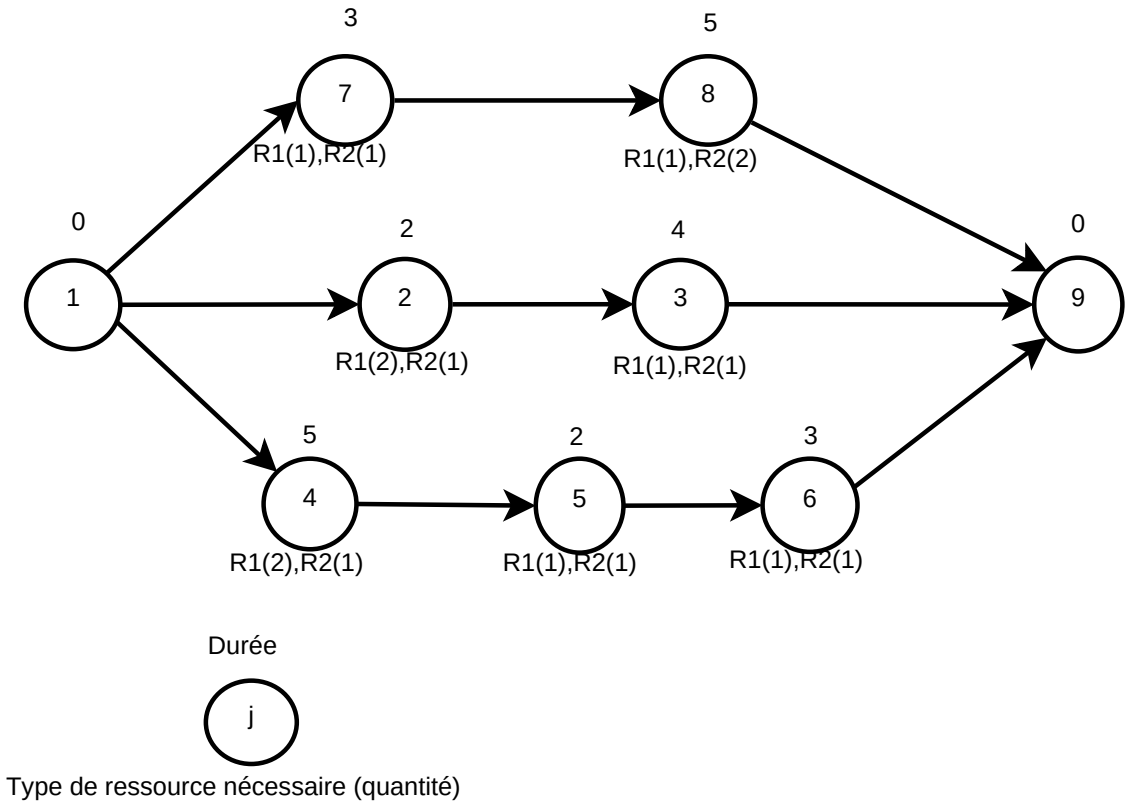

Figure 3.9 – Graphe de précédence du problème

Soit X une instance du codage σ donnée dans le tableau 3.3.

$\sigma =$	(1,2,3,4,5,6,7,8,9)

Tableau 3.3 – Un codage de type σ pour le problème

On applique l'algorithme d'ordre strict (algorithme 4) correspondant au codage σ

sur l'instance X décrite dans le tableau 3.3. On obtient la solution représentée par la diagramme de Gantt de la figure 3.10.

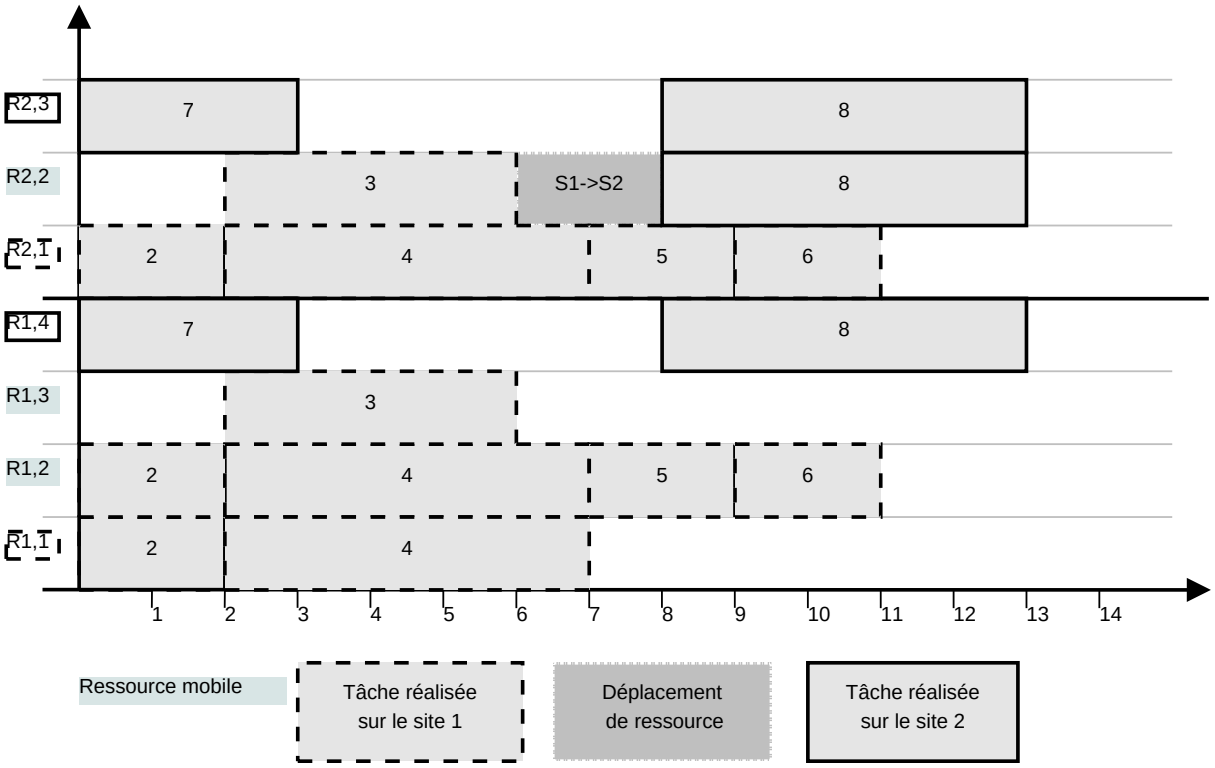

FIGURE 3.10 – Solution obtenue par l'algorithme d'ordre strict à partir de X

L'algorithme d'ordre strict ordonnance les tâches dans l'ordre σ. Les tâches 2 à 6 sont toutes les deux assignées sur le site 1. En effet, une fois les tâches 2 et 4 assignées sur le site 1, assigner les autres tâches au site 2 ne ferait que provoquer des temps de déplacement qui repousseraient la date de fin des tâches 3, 5 et 6. Les tâches 7 et 8 sont elles effectuées sur le site 2 car c'est le site qui peut les exécuter au plus tôt. L'exécution de la tâche 8 sur le site 2 entraîne cependant un déplacement de la ressource $R2,2$ du site 1 vers le site 2 pour exécuter la tâche 8 au plus tôt.

3.4.4 Le codage σ, l

3.4.4.1 Composition du codage σ, l

Le deuxième codage proposé représente une solution X par la concaténation de deux vecteurs (Laurent et al., 2017) :

$$X = (\sigma, l) \tag{3.35}$$

Le vecteur σ correspond à une séquence des tâches et le vecteur l correspond à l'affectation des tâches aux sites, tels que :

— $\sigma = (\sigma_1, \sigma_2, ..., \sigma_N)$ est une permutation des N tâches qui désigne une liste topologique respectant les contraintes de précédence. σ_1 et σ_N sont les tâches fictives de début et de fin de projet. Chaque ressource exécutera les tâches dans l'ordre de cette séquence, c'est-à-dire que si une tâche j précède une tâche i dans σ, alors aucune ressource ne pourra exécuter i avant j.

— $l = (l_2, l_3, ...l_{N-1})$ avec $l_j = 1, ..., S$ le site où la tâche $j = 2, ..., N-1$ va être exécutée. Cette affectation respecte le nombre de ressources disponibles sur le site (fixes + mobiles) : une tâche ne peut pas être affectée à un site qui ne pourra pas l'exécuter. Pour rappel, les tâches fictives n'ont pas d'affectation à un site.

Pour construire une solution, l'algorithme d'ordre strict va devoir déterminer quelles ressources exécutent les tâches ainsi que leur date de début.

3.4.4.2 Algorithme d'ordre strict

Dans ce second cas, nous disposons d'une liste topologique σ et d'un vecteur de sites l. A chaque itération de l'algorithme, le vecteur l va permettre de déterminer les temps de déplacement des ressources entre les sites et ainsi de calculer la date de fin d_j de la tâche $j = 1, .., N$. L'intérêt du vecteur l est d'explorer de nouvelles solutions où l'affectation des tâches aux sites n'est plus déterminé par l'algorithme d'ordre strict, mais par le codage. Les dates de disponibilité des ressources sont déduites comme présenté dans la partie précédente. L'algorithme détermine l'affectation des ressources aux tâches et la date de fin des tâches dans l'ordre σ en choisissant l'affectation au plus tôt. L'algorithme 5 permet de définir les dates de fin d'exécution de chaque tâche et donc de calculer différents critères d'évaluation basés sur les dates, dont le makespan.

Algorithme 5 : Algorithme d'ordre strict pour le codage σ, l (H(X))

Entrées : $X = (\sigma, l)$
Variables : d : Vecteur des dates de fin des tâches ;
av : Matrice des dates de disponibilité des ressources ;
Initialisation : $d := \{0, ..., 0\}$;
1 **Début**
2 **Pour** $j = 2$ *à* N **faire**
3 Calculer les valeurs de av pour σ_j en prenant en compte l et d ;
Affecter les ressources à σ_j ;
Calculer la date de fin d_{σ_j} (3.32) ;
Mettre à jour des dates de disponibilité des ressources ;
4 **Finpour**
5 **Retourner** d_N
6 **Fin**

3.4.4.3 Exemple

Nous reprenons dans cette partie l'exemple présenté dans la partie 3.4.3.3. Soit X une instance du codage σ, l donnée dans le tableau 3.4.

$\sigma =$	(1,2,3,4,5,6,7,8,9)
l_j	$(1, 2, 1, 2, 2, 1, 1)$

Tableau 3.4 – Un codage de type σ, l pour le problème

On applique l'algorithme d'ordre strict (algorithme 5) correspondant au codage σ, l sur l'instance X décrite dans le tableau 3.4. On obtient la solution représentée par la diagramme de Gantt de la figure 3.11.

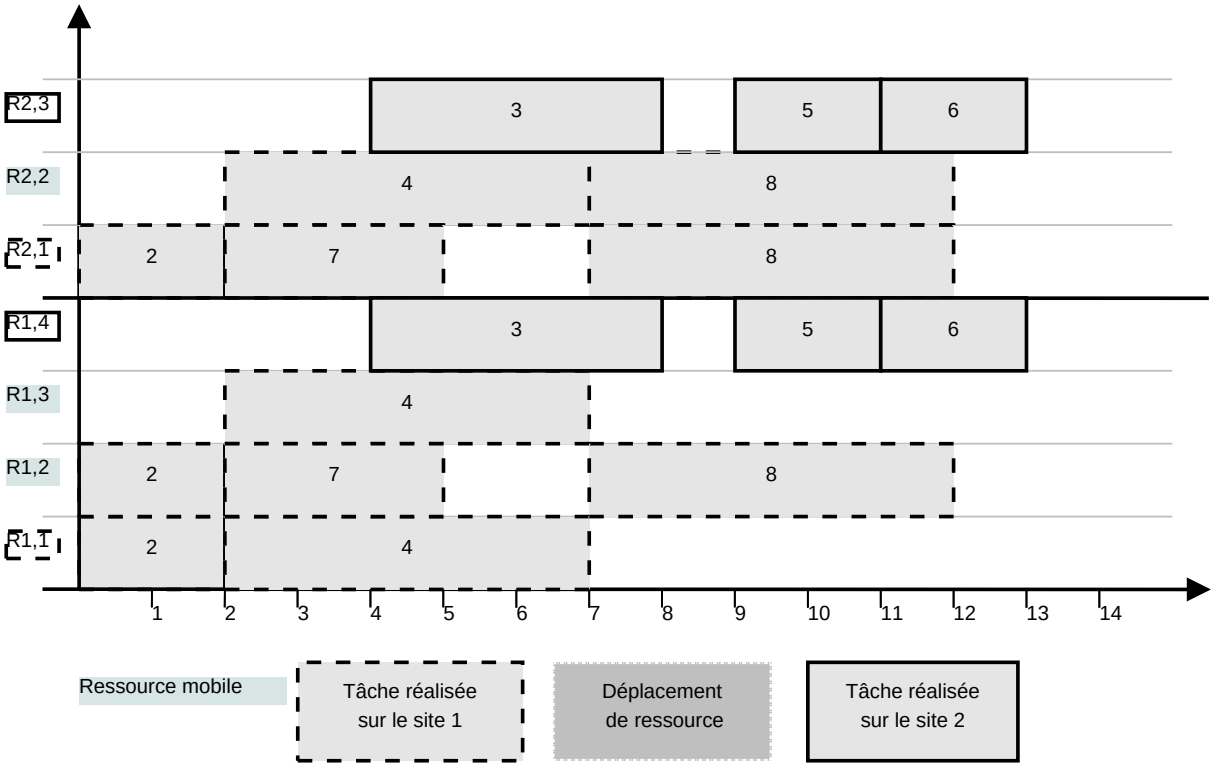

FIGURE 3.11 – Solution obtenue par l'algorithme d'ordre strict à partir de X

L'algorithme d'ordre strict ordonnance les tâches dans l'ordre σ sur le site défini par l. Aucune ressource mobile n'est déplacée dans cet exemple. Cependant un temps de transport s'applique entre les tâches 2 et 3 ainsi que les tâches 4 et 5 qui sont effectuées sur deux sites différents. Ces temps de transport sont dus aux contraintes de précédence entre les tâches 2 et 3 et les tâches 4 et 5. Ainsi la tâche 3 ne peut être effectuée sur le site 2 avant la période 4 et la tâche 5 avant la période 9.

3.4.5 Le codage σ, l, a

3.4.5.1 Composition du codage σ, l, a

Le dernier codage consiste en un ensemble de trois vecteurs (Laurent et al., 2017) :

$$X = (\sigma, l, a) \tag{3.36}$$

Le vecteur σ correspond à une séquence des tâches, le vecteur l correspond à l'affectation des tâches aux sites et a correspond à l'affectation des ressources aux tâches, tels que :

— $\sigma = (\sigma_1, \sigma_2, ..., \sigma_N)$ est une permutation des N tâches qui désigne une liste topologique respectant les contraintes de précédence. σ_1 et σ_N sont les tâches fictives de début et de fin de projet. Chaque ressource exécutera les tâches dans l'ordre

de cette séquence, c'est-à-dire que si une tâche j précède une tâche i dans σ, alors aucune ressource ne pourra exécuter i avant j.

— $l = (l_2, l_3, ...l_{N-1})$ avec $l_j \in \{1, .., S\}$ le site, parmi les S sites, où la tâche $j \in \{2, .., N-1\}$ va être exécutée. Cette affectation respecte le nombre de ressources disponibles sur le site (fixes + mobiles) : une tâche ne peut pas être affectée à un site qui ne pourra pas l'exécuter.

— $a = (a_2, a_3, ...a_{N-1})$ avec a_j l'ensemble des ressources affectées à la tâche j tel que $a_j = \{a_{j,1}, .., a_{j,\sum_{\forall k} r_{j,k}}\}$ avec $a_{j,n}$ la n ième ressources affectée à la tâche j. Cette affectation respecte la demande de la tâche j ainsi que le site d'affectation l_j. Une ressource fixe dont le site $loc_{k,r}$ est différent du site l_j ne peut être affectée à la tâche j.

3.4.5.2 Algorithme d'ordre strict

L'objectif de l'algorithme d'ordre strict est ici de calculer la date de début des tâches. L'algorithme dispose d'une liste topologique σ, d'un vecteur de sites l et d'une affectation a complète de toutes les ressources aux tâches. A chaque itération de l'algorithme, la matrice d'affectation des ressources a couplée au vecteur l va permettre de déterminer les temps de déplacement des ressources entre les sites et ainsi de calculer la date de fin d_j de la tâche $j = 1, .., N$ au plus tôt. Les dates de disponibilité des ressources sont déduites comme présenté dans la partie 3.4.3 sur le codage σ. L'algorithme détermine la date de fin au plus tôt des tâches dans l'ordre σ. L'algorithme 6 permet de définir les dates de fin d'exécution de chaque tâche et donc de calculer différents critères d'évaluation basés sur les dates, dont le makespan.

Algorithme 6 : Algorithme d'ordre strict pour le codage σ, l, a (H(X))

Entrées : $X = (\sigma, l, a)$
Variables : d : Vecteur des dates de fin des tâches ;
av : Matrice des dates de disponibilité des ressources ;
Initialisation : $d := \{0, ..., 0\}$;
Début
  Pour $j = 2$ *à* N **faire**
    Calculer la date de fin d_{σ_j} (3.32);
    Mise à jour des dates de disponibilité des ressources;
  Finpour
  Retourner d_N
Fin

3.4.5.3 Exemple

Nous reprenons dans cette partie l'exemple présentée dans la partie 3.4.3.3. Soit X une instance du codage σ, l, a donnée dans le tableau 3.5.

On applique l'algorithme d'ordre strict (algorithme 6) correspondant au codage σ, l, a sur l'instance X décrite dans le tableau 3.5. On obtient la solution représentée par la diagramme de Gantt de la figure 3.12.

$\sigma =$	(1,2,3,4,5,6,7,8,9)
l_j	$(2, 2, 1, 1, 1, 2, 2)$
a_2	$\{R1, 3R1, 4R2, 3\}$
a_3	$\{R1, 4R2, 3\}$
a_4	$\{R1, 1R1, 2R2, 1\}$
a_5	$\{R1, 2R2, 1\}$
a_6	$\{R1, 2R2, 1\}$
a_7	$\{R1, 3R2, 3\}$
a_8	$\{R1, 3R2, 2R2, 3\}$

TABLEAU 3.5 – Un codage de type σ, l, a pour le problème

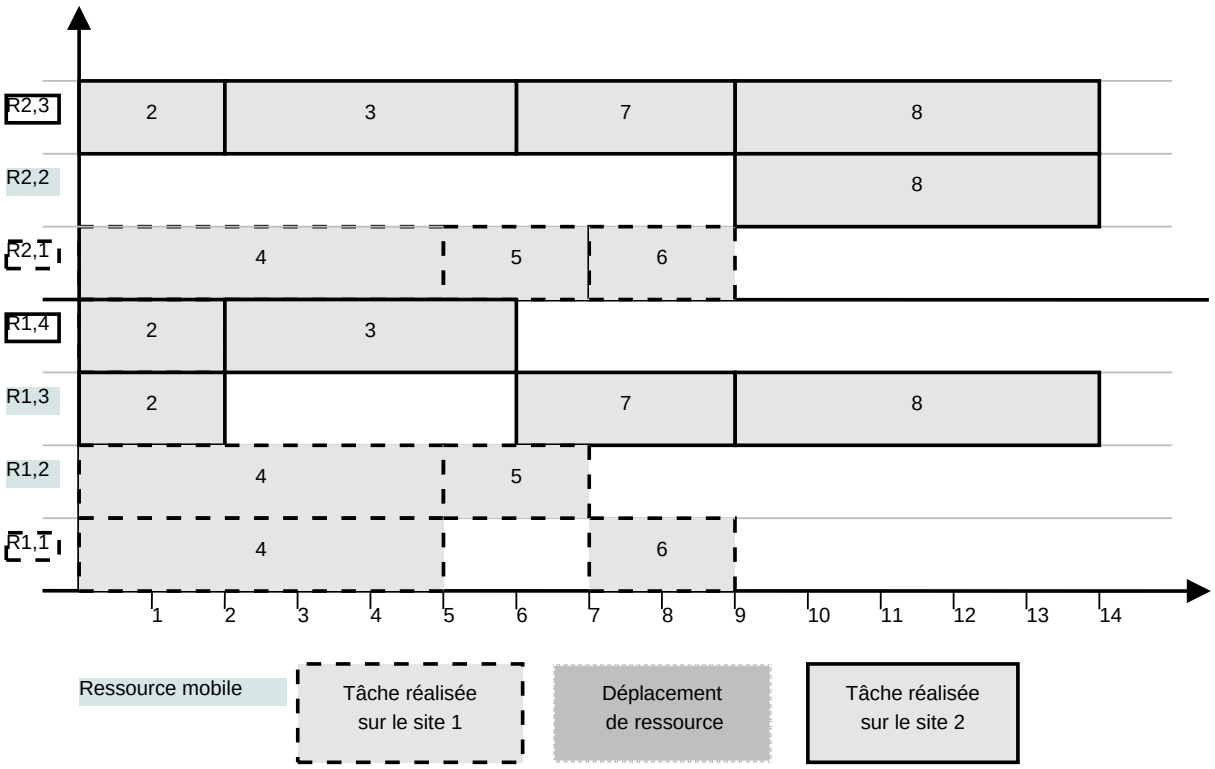

FIGURE 3.12 – Solution obtenue par l'algorithme d'ordre strict à partir de X

L'algorithme d'ordre strict ordonnance les tâches dans l'ordre σ sur le site défini par l avec les ressources définies par a. L'algorithme d'ordre strict ne fait que programmer les tâches au plus tôt sur les ressources. Pour cette solution X aucun temps de déplacement ne s'applique, ni pour les contraintes de précédence ni pour les ressources. Cependant on remarque qu'une meilleure solution peut être obtenue à partir de cette solution en modifiant simplement une affectation d'une ressource. En effet si on remplace l'affectation de la ressource $R2,3$ par la ressource $R2,2$ pour la tâche 7, le Makespan de la solution est réduit de 3 périodes. La tâche 7 sera alors programmée en période 2, ce qui permet à la tâche 8 d'être programmée à la période 6. Ainsi la dernière tâche est la tâche 8 et se termine à la période 11 (période 14 avant le changement).

La section suivante présente les différents mouvements et systèmes de voisinage utilisés pour modifier les solutions au sein des méta-heuristiques.

3.4.6 Systèmes de voisinage

Nous décomposons nos systèmes de voisinage en trois mouvements, V1 pour la séquence σ, V2 pour l'affectation des sites l et V3 pour l'affectation des ressources a.

3.4.6.1 Le mouvement V1

Le premier mouvement (algorithme 7) concerne σ. C'est un mouvement de type insertion. Le principe est de déplacer une tâche σ_p de la position $p = 2, N-1$ dans le vecteur σ à la position $p' = 2, N-1$, $p' \neq p$. On n'applique pas le déplacement de la tâche si la nouvelle position viole une contrainte de précédence. Les contraintes de précédence sont violées si :

- $p < p'$ et il existe au moins une tâche à la position p'' entre p et p' telle que σ_p précède $\sigma_{p''}$
- $p > p'$ et il existe au moins une tâche à la position p'' entre p et p' telle que $\sigma_{p''}$ précède σ_p

Algorithme 7 : Algorithme de principe du mouvement V1

Entrées : σ : séquence initiale ;
Variables : σ' : séquence voisine ;
Début
  Choisir aléatoirement et uniformément deux positions $p \in [2, N-1]$ et $p' \in [2, N-1]/p' \neq p$ tel que le déplacement de σ_p à la position p' ne viole pas les contraintes de précédence;
  Si $p < p'$ **alors**
    $\sigma' = (\sigma_1, ...\sigma_{p-1}, \sigma_{p+1}, ..., \sigma_{p'}, \sigma_p, ..., \sigma_N)$
  sinon
    $\sigma' = (\sigma_1, ..., \sigma_p, \sigma_{p'}, ...\sigma_{p-1}, \sigma_{p+1}, ..., \sigma_N)$
  Finsi
  Retourner σ'
Fin

3.4.6.2 Le mouvement V2

Le deuxième mouvement (algorithme 8) consiste à modifier le site s assigné à une tâche $j = 2, N-1$ en un site s' . Le mouvement est appliqué si et seulement si la solution reste réalisable c'est à dire si, pour chaque type r de ressource nécessaire à j, le nombre de ressources mobiles de type r, plus le nombre de ressources fixes de type r sur le site s' est supérieur ou égal au nombre de ressources demandées. Lorsque qu'un mouvement est appliqué, les ressources fixes étant affectées à la tâche déplacée ne peuvent plus exécuter la tâche sur le nouveau site. Une étape de réaffectation des ressources fixes est donc nécessaire. A noter que cette étape n'est utile que pour le codage σ, l, a, puisque pour le codage σ, l l'affectation des ressources est gérée par l'algorithme d'ordre strict. Pour cela, toutes les ressources fixes sont désaffectées de la tâche. Pour les remplacer, on sélectionne aléatoirement des ressources compatibles et qui ne sont pas encore affectées à cette tâche.

Algorithme 8 : Algorithme de principe du mouvement V2

Entrées : l : vecteur initial ;
Variables : l' : vecteur voisin ;
Début
  Choisir aléatoirement et uniformément une tâche j;
  Choisir aléatoirement et uniformément un site $l'_j \neq l_j$ tel que j puisse être exécuté sur le site l'_j;
  $l' = (l_2, ..., l'_j, ..., l_{N-1})$;
  $a'_j = a_j$ **Pour** *Toutes les ressources fixes* rf **faire**
    $a'_j = a'_j \backslash rf$;
    On détermine aléatoirement une ressource compatible r' $a'_j = a'_j \cup r'$;
  Finpour
  $a' = (a_1, ..., a'_j, ..., a_N)$;
  Retourner l' *et* a'
Fin

3.4.6.3 Le mouvement V3

Le troisième mouvement (algorithme 9) concerne a et donc uniquement le codage σ, l, a. Le principe consiste à modifier l'affectation $a_{j,r}$ d'une ressource r pour une tâche j par une autre ressource du même type r'. Le mouvement n'est pas appliqué si :

— La nouvelle ressource $r' \in a_j$
— La nouvelle ressource r' est fixe ($M_{r'} = 0$) et son site d'appartenance $loc_{r'} \neq l_j$

3.4.6.4 Construction du système de voisinage

Une fois ces mouvements définis, nous devons définir les systèmes de voisinage qui vont les utiliser. Les systèmes de voisinage consistent en un tirage aléatoire pondéré d'un des mouvements présentés précédemment. Pour chaque codage, une probabilité p_i est associée à chaque mouvement V_i telle que $\sum_{i=1}^{3} p_i = 1$. L'application d'un système de voisinage revient donc à appliquer à une solution, un **unique** mouvement V_i selon une probabilité p_i.

Algorithme 9 : Algorithme de principe du mouvement V3

Entrées : a : Affectation initiale ;
Variables : a' : Affectation voisine ;
Début
  Choisir aléatoirement et uniformément une affectation $a_{j,r}$ et une ressource de même type r telle que la substitution de $a_{j,r}$ par la ressource r' est un mouvement applicable;
  $a'_j = \{a_{j,1}, ..., r', ..., a_{j,r_{j,k}}\}$;
  $a' = (a_2, ..., a'_j, ..., a_{N-1})$;
  Retourner a'
Fin

3.4.7 Accessibilité des codages à une solution optimale

Ces trois codages couplés à leur algorithme d'ordre strict forment des ensembles de solutions imbriquées les uns dans les autres. Soit Ω_c l'ensemble des solutions atteignable par le codage $c \in \sigma, l, a; \sigma, l; \sigma$.

Proposition 1. $\Omega_\sigma \subseteq \Omega_{\sigma,l}$.

Preuve 1. *Pour un vecteur σ" donné, l'algorithme d'ordre strict du codage σ retourne une affectation des tâches aux sites l". Alors si on prend les vecteurs σ" et l" et qu'on leur applique l'algorithme d'ordre strict correspondant au codage σ, l, on construit la même solution qu'en appliquant l'algorithme d'ordre strict du codage σ au vecteur σ".*

Proposition 2. $\omega_{\sigma,l} \subseteq \Omega_{\sigma,l,a}$.

Preuve 2. *Pour un vecteur σ" et un vecteur l" donné, l'algorithme d'ordre strict du codage σ, l retourne une affectation des ressources aux tâches a". Alors si on prend les vecteurs σ", l" et a" et qu'on leur applique l'algorithme d'ordre strict correspondant au codage σ, l, a, on obtient la même solution qu'en appliquant l'algorithme d'ordre strict du codage σ, l aux vecteurs σ", l".*

On a la relation suivante sur les ensembles de solutions :

Proposition 3.

$$\Omega_\sigma \subseteq \Omega_{\sigma,l} \subseteq \Omega_{\sigma,l,a} \tag{3.37}$$

Preuve 3. *Pour prouver la proposition 3.37, nous avons prouvé que $\sigma \subseteq \sigma, l$ et $\sigma, l \subseteq \sigma, l, a$ et donc par transitivité $\sigma \subseteq \sigma, l \subseteq \sigma, l, a$*

Cela a été simplement démontré par le fait que chaque solution de Ω_σ peut être construite par un codage σ, l et que chaque solution de $\Omega_{\sigma,l}$ peut être obtenue à l'aide d'un codage σ, l, a.

On obtient donc la relation des ensembles représentée dans le schéma de la figure 3.13

A est l'ensemble des solutions optimales atteignables par le codage σ, l, a, B est l'ensemble des solutions optimales atteignables par le codage σ, l et C est l'ensemble des solutions optimales atteignables par le codage σ. Grâce aux preuves 1 et 2, on en déduit la relation suivante sur ces ensembles :

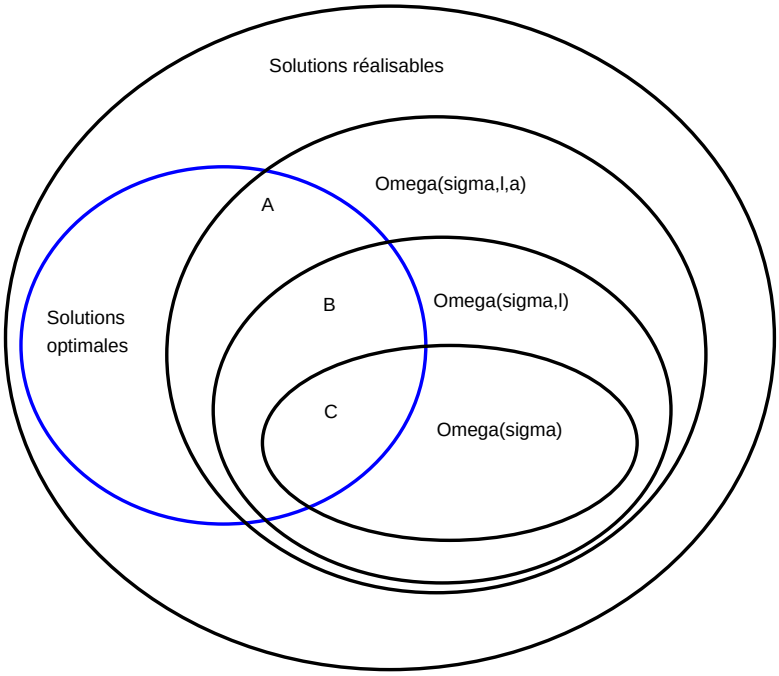

FIGURE 3.13 – Relation des ensembles de solutions définis par les codages

$$C \subseteq B \subseteq A \tag{3.38}$$

Se pose alors la question de l'existence des ensembles de solutions A, B, C définis par la figure 3.13.

Proposition 4. *Le codage σ, l, a garantit qu'au moins une solution optimale est atteignable.*

Preuve 4. *Soit une solution optimale s pour une instance donnée.*

- *Soit σ' un ordre qui respecte l'ordre dans lequel les tâches sont exécutées sur chaque ressource, c'est à dire que si une tâche j est exécutée avant une autre tâche j' sur une même ressource ou possède une relation de précédence avec j', alors j doit précéder j' dans l'ordre σ.*
- *Soit l' une affectation des tâches aux sites identique à la solution optimale s.*
- *Soit a' une affectation des ressources aux tâches identiques à la solution s.*

Alors la solution s' obtenue par l'algorithme d'ordre strict du codage σ, l, a sur les données σ', l', a' est optimale. En effet la solution s' correspond à la solution s où chaque tâche s'exécute au plus tôt. Cela ne peut pas détériorer la qualité de la solution puisque la date d'exécution pour chaque tâche ne peut être que plus tôt.

Définition 6. *Une solution X^* d'un codage c est dite optimale pour le codage c si il n'existe pas de solutions, dans l'ensemble Ω_c des solutions possibles de construire à l'aide de l'algorithme d'ordre strict de c, une solution de meilleure qualité (3.39).*

$$X^* | H(X^*) \leq H(X) \forall X \in \Omega_c \tag{3.39}$$

Proposition 5. *Le codage σ, l ne garantit pas l'accessibilité à une solution optimale.*

Preuve 5. *Pour démontrer cela nous présentons un exemple d'instance où aucune solution optimale ne peut être obtenue en appliquant l'algorithme d'ordre strict du codage σ, l. Soit une instance de 6 tâches avec 2 sites éloignés d'une durée de transport de 3 périodes. Les ressources disponibles sont données dans le tableau 3.6. Les durées et contraintes de précédence sont représentées sur le graphe de la figure 3.14.*

	Type	Mobilité	Site
R1,1 et R1,2	1	Mobile	X
R2,1 et R2,2	2	Mobile	X
R3,1 et R3,2	3	Fixe	2
R4,1 et R4,2	4	Fixe	1
R5	5	Fixe	1
R6	6	Fixe	2

Tableau 3.6 – Ressources disponibles pour exécuter les tâches

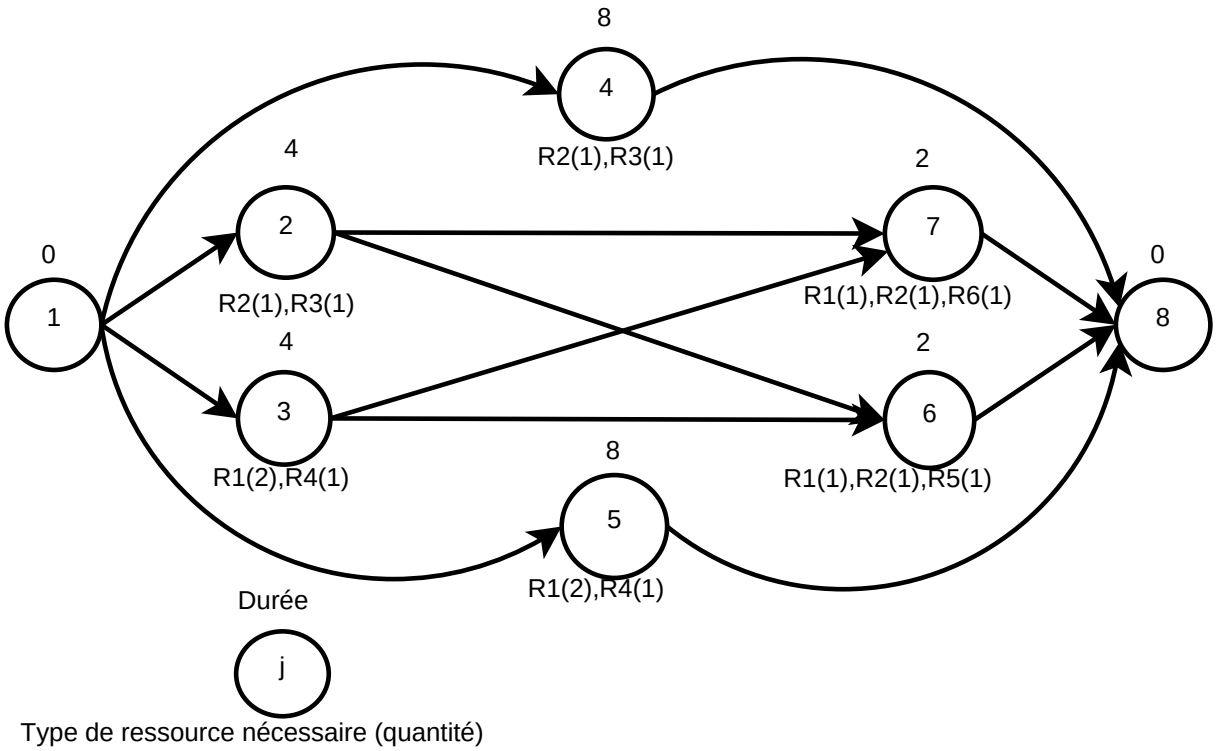

Figure 3.14 – Graphe des relations de précédence entre les tâches

Nous donnons le diagramme de Gantt d'une solution optimale sur la figure 3.15. Cette solution a un makespan de 10.

Une solution optimale pour le codage de type σ, l, a pour cette instance est donnée dans le tableau 3.7. La solution obtenue à l'aide de l'algorithme d'ordre strict a un makespan de 10 et est optimale.

Nous donnons à présent une solution optimale pour le codage σ, l, obtenu par énumération de toutes les solutions possibles, dans le tableau 3.8.

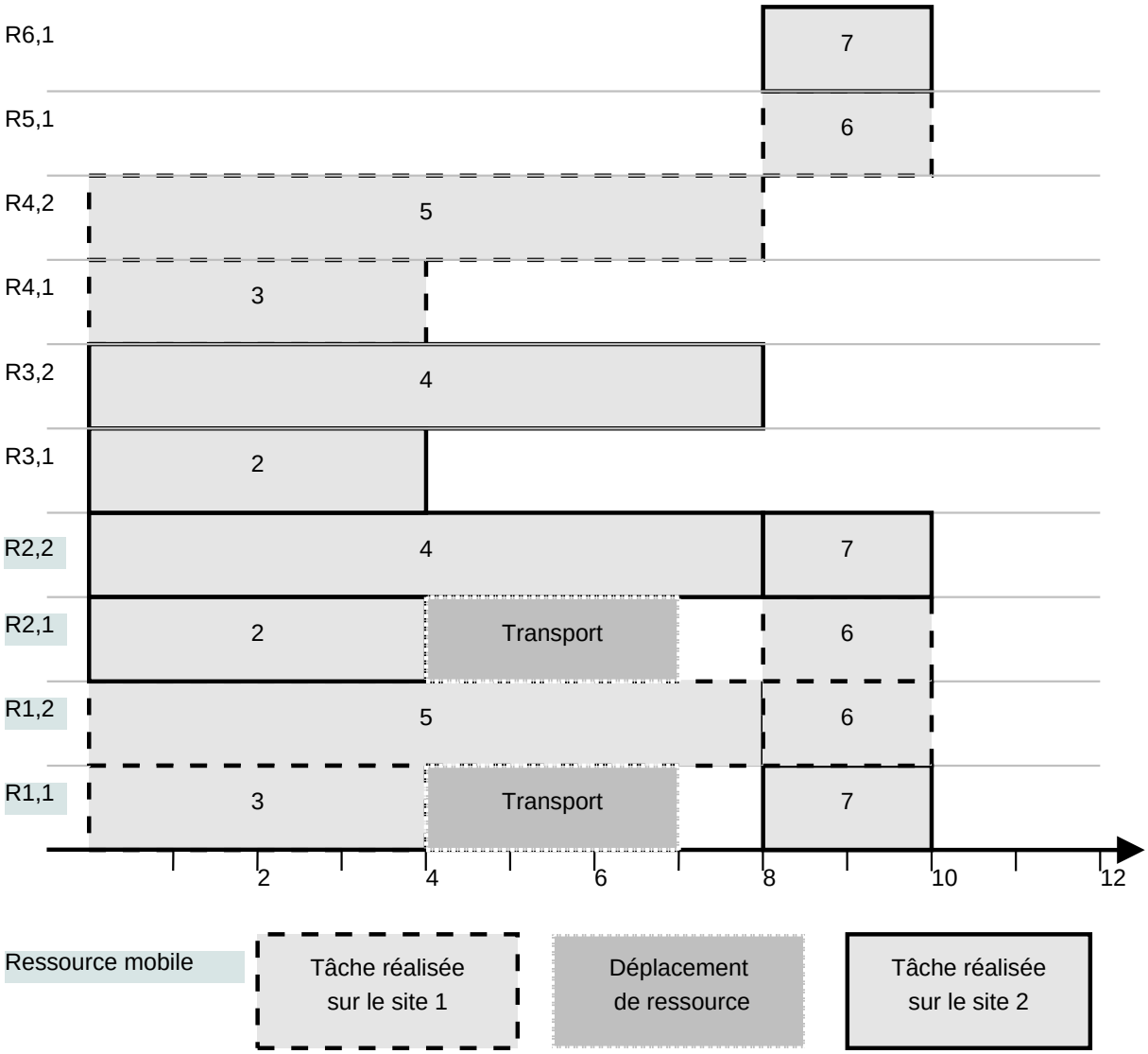

FIGURE 3.15 – Une solution optimale au problème

$\sigma =$	(1,2,3,4,5,7,6,8)
l_j	$(2,1,2,1,1,2)$
a_2	$\{R2, 1R3, 1\}$
a_3	$\{R1, 1R4, 1\}$
a_4	$\{R2, 2R3, 2\}$
a_5	$\{R1, 2R4, 2\}$
a_6	$\{R1, 2R2, 1R5, 1\}$
a_7	$\{R1, 1R2, 2R6, 1\}$

TABLEAU 3.7 – Une solution optimale pour le codage de type σ, l, a

$\sigma =$	(1,2,3,4,5,7,6,8)
l_j	$(2,1,2,1,1,2)$

TABLEAU 3.8 – Une solution optimale pour le codage de type σ, l

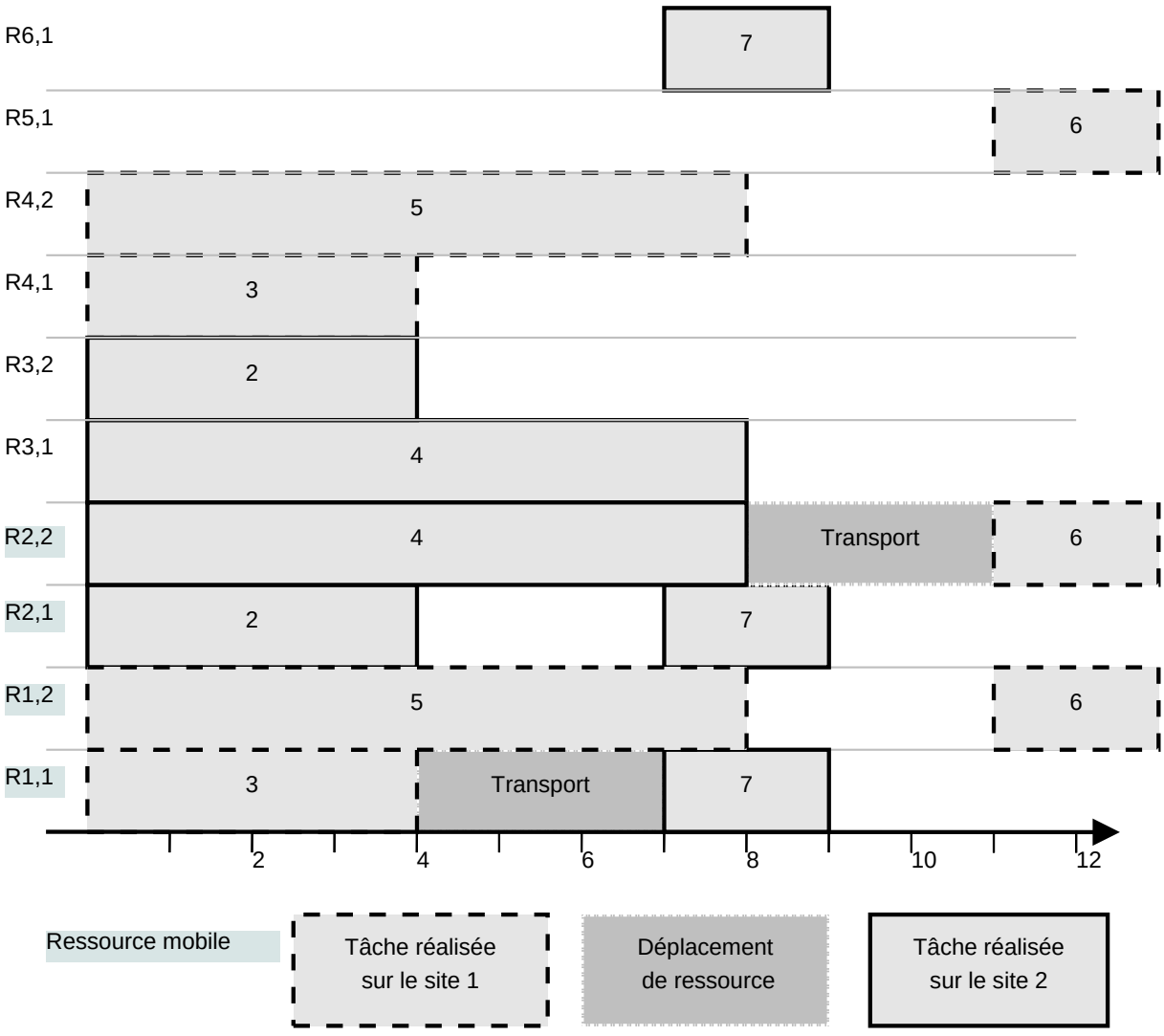

FIGURE 3.16 – Solution obtenue en appliquant l'algorithme d'ordre strict du codage σ, l

Nous noterons que cette solution possède les mêmes vecteurs σ et l que la solution optimale pour le codage σ, l, a. Cependant, la solution (figure 3.16) obtenue à l'aide de l'algorithme d'ordre strict a un makespan de 13. Cette solution n'est donc pas optimale ce qui prouve que le codage σ, l ne garantit pas l'accessibilité à une solution optimale.

Comme $\Omega_\sigma \subseteq \Omega_{\sigma,l}$ (équation 3.38) le codage σ ne garantit pas non plus l'accessibilité à une solution optimale.

Proposition 6. *Il existe des instances du RCPSP Multi-Site pour lesquelles le codage σ, l permet de trouver une solution optimale mais où le codage σ ne le permet pas.*

Preuve 6. *Pour prouver cela nous présentons une instance du RCPSP Multi-Site où une solution optimale existe dans $\Omega_{\sigma,l}$ mais pas dans Ω_σ. Soit une instance de 2 tâches liées par une contraintes de précédence (figure 3.17).*

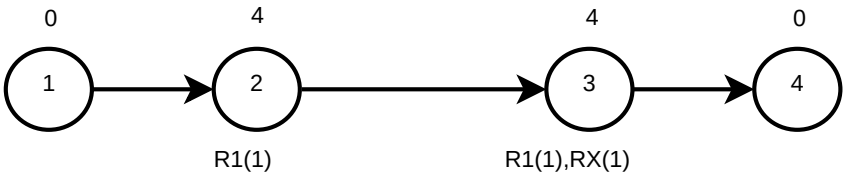

FIGURE 3.17 – Graphe de précédence de l'instance

La demande de la tâche 3 est une ressource de type 1 et une ressource de type "X". Ce type X représente en fait soit le type 2 soit le type 3. On a donc deux instances représentées par ce graphe, une instance où la tâche 3 nécessite une ressource de type 2 et une instance où la tâche 3 nécessite une ressource de type 3. Nous appellerons respectivement ces deux instances, $I2$ et $I3$. Les ressources sont réparties sur 2 sites séparés par un temps de transport de 2 périodes. Les ressources disponibles sont listées dans le tableau 3.9.

	Type	Mobilité	Site
R1,1	1	Fixe	2
R1,2	1	Fixe	1
R2,1	2	Fixe	1
R3,1	3	Fixe	2

TABLEAU 3.9 – Ressources disponibles pour exécuter les tâches

Des solutions optimales pour $I2$ et $I3$ sont respectivement données par les figures 3.18 et 3.19.

On constate que pour les deux instances le site où sont effectuées les tâches est différent. Toutes les tâches sont effectuées sur le site 1 pour l'instance $I2$ alors que toutes les tâches sont effectuées sur le site 2 pour l'instance $I3$. Cela est dû à la tâche 3 qui ne peut être effectuée que sur le site 1 pour $I2$ et que sur la site 2 pour $I3$. Ainsi la tâche 2 est affectée au même site que la tâche 3 pour éviter un temps de déplacement de 2 périodes sur le chemin critique. Les deux solutions optimales ont un makespan de 8. Deux codages σ, l, donnant les solutions optimales présentées, sont données dans le tableau 3.10 et 3.11.

Pour ce qui est du codage σ il existe une seule solution possible qui est donné dans le tableau 3.12.

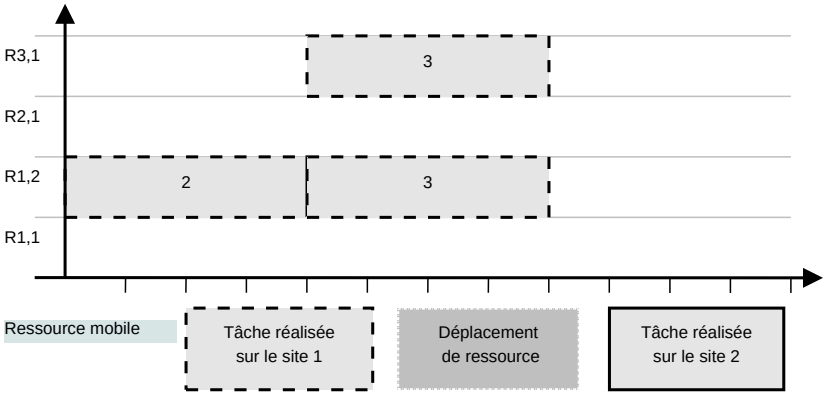

FIGURE 3.18 – Solution optimale pour l'instance $I2$

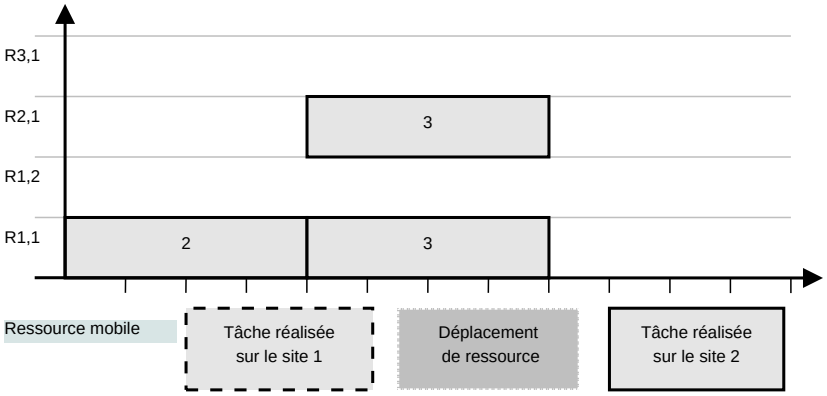

FIGURE 3.19 – Solution optimale pour l'instance $I3$

$\sigma =$	(1,2,3,4)
l_j	$(1,1)$

TABLEAU 3.10 – Codage optimal pour I2

$\sigma =$	(1,2,3,4)
l_j	$(2,2)$

TABLEAU 3.11 – Codage optimal pour I3

$\sigma =$	(1,2,3,4)

TABLEAU 3.12 – Seul codage possible Pour I2 et I3

Le codage donné dans le tableau 3.12 va construire une seule solution pour chacune des instances $I2$ et $I3$. L'algorithme d'ordre strict va donc d'abord affectée la tâche 2 à un site. Les deux sites peuvent exécuter la tâche à la période 0. Étant donné que l'algorithme d'ordre strict associé au codage σ est déterministe, l'algorithme va faire un choix déterministe, qui ne prend pas en compte les tâches qui suivent dans σ. Si l'algorithme affecte la tâche 2 au site 1, un temps de déplacement sera appliqué entre les tâches 2 et 3 et la solution ne sera donc pas optimale pour $I3$. Si l'algorithme affecte la tâche 2 au site 2, un temps de déplacement sera appliqué et la solution ne sera donc pas optimale pour $I2$. Quel que soit le choix déterministe que fera l'algorithme d'ordre strict, on aura donc des instances où les solutions construites depuis le codage σ ne possèdent pas l'accessibilité à une solution optimale et où une solution optimale pourra être trouvée pour le codage σ, l.

3.5 Conclusion

Dans cette section nous avons proposé différentes méthodes d'optimisation : deux modèles mathématiques et des méthodes de résolution approchées. Les deux modèles mathématiques se différencient par le fait que l'un individualise les ressources et l'autre non. Le premier modèle se rapproche plus d'un modèle classique de RCPSP avec des contraintes cumulatives alors que le deuxième modèle est plus facilement adaptable à un ajout de contraintes sur les ressources (incompatibilités, compétences, ...). Les méthodes approchées proposées sont basées sur un couplage entre une méta-heuristique et un algorithme d'ordre strict. Trois codages de solution sont proposés :

- (σ, l, a) le codage le plus complet de la solution, intégrant le séquencement, l'affectation aux sites et l'affectation aux ressources
- (σ, l) qui se limite au séquencement et à l'affectation aux sites
- (σ) qui se limite au séquencement

Plus le codage est complet, plus l'espace de recherche des solutions va être grand. L'intérêt est donc souvent de chercher une solution dans un espace de recherche plus petit. Cependant nous avons montré que les codages σ, l et σ ne garantissent pas l'accessibilité à une solution optimale.

Dans le chapitre 5 nous étudions donc la pertinence des différents codages et nous comparons les deux modèles mathématiques proposés. Nous adaptons des instances de la littérature pour ce problème. Enfin nous présentons des résultats obtenus sur ces instances.

Dans le chapitre suivant, nous étudions des extensions possibles pour le RCPSP Multi-Site prenant en compte des contraintes sur les ressources.

Chapitre 4

Le Resource Constrained Project Scheduling Problem Multi-Site avec contraintes

Sommaire

4.1 Introduction

Dans le chapitre précédent nous avons proposé une extension du RCPSP qui permet de modéliser des problèmes d'ordonnancement et d'affectation dans un contexte multi-site. Cependant, nous avons considéré certaines hypothèses simplificatrices pour modéliser ces problèmes. C'est notamment le cas pour les incompatibilités entre les ressources humaines et les ressources matérielles. Nous proposons donc dans cette partie des extensions possibles de notre modèle du RCPSP Multi-Site où de nouvelles contraintes, présentes dans les problèmes originaux, sont prises en compte. On ajoute au problème du RCPSP multi-site les contraintes suivantes :

Les contraintes de non-incidence. Un couple de tâches peut posséder une contrainte de non-incidence lorsque que ces deux tâches ne doivent pas se chevaucher dans leur exécution et que l'une doit précéder l'autre, peu importe l'ordre. Un patient peut avoir plusieurs examens à passer, sans avoir un ordre de passage prédéfini.

L'indisponibilité des ressources. Les ressources ne sont pas disponibles sur tout l'horizon temporel. En cas de ressources humaines, ces ressources ont des horaires de travail et ne sont donc pas disponible sur tout l'horizon temporel pour exécuter des tâches.

Les compatibilités entre ressources et tâches. Il existe des incompatibilités entre les tâches et les ressources. Les compatibilités entre les tâches et ressources peuvent représenter une qualification spéciale d'une ressource pour exécuter une tâche, par exemple une ressource humaine spécialisée en pédiatrie.

Les compatibilités entre ressources. Pour le cas de compatibilités entre ressources, cela peut modéliser des compétences de ressources humaines pour des ressources matérielles.

Dans un premier temps nous présentons le problème proposé avec l'ensemble des nouvelles contraintes prises en compte. Nous proposons ensuite des modélisations mathématiques pour ces contraintes. Nous proposons aussi des adaptations des méthodes de résolution approchées.

4.2 Présentation du problème

4.2.1 Les contraintes ajoutées

Ce problème est une extension du RCPSP Multi-Site décrit dans la section 3.2.1. A ce problème, sont ajoutées de nouvelles contraintes :

— Chaque tâche j possède une liste Q_j de tâches qui ne peuvent pas s'exécuter en parallèle de la tâche j. Ces tâches doivent soit finir avant le début de l'exécution de j ou bien commencer après la fin de l'exécution de j. Ces contraintes de non-incidence peuvent, au même titre que celle de précédence, engendrer des temps de transport car elle concerne le transfert d'un produit semi-fini.

— Une ressource r de type k pourra être affectée à une tâche j uniquement si cette ressource est disponible $avb_{r,k,t} = 1$ pour toutes les périodes t pendant lesquelles la

tâche j s'exécute. Ces disponibilités correspondent à des horaires de travail pour les ressources humaines et des heures d'ouverture pour les ressources matérielles.

— Il existe des compatibilités $\lambda_{k_1,r_1,..,K_2,r_2}$ entre la ressource r_1 de type k_1 et la ressource r_2 de type k_2, qui permettent ou non que ces deux ressources soient associées à la même tâche. Ces compatibilités correspondent à des compétences entre ressources comme par exemple la capacité d'une ressource humaine à travailler sur une ressource matérielle ou permettent de modéliser le fait que deux ressources ne veuillent pas travailler ensemble.

— Il existe des compatibilités $\mu_{r,k,j}$ entre la ressource r de type k et la tâche j, cette ressource ne peut pas être affectée à cette tâche si elle n'est pas compatible. Ces compatibilités correspondent à des tâches que seule une partie des ressources d'un certain type est habilitée à effectuer.

Un temps de transfert entre deux tâches j et h devra être pris en compte dans deux cas :

— Si il existe une relation de précédence ou de non incidence entre j et h

— Si ces deux tâches partagent au moins une ressource

Dans les deux cas le temps de transfert entre la tâche j et h sera égal à la distance : $\delta(s, s')$ avec s le site d'exécution de la tâche j qui précède la tâche h exécutée sur le site s'. En cas de contrainte de non-incidence entre j et h, la tâche j ne pourra donc pas s'effectuer pendant la durée de la tâche h plus le temps de transfert et inversement.

4.2.2 Exemple d'instance

Reprenons l'instance présentée pour le RCPSP Multi-Site dans la partie 3.2.2. L'instance comporte 2 sites, 4 types de ressources et 9 tâches. Le temps de transport d'un site à l'autre est de deux périodes. Les tâches 1 et 9 sont les deux tâches fictives de début et fin de projet.

Aux contraintes de précédence, est ajoutée une contrainte de non-incidence (voir figure 4.1) entre les tâches 4 et 6. Chaque tâche est représentée par un cercle avec au-dessus sa durée et en-dessous les ressources nécessaires à son exécution sous la forme "type (quantité)".

Les caractéristiques des ressources, leurs compatibilités et leurs disponibilités sont données dans le tableau 4.1.

Ressource	Type	Statut	Site d'appartenance	Incompatibilités de ressources	Incompatibilités de tâches	Périodes d'indisponibilités
R1,1	1	Fixe	1	R2	6	
R1,2	1	Mobile	-	R4		
R2	2	Fixe	2	R1,1		
R3	3	Mobile	-			période 9 à 12 incluse
R4	4	Mobile	-	R1,1		

Tableau 4.1 – Liste des ressources disponibles

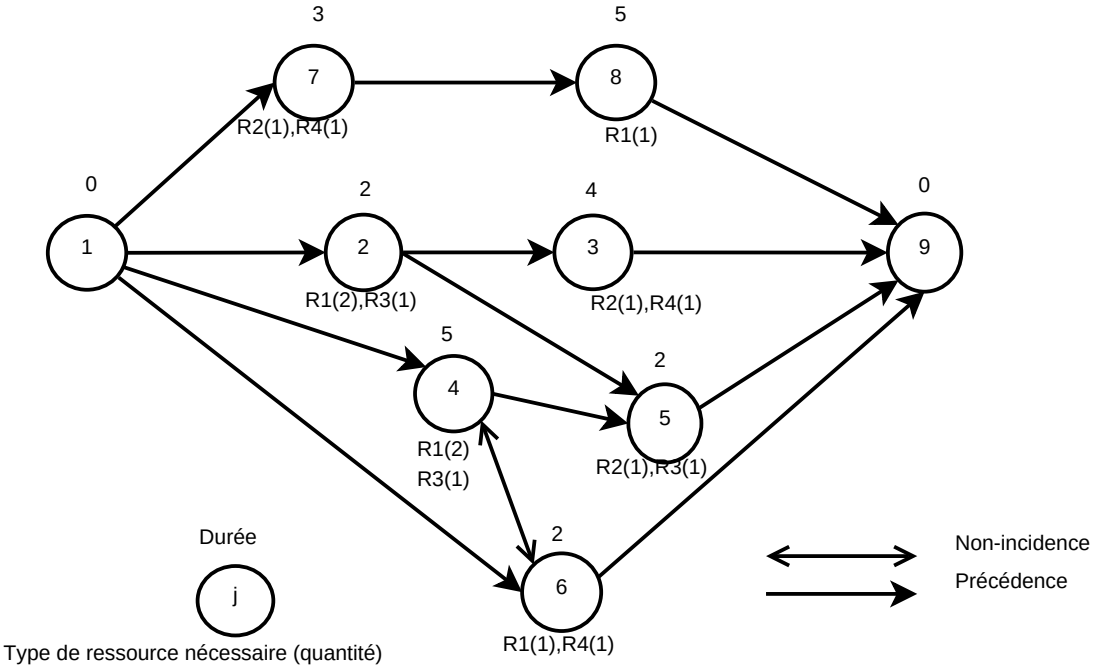

FIGURE 4.1 – Graphe de précédence et de non-incidence du problème

Une solution est donnée sur la figure 4.2. L'ajout des compatibilités et des disponibilités des ressources décale des dates d'exécution ou modifie des affectations. La tâche 6 est repoussée du fait qu'une contrainte de non-incidence existe avec la tâche 4. La tâche 6 s'exécute sur la ressource R1,2 du fait des incompatibilités entre cette tâche et la ressource R1,1 et de l'incompatibilité de la ressource R1,2 et R4. Enfin la tâche 5 est repoussée car la ressource R3 est indisponible de la période 9 à 12. Les ressources R1,2 et R3 se déplacent du site 1 au site 2 pour exécuter respectivement les tâches 6 et 5. On obtient alors une solution avec un makespan de 14 périodes.

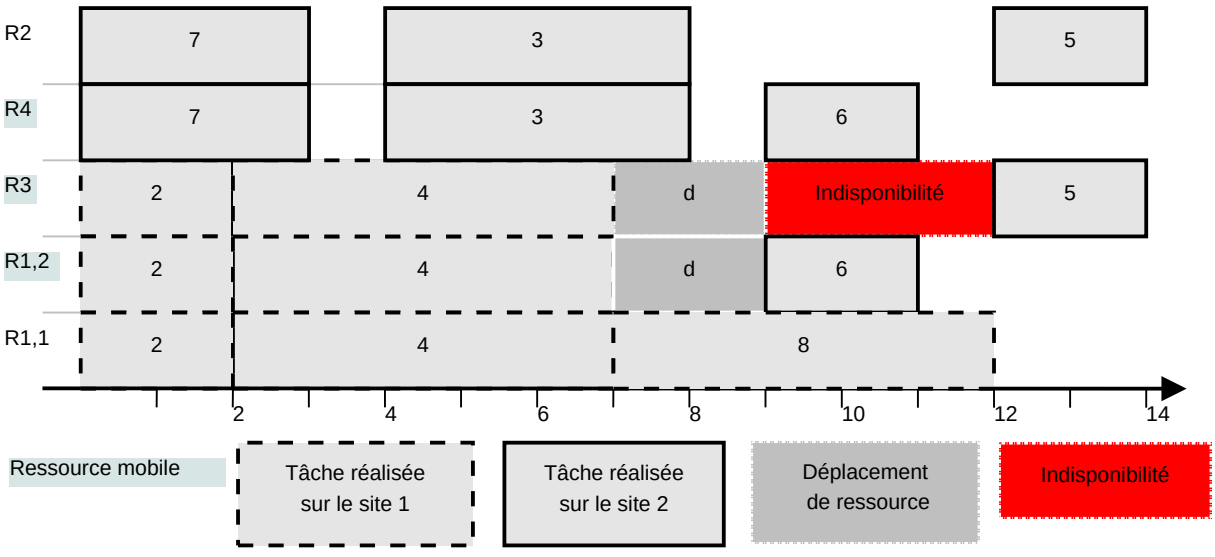

FIGURE 4.2 – Solution au problème

4.3 Modélisation mathématique

Nous proposons la formulation suivante, basée sur le modèle avec ressources individualisées proposé précédemment dans la section 3.3.1.

— **Données**

N Nombre de tâches

P_j Ensemble de tâches qui doivent précéder $j = 1, .., N$

Q_j Ensemble de tâches qui ne doivent pas être incidentes à $j = 1, .., N$

p_j Durée de la tâche $j = 1, .., N$

K Nombre de types de ressources

R_k Nombre de ressources de type $k = 1, .., K$

$r_{j,k}$ Nombre de ressources de type $k = 1, .., K$ nécessaires pour la tâche $j = 1, .., N$

T Nombre de périodes maximum

$M_{k,r}$ =1 si la ressource $r = 1, .., R_k$ de type $k = 1, .., K$ est mobile, 0 si elle est fixe

S Nombre de sites

$\delta_{s,s'}$ Temps de déplacement entre le site $s = 1, .., S$ et le site $s' = 1, .., S$;

$loc_{k,r}$ Site d'appartenance de la ressources $r = 1, .., R_k$ de type $k = 1, .., K$

$\lambda_{k_1,r_1,..,K_2,r_2}$ =1 si la ressource $r_1 = 1, R_{k_1}$ de type $k_1 = 1, .., K$ est compatible avec la ressource $r_2 = 1, R_{k_2}$ de type $k_2 = 1, .., K$, 0 sinon ;

$\mu_{j,k,r}$ =1 si la ressource $r = 1, .., R_k$ de type $k = 1, .., K$ est compatible avec la tâche $j = 1, .., N$, 0 sinon ;

$avb_{r,k,t}$ =1 si la ressource $r = 1, .., R_k$ de type $k = 1, .., K$ pour la période $t = 1, .., T$ $avb_{r,k,t} = 1$ est disponible, 0 sinon ;

— **Variables**

$X_{j,t}$ = 1 si la tâche $j = 1, .., N$ se termine à la période $t = 1, .., T$, 0 sinon

$Y_{j,k,r}$ =1 si la ressource $r = 1, .., R_k$ de type $k = 1, .., K$ est affectée à la tâche $j = 1, .., N$, 0 sinon

$Z_{j,s}$ =1 si la tâche $j = 1, .., N$ se déroule sur le site $s = 1, .., S$, 0 sinon

$\omega_{j,h}$ =1 si la tâche $j = 1, .., N$ précède la tâche $h = 1, .., N$ dans son exécution, 0 sinon

$$Minimiser \quad \sum_{t=1}^{T} t * X_{N,t} \tag{4.1}$$

$$\sum_{t=1}^{T} X_{j,t} = 1 \quad \forall j = 1, .., N; \tag{4.2}$$

$$\omega_{h,j} = 1; \quad \omega_{j,h} = 0; \quad \forall j = 1, .., N; \forall h \in P_j; \tag{4.3}$$

$$\omega_{j,h} + \omega_{h,j} = 1 \quad \forall j = 1, .., N; \forall h \in Q_j; \tag{4.4}$$

$$\sum_{t=1}^{T} t * X_{j,t} \geq \sum_{t=1}^{T} t * X_{h,t} + p_j + (Z_{j,s} + Z_{h,s'} - 1) * \delta(s,s') - M * (1 - \omega_{h,j});$$
$$(\forall j, h = 1,..,N)\ avec\ (j \neq h); \forall s, s' = 1,..,S; \quad (4.5)$$

$$\sum_{r=1}^{R_k} Y_{j,k,r} = r_{j,k}; \quad \forall j = 1,..,N; \forall k = 1,..,K; \quad (4.6)$$

$$Y_{j,k,r} + Y_{h,r,k} \leq \omega_{j,h} + \omega_{h,j} + 1;$$
$$(\forall j, h = 1,..,N)\ avec\ (j < h); \forall k = 1,..,K; \forall r = 1,..,R_k; \forall t = 1,..,T; \quad (4.7)$$

$$Y_{j,r_1,..,K_1} + Y_{j,r_2,k_2} \leq 1;$$
$$\forall j = 1,..,N; (\forall k_1,..,K_2 = 1,..,K)\ avec\ (k_1 \leq k_2); \forall r_1 = 1, R_{k_1}; \forall r_2 = 1, R_{k_2};\ avec\ \lambda_{k_1,r_1,..,K_2,r_2} = 0; \quad (4.8)$$

$$Y_{j,k,r} = 0; \quad (\forall j = 1,..,N; \forall k = 1,..,K; \forall r = 1,..,R_k)\ avec\ (\mu_{j,k,r} = 0); \quad (4.9)$$

$$\sum_{t_1=t}^{t+p_j-1} X_{j,t_1} + Y_{j,k,r} \leq 1;$$
$$\forall j = 1,..,N; (\forall k = 1,..,K; \forall r = 1,..,R_k; \forall t = 1,..,T;)\ avec\ (avb_{k,r,t} = 0;) \quad (4.10)$$

$$Y_{j,k,r} \leq Z_{j,loc_{k,r}}; \quad \forall j = 2,..,N; (\forall k = 1,..,K; \forall r = 1,..,R_k;)\ avec\ (M_{k,r} = 0); \quad (4.11)$$

$$\sum_{s=1}^{S} Z_{j,s} = 1; \quad \forall j = 1,..,N; \quad (4.12)$$

$$X_{j,t} \in \{0;1\} \quad \forall j = 1,..,N; \forall t = 1,..,T; \quad (4.13)$$
$$Y_{j,k,r} \in \{0;1\} \quad \forall j = 1,..,N; \forall k = 1,..,K; \forall r = 1,..,R_k; \quad (4.14)$$
$$Z_{j,s} \in \{0;1\} \quad \forall j = 1,..,N; \forall s = 1,..,S; \quad (4.15)$$
$$\omega_{j,h} \in \{0;1\} \quad \forall j, h = 1,..,N; \quad (4.16)$$

Le but est de minimiser le makespan (4.1). Seules les contraintes (4.6), (4.7) et (4.11) restent inchangées par rapport au modèle mathématique du RCPSP Multi-Site où les ressources sont individualisées. La non préemption et la réalisation des tâches sont assurées par la contrainte (4.2).

Les contraintes (4.3), (4.4), (4.5) et (4.7) permettent de modéliser qu'un temps de déplacement est à prendre en compte dans les deux cas suivants :

— Deux tâches sont reliées par une contrainte de précédence (contrainte (4.3)) ou de non-incidence (contrainte (4.4)).
— Une même ressource est affectée à deux tâches. Dans ce cas, les deux tâches ne peuvent pas avoir lieu en même temps (contrainte (4.7)).

Dans les deux cas, les temps de déplacement sont nuls si les deux tâches sont réalisées sur le même site. La contrainte (4.5) exprime le calcul des dates de fin des tâches en intégrant les temps de déplacement qui sont fonction des sites sur lesquels les tâches sont affectées.

Les incompatibilités entre les ressources sont modélisées par la contrainte (4.8), celles entre les tâches et les ressources par la contrainte (4.9). La contrainte (4.10) garantit qu'une ressource qui n'est pas disponible pendant une période ne sera pas affectée à une tâche pendant cette période.

La contrainte (4.6) assure que les quantités nécessaires de ressources sont affectées. Chaque tâche doit être exécutée sur le site d'appartenance des ressources fixes affectées (si des ressources fixes sont affectées) (4.11). Chaque tâche est exécutée sur un seul site (4.12). Les contraintes de bivalence sur les variables sont les contraintes (4.13), (4.14), (4.15) et (4.16).

Les nouvelles contraintes sont donc modélisées par les contraintes suivantes :

Les contraintes de non-incidence La non-incidence de deux tâches est assurée par la contrainte (4.4). Cette contrainte impose qu'une des deux tâches doit précéder l'autre.

La disponibilité des ressources Le respect de la disponibilités d'une ressource est assurée par la contrainte (4.10). Cette contrainte assure que pour toutes les tâches, si une ressource n'est pas disponible à une période, soit cette ressource n'est pas affectée à cette tâche, soit cette tâche ne s'exécute pas à cette période.

Les compatibilités entre ressources Les compatibilités entre ressources sont respectées par la contrainte (4.8). Cette contrainte assure que deux tâches incompatibles ne puissent pas être affectées à la même tâche.

Les compatibilités entre ressources et tâches Les compatibilités entre ressources et tâches sont assurées par la contrainte (4.9). Cette contrainte assure qu'une ressource n'est pas affectée à une tâche non-compatible.

4.4 Adaptation des méthodes de résolution du RCPSP multi-site

Les méthodes proposées pour résoudre le RCPSP multi-site ont été pensées pour être facilement adaptables pour résoudre des extensions du problème. Nous présentons ici des propositions d'adaptation des méthodes du RCPSP multi-site avec contraintes. Le principe du couplage présenté dans la section 3.4.2 reste le même. Dans cette partie, les contraintes ajoutées par rapport au problème de RCPSP multi-site sont considérées une à une. Pour chaque contrainte nous présentons les adaptations nécessaires au niveau de l'algorithme d'ordre strict et des systèmes de voisinage. L'intérêt de présenter les méthodes adaptées en fonction des contraintes, est de proposer une résolution modulable en fonction des contraintes prises en compte par une extension du RCPSP Multi-Site.

4.4.1 Les contraintes de non-incidence

La non-incidence est définie par :

— Q_j Ensemble de tâches qui ne doivent pas être réalisées en même temps que la tâche $j = 1, .., N$

Ces contraintes de non-incidence peuvent être vues comme des contraintes de précédence dont le sens n'est pas connu (i précède j ou j précède i). La prise en compte de ces contraintes impacte donc les algorithmes d'ordre strict des 3 codages. Soient les tâches i et j telles que $i \in Q_j$ (et $j \in Q_i$ par conséquent). L'ordre σ va déterminer si i précède j ou bien si c'est le contraire. Si i précède j dans σ et $i \in Q_j$, on notera $i \in QP_j$.

Ainsi seules les formules 3.32 de calcul de la date de fin d'exécution de la tâche, présentée dans la partie 3.4.3.2, vont changer. Les formules de calcul de la date d'exécution d'une tâche sur un site s deviennent les équations (4.17), (4.18) et (4.19).

$$d_j^s = \max(A, B) + p_j \tag{4.17}$$

$$A = \max(d_h + \delta(l_h, s); \forall h \in P_j \cup QP_j) \tag{4.18}$$

$$B = \max(av_{k,r}^s; \forall k, r | Y_{j,k,r} = 1) \tag{4.19}$$

$$\tag{4.20}$$

4.4.2 Les compatibilités entre ressources et tâches

Ces compatibilités autorisent ou interdisent l'affectation d'une ressource à une tâche. Ces compatibilités vont donc influencer les méthodes de résolution lorsque des affectations de ressources sont effectuées. C'est le cas pour :

— Les algorithmes d'ordre strict de σ et de σ, l

— Le mouvement V2 pour le calcul du nombre de ressources compatibles sur un site pour exécuter la tâche déplacée et pour la réaffectation des ressources après déplacement

— Le mouvement V3 où une ressources ne peut être remplacée que par une autre ressource compatible de même type.

Pour adapter ces méthodes, il suffit de ne considérer que les ressources compatibles avec la tâche considérée. Les ressources non-compatibles ne sont pas non plus considérées dans le décompte des ressources disponibles pour exécuter une tâche.

4.4.3 Les disponibilités des ressources

Ces disponibilités entraînent le fait qu'une ressource puisse ne pas être disponible pour exécuter une tâche pendant une ou plusieurs périodes. Ces disponibilités impactent fortement les algorithmes d'ordre strict dans leur fonctionnement. En effet, pour exécuter une tâche à une date donnée, toutes les ressources assignées doivent être disponibles pendant toute la période d'exécution de la tâche.

4.4.3.1 Algorithmes d'ordre strict de σ et σ, l

Pour les algorithmes d'ordre strict des codages σ et σ, l, il est nécessaire de pouvoir déterminer pour les tâches dans l'ordre σ quelles ressources vont pouvoir l'exécuter au plus tôt. Ces ressources doivent toutes être disponibles sur toute la durée d'exécution de la

tâche. On détermine donc quelles sont les r ressources qui sont disponibles le plus tôt sur le site s où est exécutée la tâche. La date d'exécution candidate est donc déterminée par l'équation 3.32 de la section 3.4.3.2. Cette date est dite candidate car il reste à déterminer si toutes les ressources sont disponibles sur toutes les périodes d'exécution de la tâche. Ont désignera de manière analogue ces ressources candidates. Si toutes les ressources sont disponibles sur toute la durée d'exécution, alors la date candidate devient la date d'exécution définitive de la tâche. Si ce n'est pas le cas, toutes les ressources ayant une période ou plus d'indisponibilité, entre la date de début candidate et la date de fin candidate, ont leur date de disponibilité réévaluée. Une fois ces dates de disponibilité réévaluées, on détermine la nouvelle date candidate comme fait précédemment en sélectionnant les ressources disponibles au plus tôt. On réitère ce procédé jusqu'à ce qu'une date candidate, où toutes les ressources candidates sont disponibles, soit trouvée. Les ressources et la date d'exécution de la tâche j sont déterminées selon l'algorithme 10. A noter que pour le codage σ tous les sites devront être comparés afin de sélectionner celui qui peut exécuter la tâche le plus tôt. Alors que pour le codage σ, l, le site s est une donnée.

4.4.3.2 L'algorithme d'ordre strict pour le codage σ, l, a

Les indisponibilités de ressources font que l'algorithme d'ordre strict du codage σ, l, a doit être adapté. Les indisponibilités des ressources peuvent repousser la date d'exécution au plus tôt d'une tâche. En effet toutes les ressources doivent être disponibles pendant toute la durée d'exécution d'une tâche j. L'algorithme 11 détermine pour une tâche j, affectée à un ensemble de ressources a_j défini par la matrice a, sa date d'exécution au plus tôt.

4.4.4 Les compatibilités entre ressources

Ces incompatibilités vont impacter la résolution à plusieurs niveaux, l'algorithme d'ordre strict des codages σ et σ, l et les voisinages V2 et V3.

4.4.4.1 Les systèmes de voisinage de σ, l, a

Les deux mouvements V2 et V3 pour le codage σ, l, a sont concernés par les incompatibilités des ressources. Le mouvement $V2$ garde son algorithme de principe (algorithme 8). Le principe reste de modifier le site d'exécution d'une tâche. Ensuite l'algorithme doit remplacer les ressources fixes localisées sur l'ancien site d'exécution de la tâche par d'autres ressources. Pour cela on sélectionne une ressource compatible aléatoirement et on l'affecte à la tâche. Les ressources compatibles doivent toujours être mobiles ou fixes sur le nouveau site d'exécution de la tâche. La différence avec le nouveau problème est que ces ressources sont compatibles si elle ne possède pas d'incompatibilité avec une des ressources déjà affectées à la tâche.

Remarque : On ne peut pas être sûr de trouver une affectation de ressources sur un site donné. Pour déterminer si une affectation existe, un problème NP-difficile doit être résolu. Nous présentons ce problème dans la section suivante (voir section 4.4.4.2). Ainsi le mouvement V2 peut conduire à une solution non-réalisable.

Le principe du mouvement V3 est de remplacer l'affectation à une tâche j d'une ressource r par une autre r'. Ainsi la nouvelle ressource ne sera pas compatible si :

Algorithme 10 : Algorithme d'affectation des ressources et de calcul de la date d'exécution d'une tâche pour le codage σ, l et σ

Entrées : av : Vecteur des dates de disponibilité des ressources ;
Variables : d_j : Date candidate pour l'exécution de la tâche j ;
Er : Ensemble de ressources pour exécuter la tâche j
Na : Ensemble de ressources indisponibles pour la date candidate j
Initialisation : $Stop \leftarrow Faux$;

```
1  Début
2      Tant que Stop=Faux faire
3          Stop←Vrai;
4          Na ← ∅;
5          Er ← les ressources disponibles au plus tôt en fonction de s et av;
6          d_j ← La date d'exécution au plus tôt avec les ressources Er(3.32);
7          Pour t de d_j à d_j + p_j faire
8              Pour Toutes les ressources r ∈ Er\Na faire
9                  Si La ressource r n'est pas disponible à la période t alors
10                     Stop←Faux;
11                     On ajoute r à Na;
12                 Finsi
13             Finpour
14         Finpour
15         Pour Toutes les ressources r de Na faire
16             dav_r ←la période la plus tôt telle que la ressource r n'est pas
                disponible;
17             av_r ← la période la plus tôt, supérieure à dav_r telle que la ressource r
                est disponible;
18         Finpour
19     Fintq
20     Mise à jour des dates de disponibilité des ressources Er;
21     Retourner d_j
22 Fin
```

Algorithme 11 : Algorithme d'affectation des ressources et de calcul de la date d'exécution d'une tâche pour le codage σ, l, a

Entrées : av : Vecteur des dates de disponibilité des ressources ;
a_j : Ensemble des ressources qui exécutent la tâche j
Variables : d_j : Date candidate pour l'exécution de la tâche j ;
Na : Ensemble de ressources indisponibles pour la date candidate j
Initialisation : $Stop \leftarrow Faux$;

```
1  Début
2      Tant que Stop=Faux faire
3          Stop←Vrai;
4          Na ← ∅;
5          d_j ← La date d'exécution au plus tôt avec les ressources a_j(3.32);
6          Pour t de d_j à d_j + p_j faire
7              Pour Toutes les ressources r ∈ Er\Na faire
8                  Si La ressource r n'est pas disponible à la période t alors
9                      Stop←Faux;
10                     On ajoute r à Na;
11                 Finsi
12             Finpour
13         Finpour
14         Pour Toutes les ressources r de Na faire
15             dav_r ←la période la plus tôt telle que la ressource r n'est pas
                disponible;
16             av_r ← la période la plus tôt, supérieure à dav_r telle que la ressource r
                est disponible;
17         Finpour
18     Fintq
19     Mise à jour des dates de disponibilité des ressources a_j;
20     Retourner d_j
21 Fin
```

— La nouvelle ressource r' appartient déjà à a_j
— La nouvelle ressource r' n'est pas du même type que $a_{j,r}$
— La nouvelle ressource r' est fixe et son site d'appartenance $loc_{k,r'}$ est différent du site l_j
— La nouvelle ressource r' est incompatible avec une ressource qui appartient à a_j

4.4.4.2 Le problème d'affectation des ressources aux tâches

Les algorithmes d'ordre strict correspondant aux codages σ et σ, l doivent nécessairement déterminer quelles ressources vont exécuter chaque tâche j. Ces ressources possèdent des incompatibilités entre elles et ne peuvent pas être affectées à une même tâche. Pour exécuter une tâche au plus tôt, l'algorithme doit donc déterminer quel ensemble de R_j ressources sans incompatibilité, minimise la date de fin de la tâche j. Dans le cas où **un seul type** de ressource est utilisé, on représente l'ensemble des ressources disponibles pour exécuter une tâche, sur un site s donné, par un graphe comme sur la figure 4.3. Chaque sommet représente une ressource avec pour poids W_r la date de disponibilité de la ressource r sur le site s. On considère un arc entre les sommets i, j si les deux ressources i et j sont incompatibles ($\lambda_{1,i,1,j} = 0$).

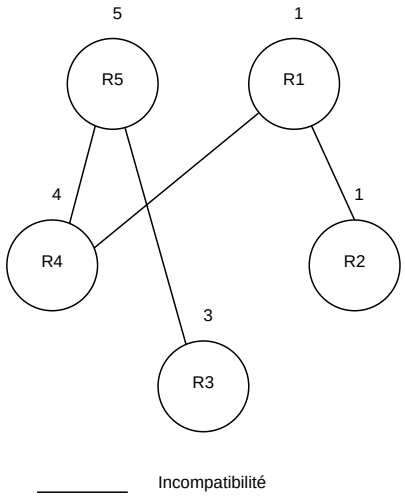

FIGURE 4.3 – Graphe d'incompatibilité des ressources pour exécuter la tâche j

Si on considère le graphe complémentaire du graphe d'incompatibilité on obtient un graphe de compatibilité entre les ressources (figure 4.4).

Trouver un ensemble de R_j ressources compatibles revient donc à trouver une clique c de taille R_j dans le graphe de compatibilité qui minimise la date de début d'exécution de la tâche. Ce problème équivaut à trouver un stable, dans le graphe d'incompatibilité, qui minimise la date de début de la tâche. Trouver l'ensemble c de ressources qui minimise la date de début d'exécution de la tâche j revient à résoudre l'équation 4.21.

$$\min(\max(W_r)\forall r \in c) \tag{4.21}$$

Ce problème est NP-difficile. En effet, trouver une clique de taille k dans un graphe quelconque est un problème NP-difficile (Karp, 1972).

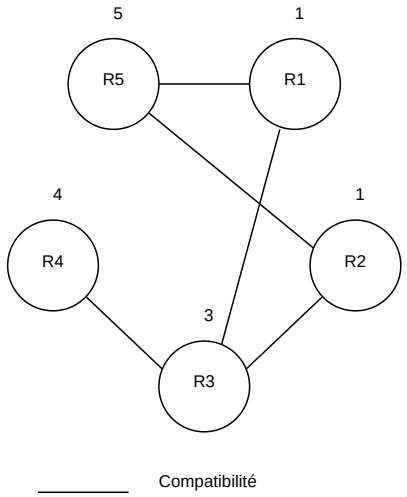

FIGURE 4.4 – Graphe de compatibilité des ressources pour exécuter la tâche j

Un problème de la littérature s'approche de cette problématique. Il s'agit du problème de clique de poids maximum (Garey and Johnson, 1979). Ce problème a pour but de trouver une clique de taille k de poids maximum. Le poids d'une clique est égale à la somme des poids des sommets de la clique, ce qui le différencie du problème d'affectation de ressources.

Une autre différence majeure avec le problème de clique de poids maximum est dans le cas où plusieurs types de ressources sont considérés. Dans ce cas là, une contrainte supplémentaire s'ajoute au problème. Les sommets du graphe de compatibilité sont répartis en plusieurs ensembles, en fonction du type de la ressource qu'ils représentent (figure 4.5). La clique c à trouver est de taille R_j, avec R_j la somme des besoins $r_{j,k}$ pour chaque type k de ressources. La particularité de ce problème est que la clique de taille R_j doit être composée exactement de $r_{j,k}$ sommets de l'ensemble k.

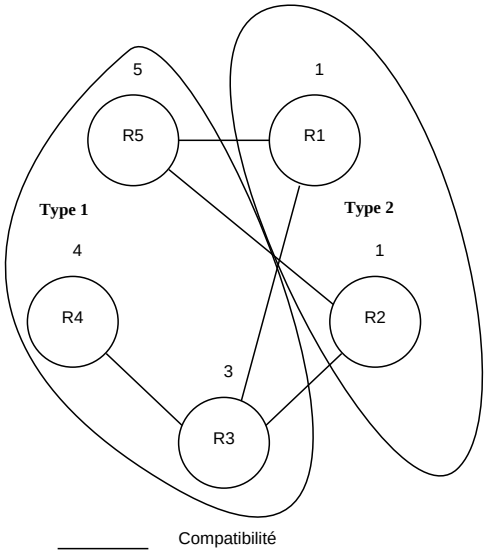

FIGURE 4.5 – Graphe de compatibilité avec plusieurs types de ressources

Pour ordonnancer une tâche au plus tôt on doit donc résoudre un problème NP-

difficile. Pour construire une solution à partir des codages σ et σ, l l'algorithme d'ordre strict doit résoudre respectivement $N * S$ et N problèmes NP-difficile. L'utilisation d'une méthode approchée rapide est donc nécessaire. Cependant ces méthodes auront forcement un défaut, comme le problème d'existence d'une solution est NP-difficile, une heuristique gloutonne ne pourra pas donner à coup sûr une solution. L'algorithme d'ordre strict devra donc prendre en compte le fait que dans certains cas l'heuristique ne trouve pas d'affectation de ressource réalisable. Dans ce cas deux choix sont possibles : soit exécuter une méthode exacte si la taille du problème le permet, soit considérer une affectation ne respectant pas les incompatibilités entre ressources. Dans le cas d'une affectation des ressources non-réalisable, on peut considérer dans la fonction objectif un second critère : le nombre d'affectation de ressources non réalisable (c'est à dire où des ressources non-compatibles exécutent une même tâche).

Nous ne proposons pas directement de méthodes de résolution pour cette extension. L'étude de ce problème demande de prendre en compte des solutions non-réalisables et dans ce cas, de modifier l'approche de résolution afin de considérer ces solutions irréalisables. L'autre choix est d'utiliser des méthodes de résolution exactes, pouvant ne pas être adaptées en fonction de la taille du problème. Cette extension demande donc une étude à part afin de comparer ces méthodes.

4.5 Conclusion

Dans cette partie nous avons présenté des extensions possibles au RCPSP Multi-Site se rapportant au problème de mutualisation de ressources présenté dans le chapitre 1. Ces extensions sont les suivantes :

— Les contraintes de non-incidence entre les tâches
— Les contraintes de disponibilité des ressources
— Les contraintes de compatibilité entre tâche et ressources
— Les contraintes de compatibilité entre ressources

Nous avons donné un modèle mathématique modélisant l'ensemble des contraintes ajoutées par l'extension. Puis dans un second temps nous avons montré comment adapter les méthodes approchées du RCPSP Multi-Site, pour la résolution des extensions en présentant les contraintes une à une. Le but est de proposer une résolution modulable en fonction des caractéristiques prises en compte par l'extension du RCPSP Multi-Site.

Cette section fait office de proposition de méthodes de résolution et de perspectives de travail. Nous ne présenterons pas d'expérimentations sur ces extensions dans ce manuscrit. Ces expérimentations feront l'objet de futurs travaux.

Chapitre 5

Mise en œuvre et résultats

Sommaire

5.1 Introduction

Le RCPSP Multi-Site est une extension directe du RCPSP, qui est un problème sur lequel beaucoup de travaux portent. Plusieurs bibliothèques d'instances du RCPSP existent, ce qui permet de comparer les résultats obtenus par différentes méthodes. Nous présenterons nos résultats en quatre parties. La première partie présente les résultats obtenus avec la résolution par méta-heuristique sur les instances du RCPSP classique. Dans la seconde section, nous présentons les instances que nous utiliserons pour nos expérimentations. Deux ensembles d'instances sont présentés : des instances de petite taille pour mesurer les performances des modèles mathématiques proposés et des instances adaptées de la littérature pour tester les méthodes approchées sur de plus grandes instances hors d'atteinte pour les méthodes exactes. Dans la troisième partie, nous donnons les résultats obtenus par les modèles mathématiques en les comparant. La quatrième partie présente les résultats obtenus par les méthodes approchées. Les trois codages sont mis en œuvre dans un premier temps sur les instances de petite taille et les résultats obtenus sont comparés avec ceux obtenus par les modèles mathématiques. Dans un second temps, les différents codages sont appliqués sur les instances adaptées de la littérature afin d'étudier les résultats obtenus sur de plus grandes instances. Enfin les codages sont comparés en fonction du temps, étant donné que les codages sont plus ou moins rapides pour construire une solution à partir des vecteurs σ, l et a.

5.2 Résolution du SM-RCPSP classique

Le problème du RCPSP Multi-Site est avant tout une extension du SM-RCPSP classique. La caractéristique d'une extension est de généraliser des problèmes que la version classique ne peut modéliser. Cela implique que l'extension puisse modéliser les cas particuliers représentés par le modèle classique.

La première étape pour valider les méthodes proposées pour le RCPSP Multi-Site est donc de les tester sur des instances du RCPSP. En effet, les instances du RCPSP correspondent à des instances du RCPSP Multi-Site avec un seul site. Pour ce faire, nous utilisons des instances de la PSPLIB, une bibliothèque d'instances utilisée dans la littérature pour comparer les performances des différentes méthodes de résolution. L'objectif est de comparer les résultats obtenus par nos méthodes approchées, avec les solutions optimales ou avec les meilleures solutions connues, données par la littérature. Cependant, les méthodes proposées ne sont pas conçues spécifiquement pour ce problème, l'objectif est de valider le comportement de nos méthodes, pas de rechercher la performance sur cette classe d'instances.

5.2.1 La bibliothèque d'instances de la PSPLIB

Pour créer cette bibliothèque (PSPLIB) les auteurs (Kolisch and Sprecher, 1997) ont utilisé le générateur ProGen. Chaque classe d'instances correspond à la variation de trois paramètres pour les créer :

Network Complexity (NC) correspond au nombre moyen de contraintes de précédence liées à une tâche, en prenant en compte les tâches fictives. Les créateurs de la bibliothèque ont généré des instances en fixant ce paramètre à

1.50, 1.80 et 2.10.

Resource Factor (RF) correspond à la quantité de ressources utilisables par une tâche pour un type de ressources. L'indice correspond à la proportion utilisable par rapport à la quantité disponible. Les instances sont générées avec les valeurs 0.25, 0.50, 0.75, 1.

Resource Strength (RS) correspond à la quantité disponible pour chaque type de ressources. Ce paramètre va déterminer si un type de ressources sera disponible en grande ou en petite quantité. Le paramètre RS détermine la proportion de ressources disponibles par rapport à un maximum fixé. Les auteurs génèrent des instances en utilisant les valeurs 0.20, 0.50, 0.70 et 1.00 pour RS.

Plusieurs ensembles d'instances sont créées en fonction du nombre de tâches à exécuter. En tout 4 ensembles d'instances sont créés, de 30, 60, 90 et 120 tâches. Chaque combinaisons de paramètres donne naissance à une classe d'instance. Il y a en tout $3 * 4 * 4 = 48$ combinaisons de paramètres NC, RF et RS, soit 48 classes d'instances par bibliothèque d'instances d'une même taille. Chaque classe est composée de 10 instances distinctes, soit 480 instances de 30, 60, 90 et 120 tâches.

5.2.2 Expérimentations

5.2.2.1 Protocole expérimental

Les méthodes approchées sont testées sur les instances du RCPSP classique. Les trois codages proposés n'apportent pas tous un intérêt dans la résolution du RCPSP classique. En effet, le vecteur l qui modélise les sites d'affectation des tâches n'apporte pas d'intérêt dans le cas où un seul site est considéré. Pour ce qui est de la matrice a d'affectation des ressources aux tâches, son intérêt est motivé par le fait que les autres codages couplés aux algorithmes d'ordre strict ne respectent pas l'accessibilité à une solution optimale. Or dans le cas du RCPSP, le codage σ respecte cette accessibilité. La matrice a n'apporte donc pas de véritable intérêt dans la résolution du RCPSP. Un seul codage sera donc testé sur cette bibliothèque d'instances, le codage σ. Ce codage est testé avec trois méta-heuristiques : la recherche locale, le recuit simulé et la recherche locale itérée. Pour la recherche locale itérée, deux critères d'acceptation sont évalués, celui de la recherche locale (Better Walk) et celui du recuit simulé (Simulated Annealing), soit 4 méthodes en tout.

Les paramètres utilisés sont les suivants :
- Nombre d'itérations total : 100 000
- Nombre de réplications par méthodes : 20
- Voisinage : V1
- Recuit simulé :
  - Température finale : 0.01
- Recherche Locale Itérée
  - Palier : 5000

5.2.2.2 Résultats obtenus

Nous utiliserons les notations suivantes dans cette section :

— Recherche locale (LS)
— Recuit simulé (SA)
— Recherche Locale Itérée avec le critère de la Recherche Locale ($ILS|LS$)
— Recherche Locale Itérée avec le critère du recuit simulé ($ILS|SA$)
— L'ensemble des méthodes $M = \{LS; SA; ILS|LS; ILS|SA\}$
— $X_{i,j}^m$ la valeur du makespan obtenu par la réplication $j = 1, 20$ de la méthode $m \in M$ sur l'instance $i = 1, 480$.
— BKS_i la meilleure solution connue pour l'instance $i = 1, 480$
— $AvRG_i^m$ est la moyenne des écarts relatifs entre le makespan obtenu par la méthode $m \in M$ et la meilleure solution connue BKS_i pour l'instance $i = 1, 480$. Le calcul de $AvRG_i^m$ est donné par l'équation 5.1.

$$AvRG_i^m = \frac{\sum_{j=1}^{20} \frac{X_{i,j}^m - BKS_i}{BKS_i}}{20} \tag{5.1}$$

— $AvRGC_c^m$ est la moyenne des écarts moyens pour la méthode $m \in M$ et les instances de la classe $c = 1, 48$. Le calcul du $AvRGC_c^m$ est assuré par l'équation 5.2.

$$AvRGC_c^m = \frac{\sum_{i \in c} AvRG_i^m}{10} \tag{5.2}$$

— $AvRG^m$ est la moyenne des écarts relatifs moyens sur toutes les instances et par la méthode $m \in M$. Le calcul du $AvRG^m$ est donné par l'équation 5.3.

$$AvRG^m = \frac{\sum_{i=1}^{480} AvRG_i^m}{480} = \frac{\sum_{c=1}^{48} AvRGC_c^m}{48} \tag{5.3}$$

— NBC^m est le nombre de classes pour lesquelles la méthode $m \in M$ donne en moyenne les meilleurs résultats. Le calcul est donné par l'équation 5.4.

$$NBC^m = \sum_{c=1}^{48} (1_{AvRGC_c^m = Min_{m' \in M}(AvRGC_c^{m'})}) \tag{5.4}$$

Tableau 5.1 – Résultats obtenus sur les instances du RCPSP de 30 tâches

Méta-heuristiques	LS	ILS|LS	ILS|SA	SA
Min $AvRGC_c^m$	**0,00**	**0,00**	**0,00**	**0,00**
$AvRG^m$	0,56	0,14	**0,12**	0,17
Max $AvRGC_c^m$	10,77	7,89	3,90	5,26
Nombre de classes où $AvRGC_c^m = 0$	18	30	30	31
% d'instances telles que $AvRG_i^m = 0$	81,88%	92,92%	**93,54%**	92,08%
NBC^m	26	**40**	39	37

Les résultats obtenus sur les instances de 30, 60, 90 et 120 tâches sont donnés respectivement dans les tableaux 5.1, 5.2, 5.3 et 5.4. À noter que pour les instances du RCPSP de la PSPLIB de 30 tâches, toutes les solutions optimales sont connues. Ce n'est le cas pour les instances de taille 60 et plus.

Les résultats obtenus montrent que les méthodes proposées donnent de bonnes solutions pour les instances du problème classique. La méthode qui donne les meilleurs

TABLEAU 5.2 – Résultats obtenus sur les instances du RCPSP de 60 tâches

Méta-heuristiques	LS	ILS|LS	ILS|SA	SA
Min $AvRGC_c^m$	**0,00**	**0,00**	**0,00**	**0,00**
$AvRG^m$	1,11	**0,84**	0,91	0,89
Max $AvRGC_c^m$	10,09	7,22	**6,90**	7,06
Nombre de classes où $AvRGC_c^m = 0$	24	24	**26**	24
% d'instances telles que $AvRG_i^m = 0$	70,21%	70,83%	70,21%	**71,04%**
NBC^m	28	**36**	28	32

TABLEAU 5.3 – Résultats obtenus sur les instances du RCPSP de 90 tâches

Méta-heuristiques	LS	ILS|LS	ILS|SA	SA
Min $AvRGC_c^m$	**0,00**	**0,00**	**0,00**	**0,00**
$AvRG^m$	1,61	1,58	1,55	**1,39**
Max $AvRGC_c^m$	14,29	11,82	10,34	**9,30**
Nombre de classes où $AvRGC_c^m = 0$	25	**26**	24	**26**
% d'instances telles que $AvRG_i^m = 0$	70,42%	**70,63%**	70,00%	70,42%
NBC^m	30	29	28	**38**

TABLEAU 5.4 – Résultats obtenus sur les instances du RCPSP de 120 tâches

Méta-heuristiques	LS	ILS|LS	ILS|SA	SA
Min $AvRGC_c^m$	**0,00**	**0,00**	**0,00**	**0,00**
$AvRG^m$	4,64	5,11	5,11	**4,25**
Max $AvRGC_c^m$	15,30	14,07	14,89	12,94
Nombre de classes où $AvRGC_c^m = 0$	3	3	3	3
% d'instances telles que $AvRG_i^m = 0$	17,29%	16,25%	16,25%	**17,50%**
NBC^m	15	9	6	21

résultats, avec ce protocole, est la recherche locale itérée utilisant le critère d'acceptation de la recherche locale. En moyenne l'écart relatif entre les meilleures solutions connues dans la littérature et les solutions données par les méthodes approchées est inférieure à 1% pour les instances de 30 et 60 tâches. Pour les instances de taille 90, ce ratio est inférieur à 2%. Plus de 70% des instances les meilleures solutions connues de la littérature sont retrouvées pour les instances de 90 tâches et moins. Pour les instances de 120 tâches les résultats sont moins bons. Ceci est dû au fait que le nombre d'itérations est trop petit pour résoudre efficacement cet ensemble d'instance.

Nous pouvons en conclure que le codage σ et le mouvement V1 sont efficace.

5.3 Création d'une bibliothèque d'instances

Pour tester nos méthodes de résolution nous avons créé des instances pour le RCPSP Multi-Site que nous proposons. Nous avons utilisé la bibliothèque d'instances de la PSPLIB pour créer ces instances. Cela a plusieurs avantages :

- Conserver les propriétés de construction des instances de la littérature
- Pouvoir comparer les résultats obtenus avec les solutions optimales et bornes inférieures connues de la littérature. En effet puisque le problème de RCPSP Multi-Site est une extension du RCPSP, la solution optimale d'une instance du RCPSP est une borne inférieure pour l'instance correspondante du RCPSP Multi-Site.

Cependant ces instances ont un défaut, leur taille (nombre de tâches) va de 30 tâches à 120 tâches. Or si nous voulons tester nos modèles mathématiques sur des instances du RCPSP Multi-Site, le temps de résolution, des instances de taille moyenne et grande, risque d'être trop conséquent. Nous avons aussi construit une bibliothèque d'instances de plus petite taille, allant de 10 à 30 tâches.

5.3.1 Instances de petite taille

Pour générer les instances de petite taille, nous faisons varier trois paramètres : le nombre de tâches, le nombre de ressources et le nombre de sites. L'objectif de ces instances est d'étudier le comportement des méthodes de résolution sur de petites instances.

Les instances sont répertoriées selon différentes classes dépendant de ces paramètres. Les valeurs que prennent ces paramètres sont les suivantes :

- nombre de ressources ($\sum_{k=1}^{K} R_k$) : 10 ou 20 (quel que soit leur type)
- nombre de sites (S) : 2 ou 3
- nombre de tâches (N) : 10, 15, 20, 25 ou 30

Pour chaque combinaison de paramètres, nous avons généré aléatoirement huit instances ce qui fait un total de 160 instances.

Pour toutes les instances, les règles de génération suivantes ont été adoptées :

- La durée des tâches est comprise entre 1 et 10 périodes.
- Les sites sont éloignés d'une durée (pas forcement symétrique) comprise entre 1 et 10 périodes.
- On dispose de 4 types de ressources (K=4).

— Une tâche nécessite pour chaque type de ressources, un nombre de ressources choisis aléatoirement entre 1 et le nombre de ressources disponibles de ce type.
— Pour chaque couple de tâches, la probabilité est de 5% qu'elles soient liées par une relation de précédence. Ainsi pour une instance de taille 30, la probabilité qu'une tâche possède au moins une relation de précédence, avec une tâche non fictive, est d'environ 75%
— La probabilité qu'une ressource soit fixe est de 50%.

5.3.2 Adaptation des instances de la PSPLIB

Pour adapter des instances, nous nous basons sur la bibliothèque d'instances de la PSPLIB proposé par (Kolisch and Sprecher, 1997). Nous devons ajouter plusieurs choses, la première est de définir un nombre de sites, ainsi que la distance qui les sépare (cette distance n'est pas forcement symétrique). Il faut aussi déterminer quelles ressources seront fixes et à quels sites elles appartiendront.

Pour toutes les instances, les règles de génération suivantes ont été adoptées :
— Nombre de sites : fixé arbitrairement à 3 sites.
— Distances entre les sites : durée aléatoire entre 1 et 10 périodes.
— Mobilité des ressources : la probabilité qu'une ressource soit fixe est de 50%.
— Répartition des ressources fixes : chaque ressource fixe est affectée aléatoirement à un site.

Pour que les solutions possèdent au moins une solution réalisable il faut que chaque tâche puisse être exécutée sur un site. Pour cela il doit exister un site où le nombre de ressources disponibles (ressources fixes + mobiles) est supérieur ou égal au nombre de ressources nécessaires pour exécuter la tâche, pour chaque type. Si cette condition n'est pas vérifiée après la phase de génération aléatoire des mobilités des ressources, on sélectionne une ressource fixe et on la rend mobile pour un type de ressources bloquant. On répète l'opération jusqu'à ce que l'instance puisse être résolue. Ces instances sont disponibles sur $http://www.isima.fr/\sim laurenta/RCPSPMS.html$. La meilleure solution connue pour chaque instance est aussi fournie sur ce même site.

5.4 Comparaison des modèles mathématiques

La différence entre les modèles mathématiques proposés se situe sur l'individualisation ou non des R ressources. Le Modèle Individualisé (MI) utilise des variables qui fixent la date de fin des tâches ($N * T$ variables), l'affectation des ressources aux tâches ($N * R$ variables), l'affectation des tâches aux sites ($N*S$ variables) et les variables qui détermine si un temps de transfert doit être appliqué ($N * N - 1$ variables). Le nombre total de variables est donc égal à $N * (T + S + R + N - 1)$. Le nombre de contraintes est de $N * (K + R + 2) + N^2 * (R + S^2)$.

Le Modèle à Contraintes Cumulatives (MCC) les variables qui fixe la date de fin des tâches ($N * T$ variables), l'affectation des tâches aux sites ($N * S$ variables), le transfert de ressources entre les sites ($S * (S - 1) * K * T$), le nombre de ressources disponibles sur chaque site à une période donnée ($K * S * T$) et les variables qui déterminent si une tâche est affectée à une période pour un site donné ($S * T * N$). Le nombre de contraintes est de $N * (2 + S^2 + S * T) + K * (1 + T * S * (1 + S))$.

Si on prend un exemple d'instance de 10 tâches avec 10 ressources de 4 types différents réparties sur 3 sites. On considère un horizon de planification de 50 périodes.

— Pour le MI on obtient un modèle avec 720 variables et 2060 contraintes

— Pour le MCC on obtient un modèle avec 3430 variables et 6930 contraintes

On constate que le nombre de variables et de contraintes pour le MCC est supérieur à celui du MI pour cette instance. Cependant, le nombre de contraintes et de variables du MCC n'est pas influencé par le nombre de ressources alors que le MI l'est. Nous présentons les résultats obtenus en 60 minutes avec les deux modèles dans le tableau 5.5. Gap_i est le rapport entre la borne inférieure et la borne supérieure données par le modèle mathématique pour une instance i. Ce calcul est donné par l'équation :

$$Gap_i = \frac{Borne\ supérieure_i - Borne\ inférieure_i}{Borne\ inférieure_i} \tag{5.5}$$

Pour chaque ensemble d'instances, on calcule la moyenne Gap^e des Gap_i pour toutes les instances i appartenant à un ensemble e. Le calcul de Gap^e est donné par l'équation :

$$Gap^e = \frac{\sum_{i \in e} Gap_i}{|e|} \tag{5.6}$$

Pour les instances de 30 tâches et plus, le solveur ne retourne pas de solution réalisable en 60 minutes pour 28% des instances pour le MI et pour 19% des instances pour le MCC. De plus, la plupart du temps, les écarts entre les bornes inférieures et supérieures sont trop grands pour que les solutions trouvées aient un intérêt. Nous ne présentons donc pas les résultats obtenus pour les instances de 30 tâches et celles adaptées de la PSPLIB. Nous présentons dans le tableau 5.5 le % Résolu, c'est à dire le pourcentage d'instances pour lesquelles une solution optimale a été trouvée dans les 60 minutes d'exécution maximum. On reporte aussi le nombre d'instances NBC_e pour lesquelles le modèle m donne les meilleures solutions pour l'ensemble e.

	Modèle individualisé			Modèle à Contraintes Cumulatives		
Instances e	MI% Résolu	Gap^e	NBC_e	MCC % Résolu	Gap^e	NBC_e
N=10	**100%**	**0**	**32/32**	**100%**	**0**	**32/32**
N=15	81%	3%	**29/32**	**94%**	**2%**	**29/32**
N=20	16%	**29%**	23/32	**44%**	34%	**25/32**
N=25	13%	**59%**	15/32	**16%**	64%	**20/32**
R=10	44%	28%	54/64	**59%**	**15%**	**56/64**
R=20	42%	**28%**	55/64	**44%**	41%	**60/64**
S=2	43%	27%	46/64	**53%**	**22%**	**60/64**
S=3	43%	**28%**	**63/64**	**50%**	34%	56/64
Total	43%	**28%**	109/128	**51%**	**28%**	**116/128**

Tableau 5.5 – Résultats obtenus par les deux modèles avec CPLEX sur 60 minutes

Les expérimentations permettent de mettre en évidence plusieurs résultats :

— Les deux modèles donnent des résultats assez proches

— Le modèle à contraintes cumulatives trouve plus souvent une solution optimale que le modèle à ressources individualisées

— Le modèle à ressources individualisées est moins impacté par l'augmentation du nombre de sites ou de ressources

Pour les instances de 10 tâches et 15 tâches, la quasi-totalité des instances est résolu en 60 minutes. Au delà, une faible partie seulement des instances est résolue dans le temps imparti. Pour les instances dont le modèle n'est pas résolu, une borne supérieure et inférieure sont données pour l'instance. Le gap entre la borne inférieure et supérieur donne une information sur la qualité de la solution donnée par le solveur. On constate que pour les instances de 20 tâches et plus, l'écart moyen obtenu est au minimum de 29%. Cet écart nous indique que la qualité des solutions obtenues n'est pas garantie pour ces instances. Nous allons vérifier cela dans une prochaine section où ces solutions seront comparées avec celles obtenues par les méthodes approchées.

Le modèle à contraintes cumulatives est résolu pour 51% des instances en 60 minutes. Le MCC est donc résolu plus souvent que le modèle individualisé (43%). Cependant la résolution du MCC est plus affectée par l'augmentation du nombre de sites et de ressources. Pour le MI, le nombre d'instances résolues ainsi que l'écart relatif reste constant pour toutes les instances composées du même nombre de tâches. Pour le MCC, les instances à 10 ressources sont résolues 59% du temps, pour l'ensemble des instances le Gap moyen est de 15%, les instances avec 20 ressources sont elles résolues seulement 44% du temps, pour un Gap de 41%. Le nombre de sites influence de la même sorte la résolution des modèles. La résolution du MI n'est que très peu affectée par le nombre de sites. Pour la résolution du MCC, le Gap passe de 22% pour les instances à 2 sites à 34% pour les instances à 3 sites.

Dans la prochaine partie nous comparons ces résultats obtenus par la résolution des modèles mathématiques, aux résultats obtenus par les méthodes approchées.

5.5 Mise en œuvre des méta-heuristiques

Dans cette section, nos objectifs sont multiples. Tout d'abord nous validons nos méthodes en les comparant aux solutions obtenues par la résolution des modèles mathématiques. Ensuite les méthodes sont comparées entre elles, pour chaque codage, sur des instances adaptées de la littérature. Enfin une comparaison des codages est faite en fonction du temps de calcul alloué.

5.5.1 Validation des méthodes

Les instances utilisées dans cette partie sont décrites dans la section 5.3.1. Les résultats obtenus par les modèles mathématiques sont décrits dans la section 5.4. Nous considérons dans cette section la meilleure solution donnée par un des deux modèles mathématiques pour la comparer à celles obtenues par les méthodes approchées.

5.5.1.1 Protocole expérimental

Pour comparer chaque codage aux résultats obtenus par la résolution des modèles mathématiques, chaque codage est testé avec les trois méta-heuristiques proposées. Ces méta-heuristiques sont la recherche locale, le recuit simulé et la recherche locale itérée.

Pour la recherche locale itérée, deux critères d'acceptation sont évalués, celui de la recherche locale (LS) et celui du recuit simulé (SA). Pour chaque codage, nous devons définir les voisinages appliqués. Après de nombreuses expérimentations, la meilleure configuration pour l'application des mouvements est une application aléatoire pondérée. Cela signifie que pour chaque application d'un voisinage, un mouvement sera choisi aléatoirement selon une probabilité définie préalablement. Ces probabilités d'application des mouvements en fonction du codage sont données dans le tableau 5.6.

Mouvements	V1	V2	V3
σ, l, a	12,5%	12,5%	75%
σ, l	50%	50%	0%
σ	100%	0%	0%

TABLEAU 5.6 – Probabilité d'application des mouvements en fonction du codage

Les paramètres utilisés sont les suivants :
— Nombre d'itérations total : 100 000
— Réplications : 20
— Recuit simulé :
  — Température finale : 0.01
— Recherche Locale Itérée
  — Palier : 5000

Nous utiliserons les notations suivantes dans cette section :
— Recherche locale (LS)
— Recuit simulé (SA)
— Recherche Locale Itérée avec le critère d'acceptation de la Recherche Locale (ILS—LS)
— Recherche Locale Itérée avec le critère d'acceptation du recuit simulé (ILS—SA)
— MM Nous avons retenue la meilleure solution obtenue par les deux modèles au terme des 60 minutes.
— L'ensemble des méthodes $M = \{MM; LS; SA; ILS|LS; ILS|SA\}$
— L'ensemble des codages $D = \{(\sigma, l, a); (\sigma, l); (\sigma)\}$
— $X_{i,j}^{m,d}$ la valeur du makespan obtenu par la réplication $j = 1, 20$ sur l'instance $i = 1, 160$ avec la méthode $m \in M$ pour le codage $d \in D$.
— BKS_i la meilleure solution connue pour l'instance $i = 1, 160$
— $AvBKS^e$ la moyenne du makespan des meilleures solutions connues pour l'ensemble d'instances e. Le calcul du $AvBKS^e$ est donné par l'équation 5.7.

$$AvBKS^e = \frac{\sum_{i \in e} BKS_i}{|e|} \tag{5.7}$$

— $AvRG_i^{m,d}$ est la moyenne des écarts relatifs entre le makespan obtenu par la méthode $m \in M$ avec le codage $d \in D$ et la meilleure solution connue BKS_i pour l'instance $(i = 1, 160)$. Le calcul du $AvRG_i^m$ est donné par l'équation 5.8.

$$AvRG_i^{m,d} = \frac{\sum_{j=1}^{20} \frac{X_{i,j}^{m,d} - BKS_i}{BKS_i}}{20} \tag{5.8}$$

— $AvRGC_c^{m,d}$ est la moyenne des écarts relatifs moyen entre les solutions obtenues par la méthode $m \in M$ pour le codage $d \in D$ et les meilleures solutions connues pour les instances de la classe $(c = 1, 20)$. Le calcul de $AvRGC_c^{m,d}$ est assuré par l'équation 5.2.

$$AvRGC_c^{m,d} = \frac{\sum_{i \in c} AvRG_i^{m,d}}{8} \tag{5.9}$$

— $AvRG^{m,d}$ est la moyenne des écarts relatifs moyen entre les solutions obtenues par la méthode $m \in M$ pour le codage $d \in D$ et les meilleures solutions connues pour toutes les instances. Le calcul du $AvRG^{m,d}$ est donné par l'équation 5.10.

$$AvRG^{m,d} = \frac{\sum_{i=1}^{160} AvRG_i^{m,d}}{160} = \frac{\sum_{c=1}^{20} AvRGC_c^{m,d}}{20} \tag{5.10}$$

Les résultats obtenus par les différentes méthodes proposées sont reportés dans le tableau 5.8, 5.9 et 5.10 respectivement pour le codage (σ), (σ, l) et (σ, l, a). Les résultats retournés dans ces tableaux sont des pourcentages. La qualité des solutions apportées par les méthodes approchées est ici comparée à celles obtenues par la résolution des modèles mathématiques. Il est important de garder à l'esprit que les temps de calcul des méthodes approchées sont bien moindres que les 60 minutes accordées au solveur pour résoudre les modèles mathématiques. Ces temps de calcul moyens sont reportés dans le tableau 5.7. Dans ce tableau, les temps de calculs moyens pour une réplication sont donnés en fonction du nombre de tâches. Les temps de calcul ne dépendent pas des méta-heuristiques utilisées, seul le codage impacte ces temps.

Nombre de tâches	10	15	20	25	30
σ	5,49s	12,29s	21,85s	28,79s	34,64s
σ, l	2,11s	5,82s	10,21s	13,18s	16,28s
σ, l, a	0,86s	2,39s	3,57s	4,76s	5,64s

Tableau 5.7 – Temps de calcul moyen en fonction des codages

On constate dans le tableau 5.7 que plus le codage est complet, plus l'algorithme d'ordre strict sera rapide. On ne peut donc pas comparer les résultats obtenus par les différents codages pour un même nombre d'itérations. En effet, les méthodes rapides ont un espace de recherche plus grand à parcourir, mais peuvent parcourir plus de solutions en un temps donné. Cette comparaison sera l'objet de la section 5.5.3.

Les résultats du tableau 5.8 montrent que le codage (σ) donne de bons résultats sur ces petites instances générées. L'écart par rapport à la meilleure solution connue ne dépasse que rarement les 2% pour le recuit simulé et les recherches locales itérées. La recherche locale simple donne des résultats de moins bonne qualité, ceci étant dû aux nombreux minimums locaux dans l'espace de recherche. Les résultats des trois autres méthodes sont assez proches, mais on peut noter de meilleures performances de la recherche locale itérée avec le test de la recherche locale (ILS—LS) pour les instances de 25 tâches et plus.

Quant aux résultats obtenus par le codage σ, l, ils sont semblables à ceux obtenus pour le codage σ. Le recuit simulé et les deux recherches locales itérées obtiennent les meilleurs résultats et la recherche locale donne de moins bons résultats. Pour ce codage, bien que les résultats obtenus par les trois meilleures méthodes soient assez proches, la recherche locale

				LS	$ILS\|BW$	$ILS\|SA$	SA	MM
N	S	Ressources	$AvBKS^e$	$AvRG_c^{LS,d}$	$AvRG_c^{ILS\|BW,d}$	$AvRG_c^{ILS\|SA,d}$	$AvRG_c^{SA,d}$	$AvRG_c^{MM,d}$
10	2	10	26,25	0,19	**0,00**	**0,00**	**0,00**	**0,00**
	3	10	18,13	3,23	1,04	1,04	1,04	**0,00**
	2	20	28,25	2,54	0,63	0,63	0,63	**0,00**
	3	20	25	2,77	2,77	2,77	2,77	**0,00**
15	2	10	46	0,62	0,33	0,33	0,33	**0,00**
	3	10	29,38	3,29	1,02	1,02	1,14	**0,00**
	2	20	39,88	1,37	**0,00**	**0,00**	**0,00**	**0,00**
	3	20	39,88	3,71	2,29	2,21	2,14	**0,00**
20	2	10	62,5	0,58	0,38	0,28	0,07	**0,00**
	3	10	45,38	2,72	1,72	1,80	1,65	**0,27**
	2	20	58,38	3,01	**0,98**	1,23	1,60	5,36
	3	20	49,00	5,60	**2,66**	2,79	2,95	17,27
25	2	10	64,75	1,39	0,54	**0,34**	0,61	2,81
	3	10	56,25	2,92	**2,13**	2,20	2,79	10,88
	2	20	70,88	2,96	2,10	**1,98**	2,67	10,00
	3	20	72,63	3,72	**1,28**	1,51	1,69	27,82
30	2	10	79,25	1,07	**0,48**	0,54	0,75	X
	3	10	77,63	1,81	**0,82**	0,90	1,18	X
	2	20	74,13	3,48	**1,45**	1,55	1,80	X
	3	20	80,88	7,40	**3,04**	3,32	4,16	X
		Moyenne	52,78	2,72	**1,28**	1,32	1,50	X

TABLEAU 5.8 – Résultats obtenus avec le codage (σ)

				LS	$ILS\|BW$	$ILS\|SA$	SA	MM
N	S	Ressources	$AvBKS^e$	$AvRG_c^{LS,d}$	$AvRG_c^{ILS\|BW,d}$	$AvRG_c^{ILS\|SA,d}$	$AvRG_c^{SA,d}$	$AvRG_c^{MM,d}$
10	2	10	26,25	0,88	0,31	0,04	0,04	**0,00**
	3	10	18,13	3,66	0,20	0,36	**0,00**	**0,00**
	2	20	28,25	3,24	0,13	0,13	**0,00**	**0,00**
	3	20	25	2,81	0,19	**0,00**	**0,00**	**0,00**
15	2	10	46	0,95	0,17	0,20	0,23	**0,00**
	3	10	29,38	6,32	0,98	1,29	1,26	**0,00**
	2	20	39,88	2,46	0,34	0,15	0,38	**0,00**
	3	20	39,38	5,25	1,56	1,81	1,96	**0,00**
20	2	10	62,50	0,70	0,26	0,20	0,17	**0,00**
	3	10	45,38	3,82	0,93	1,01	1,43	**0,27**
	2	20	58,38	2,70	**0,75**	0,95	1,70	5,36
	3	20	49	6,94	**1,80**	1,89	2,29	17,27
25	2	10	64,75	2,52	**0,70**	1,25	1,63	2,81
	3	10	56,25	6,62	**2,97**	3,39	4,20	10,88
	2	20	70,88	3,99	1,41	**1,34**	2,28	10,00
	3	20	72,63	5,11	**1,92**	2,36	2,48	27,82
30	2	10	79,25	2,82	1,39	**0,93**	1,69	X
	3	10	77,63	2,66	1,67	**1,46**	1,91	X
	2	20	74,13	5,77	**3,05**	**3,05**	4,04	X
	3	20	80,88	11,27	**5,60**	6,72	6,72	X
		Moyenne	52,78	4,02	**1,32**	1,43	1,72	X

Tableau 5.9 – Résultats obtenus avec le codage (σ, l)

				LS	$ILS\|BW$	$ILS\|SA$	SA	MM
N	S	Ressources	$AvBKS^e$	$AvRG_c^{LS,d}$	$AvRG_c^{ILS\|BW,d}$	$AvRG_c^{ILS\|SA,d}$	$AvRG_c^{SA,d}$	$AvRG_c^{MM,d}$
10	2	10	26,25	10,63	0,52	1,21	1,75	**0,00**
	3	10	18,13	21,79	11,36	12,77	17,74	**0,00**
	2	20	28,25	9,47	2,65	3,14	2,85	**0,00**
	3	20	25	8,28	3,29	3,60	5,44	**0,00**
15	2	10	46	9,03	2,30	2,97	6,30	**0,00**
	3	10	29,38	21,03	13,30	14,78	24,50	**0,00**
	2	20	39,88	12,61	4,64	3,78	8,03	**0,00**
	3	20	39,88	13,74	10,11	9,85	11,47	**0,00**
20	2	10	62,5	4,77	3,68	3,46	4,16	**0,00**
	3	10	45,38	20,45	15,72	17,81	22,69	**0,27**
	2	20	58,38	18,45	8,81	9,74	12,86	**5,36**
	3	20	49	24,36	19,00	19,22	21,77	**17,27**
25	2	10	64,75	9,90	5,08	5,14	9,26	**2,81**
	3	10	56,25	22,90	18,79	21,32	29,09	**10,88**
	2	20	70,88	10,09	**6,50**	6,66	8,27	10,00
	3	20	72,63	20,61	**17,08**	18,27	21,50	27,82
30	2	10	79,25	7,32	5,37	**4,70**	5,52	X
	3	10	77,63	12,81	**12,33**	13,67	14,29	X
	2	20	74,13	12,83	**10,07**	10,52	12,40	X
	3	20	80,88	27,35	**26,16**	27,75	31,27	X
		Moyenne	52,78	14,92	**9,84**	10,52	13,56	X

TABLEAU 5.10 – Résultats obtenus avec le codage (σ, l, a)

itérée, avec le critère d'acceptation de la recherche locale, donne de meilleurs résultats pour les instances de 20 tâches et plus.

Les résultats obtenus pour le codage σ, l, a sont plus contrastés que pour les deux autres codages. Les résultats obtenus sont bons pour les instances de 10 tâches, sauf la classe d'instances avec 10 ressources et 3 sites. Pour les autres classes d'instances, les résultats sont moins bons et montrent clairement que les 100 000 itérations ne suffisent pas à bien explorer cet espace de recherche, qui est le plus grand des trois. Les méta-heuristiques ne donnent de meilleurs résultats que pour les instances de 25 tâches et plus. Cependant comme le montre le tableau 5.7, cette méthode est la plus rapide, et peut donc explorer davantage de solutions que les autres, pour un même temps de calcul.

5.5.2 Comparaison des méta-heuristiques sur les instances adaptées de la littérature

Cette section est consacrée aux expérimentations menées sur les instances adaptées de la littérature présentées dans la section 5.3.2. Le RCPSP est un cas particulier du RCPSP Multi-Site où un seul site est considéré. De ce fait, les solutions optimales des instances classiques de la littérature deviennent des bornes inférieures pour les instances adaptées du RCPSP Multi-Site. Dans le cas où une solution optimale n'est pas connue, la meilleure borne inférieure connue pour cette instance du RCPSP est aussi une borne inférieure pour l'instance adaptée. Dans la suite du chapitre nous utilisons les notations suivantes :

- Recherche locale (LS)
- Recuit simulé (SA)
- Recherche Locale Itérée avec le critère d'acceptation de la Recherche Locale ($ILS|LS$)
- Recherche Locale Itérée avec le critère d'acceptation du recuit simulé ($ILS|SA$)
- L'ensemble des méthodes $M = \{LS; SA; ILS|LS; ILS|SA\}$
- L'ensemble des codages $D = \{(\sigma, l, a); (\sigma, l); (\sigma)\}$
- $X_{i,j}^{m,d}$ la valeur du makespan obtenu par la réplication $j = 1, 20$ sur l'instance $i = 1, 480$ avec la méthode $m \in M$ pour le codage $d \in D$.
- BKS_i la meilleure solution connue pour l'instance $i = 1, 480$
- $BKLB_i$ la meilleure borne inférieure connue pour l'instance $i = 1, 480$
- $RGLB_i$ l'écart relatif entre la meilleure solution connue BKS_i pour l'instance $i = 1, 480$ et la meilleure borne inférieure connue $BKLB_i$.

$$RGLB_i = \frac{BKS_i - BKLB_i}{BKS_i}; \tag{5.11}$$

- $AvRGLB$ la moyenne des écarts aux meilleures bornes inférieures connues pour toutes les instances $i = 1, 480$

$$AvRGLB = \frac{\sum_{i=1}^{480} RGLB_i}{480}; \tag{5.12}$$

- $AvRG_i^{m,d}$ est la moyenne des écarts relatifs entre le makespan obtenu par la méthode $m \in M$ avec le codage $d \in D$ et la meilleure solution connue BKS_i

pour l'instance $i = 1, 480$. Le calcul du $AvRG_i^m$ est donné par l'équation 5.13.

$$AvRG_i^{m,d} = \frac{\sum_{j=1}^{20} \frac{X_{i,j}^{m,d} - BKS_i}{BKS_i}}{20} \tag{5.13}$$

— $AvRGC_c^{m,d}$ est la moyenne des écarts relatif entre la solution obtenue par la méthode $m \in M$ pour le codage $d \in D$ et les meilleures solutions connues pour les instances de la classe $c = 1, 48$. Le calcul du $AvRGC_c^{m,d}$ est assuré par l'équation 5.14.

$$AvRGC_c^{m,d} = \frac{\sum_{i \in c} AvRG_i^{m,d}}{10} \tag{5.14}$$

— $AvRG^{m,d}$ est l'écart relatif moyen entre les solutions obtenues par la méthode $m \in M$ pour le codage $d \in D$ et les meilleures solutions connues pour toutes les instances. Le calcul du $AvRG^{m,d}$ est donné par l'équation 5.15.

$$AvRG^{m,d} = \frac{\sum_{i=1}^{480} AvRG_i^{m,d}}{480} = \frac{\sum_{c=1}^{48} AvRGC_c^{m,d}}{48} \tag{5.15}$$

— $NBC^{m,d}$ est le nombre de classes pour lesquelles la méthode $m \in M$ pour le codage $d \in D$ donne les meilleurs écarts par rapport aux meilleures solutions connues. Le calcul est donné par l'équation 5.16.

$$NBC^{m,d} = \sum_{c=1}^{48} 1_{\{AvRGC_c^{m,d} = Min_{m' \in M}(AvRG_c^{m'})\}} \tag{5.16}$$

— $SD^{m,d}$ est la moyenne des écarts-types des résultats obtenus par la méthode $m \in M$ avec le codage $d \in D$. Le calcul est donné par l'équation 5.17.

$$SD^{m,d} = \frac{\sum_{c=1}^{48} \sqrt{\frac{\sum_{j=1}^{20} (X_{i,j}^{m,d} - \frac{\sum_{j=1}^{20} X_{i,j}^{m,d}}{20})^2}{20}}}{48} \tag{5.17}$$

Les expérimentations sont menées avec le même protocole expérimental que pour les petites instances générées. Ce protocole est décrit dans la section 5.5.1.1.

Pour chaque méta-heuristique, les temps de calcul sont équivalents pour un même ensemble d'instance. Nous reportons dans le tableau 5.11 les temps de calcul moyens en fonction du codage.

Nombre de tâches	30	60	90	120
σ	240,3s	695,0s	1136,1s	1595,3s
σ, l	74,2s	181,0s	316,0s	337,3s
σ, l, a	15,8s	29,3s	42,7s	52,1s

Tableau 5.11 – Temps de calcul moyen en fonction des codages

Les écarts relatifs entre les meilleures solutions connues et les meilleures bornes inférieures connues sont donnés dans le tableau 5.12.

Les résultats sont reportés dans le tableau 5.13 pour le codage (σ), le tableau 5.14 pour le codage (σ, l) et le tableau 5.15 pour le codage σ, l, a.

Tableau 5.12 – Écart relatif moyen par rapport à la meilleure borne inférieure connue

Nombre de tâches	AvRGLB
30	16,32%
60	20,38%
90	23,60%
120	43,91%

Tableau 5.13 – Résultats obtenus pour le codage (σ)

Méta-heuristiques	LS	ILS|LS	ILS|SA	SA
Résultats obtenus pour les instances de 30 tâches				
Min $AvRG_i^{m,d}$	**0,00**	**0,00**	**0,00**	**0,00**
$AvRG_i^{m,d}$	8,19	**5,95**	6,07	6,24
Max $AvRG_i^{m,d}$	29,82	**29,47**	**29,47**	29,65
$NBC^{m,d}$	7	**32**	28	24
$SD_i^{m,d}$	1,47	**0,52**	0,54	0,57
Résultats obtenus pour les instances de 60 tâches				
Min $AvRG_i^{m,d}$	**0,00**	**0,00**	**0,00**	**0,00**
$AvRG^{m,d}$	5,85	**3,62**	3,74	3,77
Max $AvRG_i^{m,d}$	13,81	**9,07**	9,17	10,31
$NBC^{m,d}$	5	25	**26**	21
$SD_i^{m,d}$	2,52	1,37	**1,23**	1,45
Résultats obtenus pour les instances de 90 tâches				
Min $AvRG_i^{m,d}$	**0,00**	**0,00**	**0,00**	**0,00**
$AvRG^{m,d}$	7,02	**3,21**	3,40	4,44
Max $AvRG_i^{m,d}$	15,99	8,97	**8,82**	10,62
$NBC^{m,d}$	0	**26**	18	4
$SD_i^{m,d}$	3,16	2,00	**1,73**	2,09
Résultats obtenus pour les instances de 120 tâches				
Min $AvRG_i^{m,d}$	2,33	2,63	**1,64**	1,74
$AvRG^{m,d}$	7,56	6,22	5,94	**5,41**
Max $AvRG_i^{m,d}$	15,77	16,76	**12,54**	16,34
$NBC^{m,d}$	3	5	13	**29**
$SD_i^{m,d}$	4,83	3,25	**3,17**	3,95

Dans le tableau 5.13, sont répertoriés les résultats obtenus avec le codage σ. On constate que la recherche locale itérée avec le critère d'acceptation de la recherche locale donne les meilleurs résultats. Les $AvRG^{m,d}$ moyens pour les instances de 60 et 90 tâches sont meilleurs que pour les instances de 30 tâches. Ceci s'explique par le fait que cet écart relatif est obtenu par comparaison aux meilleures solutions données par l'ensemble des méthodes sur l'ensemble des codages. Or, pour les instances de 30 tâches, les meilleures solutions connues sont données par le codage σ, l et pour les instances de 60 tâches et plus, par le codage (σ) ce qui explique ce phénomène.

Tableau 5.14 – Résultats obtenus pour le codage (σ, l)

Méta-heuristiques	LS	ILS|LS	ILS|SA	SA
Résultats obtenus pour les instances de 30 tâches				
Min $AvRG_i^{m,d}$	**0,00**	**0,00**	**0,00**	**0,00**
$AvRG^{m,d}$	5,72	**2,71**	2,99	3,16
Max $AvRG_i^{m,d}$	13,77	8,98	**8,37**	8,57
$NBC^{m,d}$	3	**27**	23	16
$SD_i^{m,d}$	2,12	**1,07**	1,24	1,31
Résultats obtenus pour les instances de 60 tâches				
Min $AvRG_i^{m,d}$	**0,00**	**0,00**	**0,00**	**0,00**
$AvRG^{m,d}$	6,47	5,05	5,38	**4,66**
Max $AvRG_i^{m,d}$	19,34	16,81	17,03	**15,93**
$NBC^{m,d}$	3	16	9	**27**
$SD_i^{m,d}$	3,13	2,29	**2,12**	2,42
Résultats obtenus pour les instances de 90 tâches				
Min $AvRG_i^{m,d}$	**0,00**	**0,00**	**0,00**	**0,00**
$AvRG^{m,d}$	8,82	8,22	8,20	**6,62**
Max $AvRG_i^{m,d}$	23,02	23,02	26,79	**20,28**
$NBC^{m,d}$	3	4	7	**36**
$SD_i^{m,d}$	3,67	2,97	3,06	**2,87**
Résultats obtenus pour les instances de 120 tâches				
Min $AvRG_i^{m,d}$	**2,13**	4,52	3,14	3,25
$AvRG^{m,d}$	11,99	11,88	11,94	**9,14**
Max $AvRG_i^{m,d}$	28,68	28,17	28,82	25,48
$NBC^{m,d}$	5	1	5	**37**
$SD_i^{m,d}$	4,76	4,48	**3,97**	4,21

Le tableau 5.14 donne les résultats obtenus avec le codage (σ, l). La recherche locale itérée avec le critère d'acceptation de la recherche locale donne les meilleurs résultats. On constate cependant que les résultats des méta-heuristiques sont proches des résultats obtenus par une simple recherche locale sur les instances de 120 tâches. Ceci est dû au fait que, sur ces instances de cette taille, les méta-heuristiques acceptant des perturbations de la solution convergent trop lentement vers une solution optimale.

Le tableau 5.15 donne les résultats obtenus avec le codage (σ, l, a). La recherche locale donne les meilleurs résultats. Ceci est dû au fait que les autres méthodes ont besoin de plus d'itérations pour converger. Cependant les résultats restent de mauvaise qualité comme

TABLEAU 5.15 – Résultats obtenus pour le codage (σ, l, a)

Méta-heuristiques	LS	ILS|LS	ILS|SA	SA
Résultats obtenus pour les instances de 30 tâches				
Min $AvRG_i^{m,d}$	**0,00**	**0,00**	**0,00**	0,13
$AvRG^{m,d}$	**40,49**	40,90	40,70	40,60
Max $AvRG_i^{m,d}$	**135,58**	141,22	142,65	136,73
$NBC^{m,d}$	**18**	11	16	9
$SD_i^{m,d}$	3,57	3,27	**2,91**	3,33
Résultats obtenus pour les instances de 60 tâches				
Min $AvRG_i^{m,d}$	4,02	3,55	**3,36**	4,21
$AvRG^{m,d}$	**64,17**	66,70	66,94	66,02
Max $AvRG_i^{m,d}$	**171,43**	176,36	171,30	173,77
$NBC^{m,d}$	**26**	6	8	9
$SD_i^{m,d}$	5,59	5,49	5,46	**5,42**
Résultats obtenus pour les instances de 90 tâches				
Min $AvRG_i^{m,d}$	**5,59**	9,18	9,68	8,71
$AvRG^{m,d}$	**83,99**	88,37	88,26	86,04
Max $AvRG_i^{m,d}$	**208,88**	216,13	212,13	211,13
$NBC^{m,d}$	**31**	0	1	16
$SD_i^{m,d}$	**6,52**	7,13	6,75	6,53
Résultats obtenus pour les instances de 120 tâches				
Min $AvRG_i^{m,d}$	18,66	20,83	**18,53**	19,07
$AvRG^{m,d}$	**89,75**	91,71	92,44	91,26
Max $AvRG_i^{m,d}$	238,97	**237,16**	235,49	249,71
$NBC^{m,d}$	**27**	4	4	14
$SD_i^{m,d}$	8,50	9,09	8,76	**8,24**

c'était déjà le cas pour les petites instances.

5.5.3 Évolution du comportement des méthodes dans le temps

Cette section a pour objectif de comparer les codages et de déterminer leurs avantages et inconvénients en fonction du temps de calcul alloué. Nous avons décidé de comparer ces codages en fonction du temps de calcul. Ce choix se justifie par le fait que les espaces de recherche sont de taille différentes et que les méthodes sont plus ou moins rapides pour construire et évaluer une solution. Par exemple, le codage (σ) possède un espace de recherche réduit par rapport au codage (σ, l, a). Cependant, l'algorithme d'ordre strict associé au codage (σ) est plus complexe et donc plus lent que celui du codage (σ, l, a). Pour un même temps de calcul, la première méthode évalue donc moins de solutions, mais dans un espace de recherche plus petit, avec une qualité de solution moyenne supérieure. Pendant ce même temps de calcul, la deuxième méthode évalue plus de solutions, dans un espace de recherche plus vaste et avec une qualité de solution moyenne plus faible. Notre objectif est donc de déterminer quelle stratégie est la meilleure, en fonction du temps de calcul alloué et des caractéristiques des instances traitées.

Une seul méthode sera utilisée dans cette partie, la recherche locale itérée avec le critère d'acceptation de la recherche locale. Cette méthode a donné les meilleurs résultats en moyenne pour chacun des codages sur les petites instances et sur les instances de la littérature pour les codages (σ) et (σ, l). Les paramètres utilisés sont les suivants :

- Temps de résolution : 10 minutes par réplication
- Réplications : 20
- Méthode utilisée : Recherche Locale Itérée
  - Palier : 5000
  - Critère d'acceptation : Recherche Locale $(ILS|LS)$

Pour pouvoir faire des comparaisons, nous reportons pour chaque méthode les résultats en fonction du temps sur les 10 minutes d'exécution totales. Ces résultats sont reportés sous forme de graphique. Ces graphiques correspondent à la moyenne des makespans en fonction du temps de calcul pour un ensemble d'instances. Le graphe de la figure 5.1 montre les valeurs du makespan en fonction du temps et du codage utilisé pour les instances de 30 tâches.

Les résultats présentés sur la figure 5.1 montrent deux choses :

- Le codage σ, l, a ne donne pas de bons résultats, même au bout de 10 minutes
- Le codage σ, l donne les meilleurs résultats au bout de 10 minutes

Ce qui reste à déterminer, c'est à partir de quel temps de calcul le codage (σ, l) donne de meilleurs résultats que le codage (σ). Pour cela, on retranscrit sur la figure 5.2 les valeurs du makespan obtenus par les codages (σ) et (σ, l) sur une période de temps plus réduite.

On constate sur la figure 5.2 que le codage (σ, l) donne de meilleurs résultats pour les instances de 30 tâches à partir de 2,5 secondes de calcul en moyenne. L'objectif de cette section sera de déterminer à partir de combien de temps le codage (σ, l) donne de meilleurs résultats que le codage (σ). Par la suite nous donnerons donc uniquement les graphes montrant l'intersection des courbes de résultats pour les codages (σ) et (σ, l). Pour les instances de 60 tâches, les résultats en fonction du temps sont donnés dans le graphique de la figure 5.3. Les résultats montrent que le codage (σ, l) donne de meilleurs

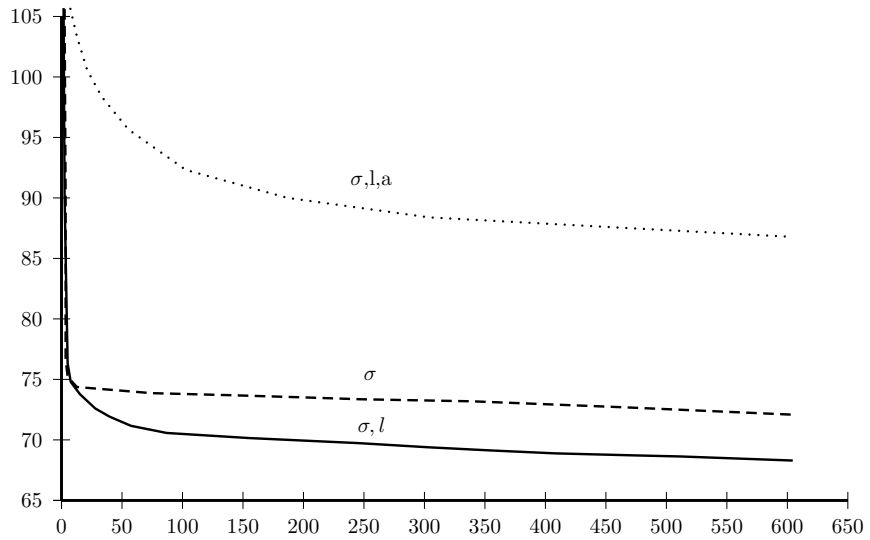

FIGURE 5.1 – Graphe du makespan moyen en fonction du temps (s) pour les instances de 30 tâches

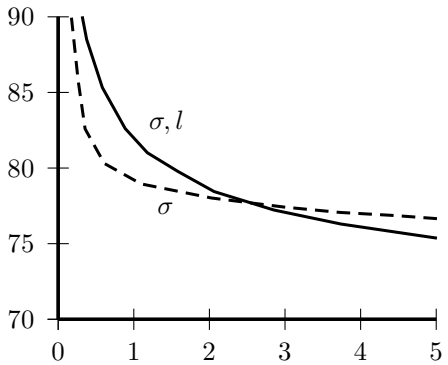

FIGURE 5.2 – Graphe des résultats des codages (σ) et (σ, l) en fonction du temps (s)

résultats en moyenne pour les instances de 60 tâches au bout de 28 secondes.

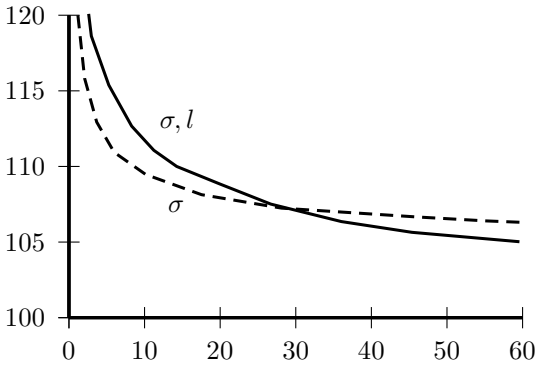

FIGURE 5.3 – Graphe du makespan moyen en fonction du temps (s) pour les instances de 60 tâches

Pour les instances de 90 tâches, les résultats sont donnés dans le graphique de la figure 5.4. Ces résultats montrent que le codage (σ, l) donne de meilleurs résultats en moyenne au bout de 370 secondes.

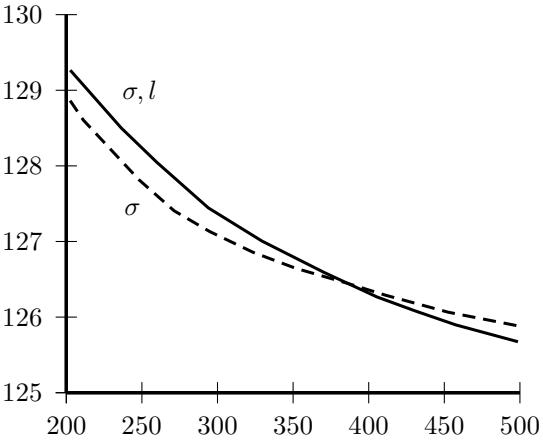

FIGURE 5.4 – Graphe du makespan moyen en fonction du temps (s) pour les instances de 90 tâches

Pour les instances de 120 tâches, les résultats sont donnés dans le graphique de la figure 5.5. Ces résultats montrent que le codage (σ, l) ne donne pas de meilleurs résultats en moyenne au bout des 10 minutes de temps de calcul.

Nous préconisons donc d'utiliser le codage (σ, l) sur les instances de taille petite et moyenne, le codage (σ) au delà.

5.6 Conclusion

Dans cette section, nous avons présenté les résultats obtenus avec les différentes méthodes de résolution. Tout d'abord, nous avons testé les méthodes approchées sur les instances du RCPSP classique. En effet, le RCPSP étant un cas particulier du RCPSP Multi-Site, ces méthodes doivent être capables de résoudre ces instances. Nous avons résolu

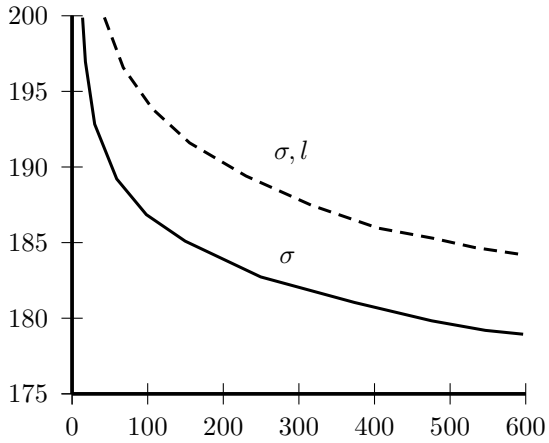

FIGURE 5.5 – Graphe du makespan moyen en fonction du temps (s) pour les instances de 120 tâches

les instances de la littérature de la PSPLIB (Kolisch and Sprecher, 1997). Les résultats montrent que les méthodes fonctionnent sur ces instances, cependant cela ne garantit pas leur fonctionnement pour des instances spécifiques du RCPSP Multi-Site. Pour cela nous avons dû créer des instances spécifiques au RCPSP Multi-Site. Nous avons créé deux types d'instances, des petites pour obtenir des résultats avec les modèles mathématiques et des grandes adaptées des instances de littérature du RCPSP classique.

Nous avons comparé les deux modèles mathématiques entre eux sur les petites instances créées. Les résultats montrent que les modèles sont assez proches, bien que le modèle à contraintes cumulatives donne de meilleurs résultats.

Nous avons ensuite validé nos méthodes de résolution approchées sur les petites instances créées. Nous avons comparé les différentes méta-heuristiques pour chaque codage sur ces instances. Globalement la recherche locale itérée avec le critère d'acceptation de la recherche locale donne les meilleurs résultats. Nous avons ensuite testé ces mêmes méthodes sur les instances adaptées de la littérature et encore une fois la recherche locale itérée avec le critère d'acceptation de la recherche locale donne les meilleurs résultats.

Pour comparer les codages entre eux, nous avons exécuté les différents codages avec la recherche locale itérée sur une période de dix minutes maximum. L'objectif était de déterminer quel codage est le plus performant en fonction du temps de calcul alloué. La conclusion est que le codage (σ, l, a) ne donne pas de solutions satisfaisantes en moins de 10 minutes pour les instances de plus de 30 tâches. Pour ce qui est des deux autres codages, le codage (σ) donne les meilleurs résultats à court terme en moyenne. A partir d'une certaine durée, le codage (σ, l) donne de meilleurs résultats en moyenne. Cette durée dépend du nombre de tâches des instances traitées : plus une instance comporte de tâches, plus la durée nécessaire pour que le codage (σ, l) donne de meilleurs résultats sera longue.

Conclusion et perspectives

Dans ce document nous nous sommes intéressés aux problématiques de mutualisation de ressources dans un contexte multi-site. Ces problématiques proviennent du domaine industriel et depuis peu le domaine médical s'y intéresse. Les intérêts de la mutualisation entre plusieurs entités sont multiples. La mutualisation permet tout d'abord de réduire des coûts, soit des coûts de production, soit des coûts de main d'oeuvre ou encore des coûts logistiques. Un autre avantage de la mutualisation est la flexibilité de la production : en mutualisant les ressources entre plusieurs entités, ces ressources pourront être mieux utilisées, préemptées en cas de forte demande ou bien "louées" en cas de creux dans la demande. L'échange d'expérience et le fait de limiter la précarité de l'emploi pour les salariés sont aussi des avantages de la mutualisation de ressources.

Ces problématiques sont récentes et peu de travaux s'y sont intéressés d'un point de vue opérationnel. Nous avons proposé un modèle afin de considérer la mutualisation de ressources entre plusieurs sites distants. Cette modélisation est une extension du problème classique du RCPSP. Ce problème a beaucoup été étudié dans la littérature, cependant assez peu d'extensions considèrent les temps de transport. Aucune d'entre elles ne considèrent tous les aspects du transport dû à la mutualisation de ressources en contexte multi-site. Le premier aspect est l'affectation des tâches aux sites, qui va entraîner des temps de transport de produits semi-finis entre les différentes entités. Le second aspect est le transport de ressources. Les ressources étant mutualisées, elles sont sujettes aux déplacements entre les sites. Le but de cette extension est de considérer ces deux aspects de la mutualisation de ressources.

Ce problème étant nouveau, il n'existe donc pas de méthode dans la littérature pour le résoudre. Nous avons proposé deux modèles mathématiques pour la résolution du RCPSP Multi-Site. Cependant ce problème étant NP-difficile au sens fort, les instances de moyenne et grande taille ne peuvent pas être résolues en temps raisonnable. Nous avons donc proposé des méthodes de résolution approchées à base de méta-heuristiques. Pour résoudre ce problème nous couplons des algorithmes d'ordre strict à des méta-heuristiques. Le principe est de considérer des solutions partielles représentées par un codage. A chaque codage correspond un algorithme d'ordre strict qui a pour but de construire une solution à partir d'une représentation partielle. Nous avons proposé trois codages différents, un codage simple composé d'une liste topologique de tâches, un codage composé d'une liste topologique et d'une affectation des tâches aux sites, et enfin un codage complet, composé d'une liste topologique, d'une affectation des tâches aux sites et d'une affectation des ressources aux tâches. Ces codages sont plus ou moins complets et ne représentent pas le

même espace de recherche.

Pour comparer ces méthodes de résolution entre elles, afin de déterminer leurs performances en fonction des instances considérées, nous avons créé deux librairies d'instances :

— Une librairie d'instances de petite taille, dont la plupart a pu être résolue par la résolution des modèles mathématiques proposés. Ces instances permettent de valider les modèles mathématiques, ainsi que d'apporter un premier élément de comparaison des méthodes approchées.
— Une libraire d'instances de moyenne et grande taille, construite en adaptant les instances de la PSPLIB, qui permettent d'étudier le comportement des méta-heuristiques dans des conditions plus difficiles.

Enfin nous avons comparé les performances des codages utilisés en fonction du temps de calcul alloué afin de déterminer quel codage est le plus approprié à la résolution d'une instance en fonction de sa taille.

Nous avons également proposé des pistes de résolution pour une extension du RCPSP Multi-site où de nouvelles contraintes sont prises en considération. Ces contraintes additionnelles sont celles que l'on retrouve dans des problèmes de mutualisation tel que celui du Groupement Hospitalier de Territoire. Des incompatibilités peuvent exister entre les ressources ou bien entre les ressources et les tâches. De plus, les ressources ne sont pas en permanence disponibles, on considère donc des périodes d'indisponibilités pour les ressources. Nous avons proposé une formulation mathématique pour ce problème. Nous avons également proposé des adaptation possibles de nos méthodes de résolution approchées pour ce problème. Nous avons eu une démarche prospective dans cette section, le but a été de proposer des adaptations possibles de nos méthodes pour chacune des contraintes. L'objectif est de proposer une résolution modulable en fonction des contraintes considérées.

Les perspectives que nous souhaitons donner à ces travaux sont les suivantes :

— Proposer de nouveaux algorithmes d'ordre strict ne considérant pas l'individualisation des ressources. Ces algorithmes pourraient utiliser par exemple une méthode de résolution de problèmes de multi-flot pour l'ordonnancement des tâches. Cette méthode de résolution, pour le RCPSP, est présentée dans la section 2.3.4.
— Étudier le problème de RCPSP Multi-Site en considérant d'autre critères. Le nombre de déplacements effectués par les ressources durant toute la durée du projet pourrait être une piste de travail.
— Utiliser des méta-heuristiques basée population. L'objectif étant de sortir de certaine configuration sur les répartition des tâches sur les sites. Dans certains cas les méthodes individu vont, par exemple, avoir tendance à converger vers une solution où toutes les tâches sont exécutées sur un seul site, alors que de meilleures solutions existent en répartissant les tâches sur plusieurs sites.
— Poursuivre nos travaux sur le problème de RCPSP Multi-Site avec contraintes. Créer des instances pour ce problème et tester les méthodes de résolution proposées. Proposer et tester des méthodes de résolution pour le problème des incompatibilités entre ressources. Cette contrainte est l'une des composantes les plus complexes à résoudre. Cela est dû au fait que trouver une affectation d'un ensemble de ressources compatibles pour exécuter une tâche, revient à résoudre un problème NP-difficile.
— Considérer l'aspect stochastique de l'ordonnancement de tâches dans un contexte multi-site. Cet aspect stochastique peut être dû à l'absence de ressources humaines

ou la panne de ressources matérielles. La durée d'exécution des tâches ou les temps de transport peuvent aussi être impactés par des aléas. L'objectif serait de proposer des méthodes robustes à tous ces aléas, ou bien des méthodes de correction de l'ordonnancement en cas de manque de ressources.

Bibliographie

A. Alan B. Pritsker, Lawrence J. Watters, P. M. W. (1969). Multiproject scheduling with limited resources : A zero-one programming approach. *Management Science*, 16(1) :93–108.

Artigues, C., Demassey, S., and Neron, E. (2013). *Resource-constrained project scheduling : models, algorithms, extensions and applications*. John Wiley & Sons.

Bagherinejad, J. and Majd, Z. R. (2014). Solving the mrcpsp/max with the objective of minimizing tardiness/earliness cost of activities with double genetic algorithms. *The International Journal of Advanced Manufacturing Technology*, 70(1) :573–582.

Ballestín, F., Barrios, A., and Valls, V. (2013). Looking for the best modes helps solving the mrcpsp/max. *International Journal of Production Research*, 51(3) :813–827.

Barrios, A., Ballestín, F., and Valls, V. (2011). A double genetic algorithm for the mrcpsp/max. *Computers & Operations Research*, 38(1) :33 – 43. Project Management and Scheduling.

Bellenguez, O. and Néron, E. (2005). *Lower Bounds for the Multi-skill Project Scheduling Problem with Hierarchical Levels of Skills*, pages 229–243. Springer Berlin Heidelberg, Berlin, Heidelberg.

Bellenguez-Morineau, O. (2008). Methods to solve multi-skill project scheduling problem. *4OR*, 6(1) :85–88.

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

Boutevin, C. (2003). *Problèmes d'ordonnancement et d'affectation avec contraintes de ressources de type RCPSP et line balancing*.

Cai, Z. and Li, X. (2012). A hybrid genetic algorithm for resource-constrained multi-project scheduling problem with resource transfer time. In *Automation Science and Engineering (CASE), 2012 IEEE International Conference on*, pages 569–574. IEEE.

Carlier, J. (1984). *Problèmes d'ordonnancement à contraintes de ressources : algorithmes et complexité*. PhD thesis, Université Paris VI, Paris.

Chassiakos, A. and Sakellaropoulos, S. (2005). Time-cost optimization of construction projects with generalized activity constraints. *Journal of Construction Engineering and Management*, 131(10) :1115–1124.

Cierge, P. (2005). Contribution à la planification de tests. Mémoire d'ingénieur, CNAM Clermont-Ferrand.

Correia, I., Lourenço, Lìdia, L., and Saldanha-da Gama, F. (2012a). Project scheduling with flexible resources : formulation and inequalities. *OR Spectrum*, 34(3) :635–663.

Correia, I., Lourenço, L. L., and Saldanha-da Gama, F. (2012b). Project scheduling with flexible resources : formulation and inequalities. *OR spectrum*, 34(3) :635–663.

De Reyck, B. and Herroelen, W. (1999). The multi-mode resource-constrained project scheduling problem with generalized precedence relations. *European Journal of Operational Research*, 119(2) :538–556.

Dujardin, B. (2006). Mutualiser pour répondre à de nouveaux besoins.

Elmaghraby, S. E. (1977). *Activity networks : Project planning and control by network models*. Wiley New York.

Garey, M. R. and Johnson, D. S. (1979). Computers and intractability : a guide to the theory of np-completeness. *WH Freeman & Co., San Francisco*.

Gourgand, M., Grangeon, N., and Klement, N. (2015). Activities planning and resources assignment on distinct places : A mathematical model. *RAIRO - Operations Research*, 49, à paraitre.

Granata, J. (2011). La mutualisation.

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

Heilmann, R. (2001). Resource–constrained project scheduling : a heuristic for the multi–mode case. *OR-Spektrum*, 23(3) :335–357.

Heilmann, R. (2003). A branch-and-bound procedure for the multi-mode resource-constrained project scheduling problem with minimum and maximum time lags. *European Journal of Operational Research*, 144(2) :348 – 365.

Icmeli, O., Selcuk Erenguc, S., and Zappe, C. J. (1993). Project scheduling problems : a survey. *International Journal of Operations & Production Management*, 13(11) :80–91.

Jedrzejowicz, P. and Ratajczak-Ropel, E. (2011a). *A-Team for Solving MRCPSP/max Problem*, pages 466–475. Springer Berlin Heidelberg, Berlin, Heidelberg.

Jedrzejowicz, P. and Ratajczak-Ropel, E. (2011b). *Double-Action Agents Solving the MRCPSP/Max Problem*, pages 311–321. Springer Berlin Heidelberg, Berlin, Heidelberg.

Jedrzejowicz, P. and Skakovski, A. (2010). *A Cross-Entropy Based Population Learning Algorithm for Multi-mode Resource-Constrained Project Scheduling Problem with Minimum and Maximum Time Lags*, pages 383–392. Springer Berlin Heidelberg, Berlin, Heidelberg.

Kan, A. R. (1976). *Machine scheduling problem : Classification, complexity and computations*. Martinus Nijhoff.

Karp, R. M. (1972). *Reducibility among Combinatorial Problems*, pages 85–103. Springer US, Boston, MA.

Kazemipoor, H., Tavakkoli-Moghaddam, R., and Shahnazari-Shahrezaei, P. (2002). Solving a mixed-integer linear programming model for a multi-skilled project scheduling problem by simulated annealing. *Management Science Letters*, 2(2) :681–688.

Kelley, J. L., Namioka, I., Donoghue Jr, W. F., Lucas, K. R., Pettis, B., Poulsen, E. T., Price, G. B., Robertson, W., Scott, W., and Smith, K. T. (1963). *Linear topological spaces*. Springer.

Kirkpatrick, S., Vecchi, M. P., et al. (1983). Optimization by simulated annealing. *science*, 220(4598) :671–680.

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

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

Kolisch, R. (1995). *Project Scheduling under Resource Constraints : Efficient Heuristics for Several Problem Classes*, chapter Project Scheduling with Setup Times, pages 177–185. Physica-Verlag HD, Heidelberg.

Kolisch, R. (1998). Integrated scheduling, assembly area- and part-assignment for large scale, make-to-order assemblies. *International Journal of Production Economics*, 64 :127–141.

Kolisch, R. and Hartmann, S. (1999). *Heuristic algorithms for the resource-constrained project scheduling problem : Classification and computational analysis*. Springer.

Kolisch, R. and Hartmann, S. (2006). Experimental investigation of heuristics for resource-constrained project scheduling : An update. *European journal of operational research*, 174(1) :23–37.

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

Krüger, D. and Scholl, A. (2009). A heuristic solution framework for the resource constrained (multi-) project scheduling problem with sequence-dependent transfer times. *European Journal of Operational Research*, 197(2) :492–508.

Krüger, D. and Scholl, A. (2010). Managing and modelling general resource transfers in (multi-) project scheduling. *OR spectrum*, 32(2) :369–394.

Kumar, N. (2014). A simulated annealing algorithm for resource allocation in multiple projects. *International Journal of Engineering, Management & Sciences (IJEMS)*, 1(1).

Larcher, G. (2008). Rapport de la commission de concertation sur les mission de l'hôpital, présidée par m. gérard larcher. Rapport ministériel, Commission de concertation sur les missions de l'hôpital.

Laurent, A., Deroussi, L., Grangeon, N., and Norre, S. (2014). UNE EXTENSION DU RCPSP POUR LA MUTUALISATION DE RESSOURCES ENTRE PLUSIEURS SITES : LE MULTI LOCATION RCPSP. In *MOSIM 2014, 10ème Conférence Francophone de Modélisation, Optimisation et Simulation*, Nancy, France. Colloque avec actes et comité de lecture. internationale.

Laurent, A., Deroussi, L., Grangeon, N., and Norre, S. (2015). MULTI-SITE RCPSP : NOTATIONS, MATHEMATICAL MODEL AND RESOLUTION METHOD . In *45th International Conference on Computers & Industrial Engineering (CIE45)*, Metz, France.

Laurent, A., Deroussi, L., Grangeon, N., and Norre, S. (2016). *Modelisation and resolution of a problem of resource pooling in a hospital context*, pages 667–702. Lavoisier.

Laurent, A., Deroussi, L., Grangeon, N., and Norre, S. (2017). Comparison of Three Solution Encodings and Schedule Generation Scheme for the Multi-Site RCPSP. In *20th IFAC World Congress*, Toulouse, France.

Lombardi, M. and Milano, M. (2009). A precedence constraint posting approach for the rcpsp with time lags and variable durations. In Gent, I., editor, *Principles and Practice of Constraint Programming - CP 2009*, volume 5732 of *Lecture Notes in Computer Science*, pages 569–583. Springer Berlin Heidelberg.

Lourenço, H. L., Martin, O. C., and Stutzle, T. (2003). *Handbook of Metaheuristics*, chapter Iterated local search, pages 321–353. Kluwers Academic Publishers.

Metropolis, N., Rosenbluth, A. W., Rosenbluth, M. N., Teller, A. H., and Teller, E. (1953). Equation of state calculations by fast computing machines. *The journal of chemical physics*, 21(6) :1087–1092.

Mika, M., Waligóra, G., and Weglarz, J. (2004). A metaheuristic approach to scheduling workflow jobs on a grid. In *Grid resource management*, pages 295–318. Springer.

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

Mika, M., Waligóra, G., and Weglarz, J. (2011). Modelling and solving grid resource allocation problem with network resources for workflow applications. *Journal of Scheduling*, 14(3) :291–306.

Mittal, M. and Kanda, A. (2009). Scheduling of multiple projects with resource transfers. *International journal of mathematics in operational research*, 1(3) :303–325.

Neumann, K., Schwindt, C., and Zimmermann, J. (2012). *Project scheduling with time windows and scarce resources : temporal and resource-constrained project scheduling with regular and nonregular objective functions*. Springer Science & Business Media.

Norre, S. (2005). *Heuristiques et métaheuristiques pour la résolution de problèmes d'optimisation combinatoire dans les système de production*. Habilitation à diriger des recherches en informatique, Université Blaise Pascal. 296 pages.

Oğuz, O. and Bala, H. (1994). A comparative study of computational procedures for the resource constrained project scheduling problem. *European Journal of Operational Research*, 72(2) :406 – 416.

Ponz-Tienda, J. L., Pellicer, E., Benlloch-Marco, J., and Andrés-Romano, C. (2015). The fuzzy project scheduling problem with minimal generalized precedence relations. *Computer-Aided Civil and Infrastructure Engineering*, 30(11) :872–891.

Sabzehparvar, M. and Seyed-Hosseini, S. M. (2008). A mathematical model for the multi-mode resource-constrained project scheduling problem with mode dependent time lags. *The Journal of Supercomputing*, 44(3) :257–273.

Schnell, A. and Hartl, R. F. (2016). On the efficient modeling and solution of the multi-mode resource-constrained project scheduling problem with generalized precedence relations. *OR Spectrum*, 38(2) :283–303.

Schwindt, C. (2006). *Resource allocation in project management*. Springer Science & Business Media.

Sprecher, A., Kolisch, R., and Drexl, A. (1995). Semi-active, active, and non-delay schedules for the resource-constrained project scheduling problem. *European Journal of Operational Research*, 80(1) :94 – 102.

T'Kindt, V. and Billaut, J. (2000). *L'ordonnancement multicritère*. Presses Universitaires de Tours.

Toussaint, H. (2010). *Algorithmique rapide pour les problèmes de tournées et d'ordonnancement*. PhD thesis, Université Blaise Pascal.

Vanhoucke, M. (2004). Work continuity constraints in project scheduling. Working Papers of Faculty of Economics and Business Administration, Ghent University, Belgium 04/265, Ghent University, Faculty of Economics and Business Administration.

Yang, B., Geunes, J., and O'brien, W. J. (2001). Resource-constrained project scheduling : Past work and new directions.

Yang, K.-K. and Sum, C.-C. (1993). A comparison of resource allocation and activity scheduling rules in a dynamic multi-project environment. *Journal of Operations Management*, 11(2) :207–218.

Yang, K.-K. and Sum, C.-C. (1997). An evaluation of due date, resource allocation, project release, and activity scheduling rules in a multiproject environment. *European Journal of Operational Research*, 103(1) :139–154.