GÉNÉRALISATIONS DU PROBLÈME D'ORDONNANCEMENT DE PROJET À RESSOURCES LIMITÉES

Thèse

Roubila Lilia Kadri

Doctorat en sciences de l'administration-opérations et systèmes de décision
Philosophiæ doctor (Ph.D.)

Québec, Canada

GÉNÉRALISATIONS DU PROBLÈME D'ORDONNANCEMENT DE PROJET À RESSOURCES LIMITÉES

Thèse

Roubila Lilia Kadri

Sous la direction de:

Fayez Fouad Boctor, directeur de recherche
Jacques Renaud, codirecteur de recherche

Résumé

Un problème d'ordonnancement de projet à ressources limitées (POPRL) consiste en l'ordonnancement d'un ensemble de tâches, nécessitant un ou plusieurs types de ressources, renouvelables ou non renouvelables, en quantités limitées.

La résolution d'un POPRL a pour but la détermination des dates d'exécution des tâches en tenant compte des contraintes de préséance et de disponibilité des ressources et ayant comme objectif la minimisation de la durée totale du projet. Le POPRL est un problème d'optimisation combinatoire de complexité NP-dur (Blazewicz et al. 1983).

Une revue de littérature du (POPRL) est présentée au chapitre 2. Plus de 125 articles scientifiques sont analysés. Les contributions relatives à ce problème portent sur les méthodes exactes de résolution, la détermination de bornes inférieures sur la durée du projet et les méthodes heuristiques (approchées) de résolution. L'aspect pratique de ce problème dans des contextes industriels divers a conduit à de nombreuses généralisations du problème classique.

On constate que malgré les efforts déployés pour définir des POPRL plus généraux, les contraintes de transfert des ressources continuent à être ignorées, nous constatons aussi que l'optimisation du problème en considérant les coûts a été très peu traitée dans la littérature. Ce qui forcent les gestionnaires dans la plus part des cas à se baser uniquement sur leur expérience pour réaliser ou ajuster manuellement les ordonnancements produits par des heuristiques conçues pour résoudre des versions simplifiées du problème. Cette thèse tente de combler partiellement ces lacunes.

Le chapitre 3 traite le problème d'ordonnancement de projet à ressources limitées POPRLTT avec des temps de transfert des ressources. Un temps de transfert est le temps nécessaire pour transférer une ressource du lieu d'execution d'une activité vers un autre. Ainsi, le temps de transfert d'une ressource dépend des lieux des activités à exécuter, ainsi que des caractéristiques des ressources à transférer.
L'objectif dans un POPRLTT est la détermination des dates d'exécution des tâches en tenant compte des contraintes de préséance et de disponibilité des ressources et les temps de transfert des ressources. L'objectif est de minimiser la durée totale du projet. Nous proposons un nouvel algorithme génétique basé sur un opérateur de croisement de deux positions. L'étude expérimentale menée sur un grand nombre de problèmes test prouve que l'algorithme proposé est meilleur que les deux méthodes déjà existantes dans la littérature.

Une généralisation du problème d'ordonnancement de projet à ressources limitées et des temps de transfert des ressources au contexte multi mode (POPRL/PMETT) est présentée au chapitre 4. Dans ce problème, nous supposons que la préemption est non autorisée, et les ressources utilisées sont renouvelables et non renouvelables, chaque activité a plusieurs modes d'exécution, et les relations de préséance sont de type dit début-fin sans décalage. L'objectif est de choisir un temps de début (ou de fin) et un mode d'exécution pour chaque tâche du projet, pour que la durée du projet soit minimisée tout en respectant les contraintes de préséance, de disponibilité de ressources et les temps de transfert. Au meilleur de notre connaissance, cette version du problème n'a jamais été abordée auparavant. Nous proposons une formulation mathématique de ce problème, ensuite nous présentons un algorithme génétique, que nous avons conçu pour résoudre les instances de grandes tailles. Pour tester les méthodes proposées nous développons des nouveaux ensembles de problèmes-tests pour le POPRL/PMETT, qui pourront être utilisés dans l'avenir pour mener des recherches dans ce domaine.

Dans le chapitre 5, nous définissons une nouvelle généralisation du problème d'ordonnancement de projet à ressources limitées en considérant l'objectif de minimiser le coût total d'exécution du projet. Celui-ci est composé de deux éléments principaux: le coût direct des ressources à utiliser et les frais généraux qui ne dépendent pas de la quantité de ressources allouées, mais qui sont proportionnels à la durée du projet. Ce problème, que nous appelons Problème général d'allocation et de nivellement des ressources d'un projet (PGANRP) est très commun en pratique, mais très peu de recherche est consacrée à ce problème. Dans un PGANRP, nous devons simultanément déterminer les quantités des ressources à allouer au projet au cours de son exécution et réduire la variabilité de l'utilisation des ressources au minimum tout en essayant de terminer le projet à une date de fin acceptable. Les quantités des ressources à allouer au projet devraient permettre l'accomplissement du projet à cette date et devient une limite sur la disponibilité de ces ressources durant toute l'exécution du projet. Nous proposons, une formulation mathématique du problème et deux approches de recherche dans le voisinage pour les instances de grandes tailles.

Abstract

The resource-constrained project scheduling problem (RCPSP) consists of scheduling a set of activities or tasks using one or more resource types available in limited quantity. In the standard version of this problem, pre-emption is not allowed, precedence relations are of the no-lag, finish-to-start type, and the used resources are renewable meaning that the same resources quantity are available each time period.

Solving this NP-hard optimization problem requires the determination of tasks execution date such that the project duration is minimized without using more than the available resource quantities.

In the first chapter of this thesis, the research problem and research objectives are presented while chapter 2 reviews the literature and contributions to the RCPSP and some of its extended versions. More than 125 published papers are reviewed. These contributions are divided into 4 groups of contributions. Those proposing optimal solution methods, those developing lower bounds on the project duration, those proposing heuristic and approximate solution methods, and those extending the standard version of the problem in order to make it closer to the real-life problem. This literature review revealed that very few contributions explicitly take into consideration the time required to transfer resources between execution sites of the project. Only three such contributions are published and none of these three publication deal with the case where tasks have more than one execution mode. This review also revealed that the large majority of the published research deals with the problem where the objective is to minimize the duration of the project. However, in almost all real-life situations, the objective is to minimise the total cost of the project. That is why this thesis is dedicated to solve these neglected extensions of the RCPSP.

Chapter 3 deals with the resource-constrained project scheduling problem with transfer times (RCPSPTT). Thus the goal in this case is to determine execution dates that allows for resources to be transferred between execution sites while respecting the precedence relations between these tasks as well as resources availability. A new genetic algorithm (GA) is developed to solve the RCPSPTT. This algorithm uses a new and efficient crossover operator. The chapter also study the performance of the proposed genetic algorithm and shows that it produces better results than the two previously published solution heuristics. It is to notice that the proposed GA considers renewable resource types and assume that tasks have only one execution mode.

Chapter 4 deals with the multi-mode resource-constrained project scheduling problem with transfer times (MRCPSPTT). Thus, it extends the problem studied in the previous chapter to the multi-mode case under the assumptions of no pre-emption while using renewable and non-renewable resources. This problem has never been the subject of any published research before. An integer linear mathematical formulation of the problem is given as well as new genetic algorithm is developed to solve it. An extensive empirical analysis is then presented and shows that the proposed GA is able to produce the optimal solution for 529 test instances with 10, 20 and 30 activities.

Chapter 5 introduces the generalized resource allocation and leveling problem (GRALP). This problem can be stated as follows. Given a set of project tasks to execute, their possible execution modes and precedence relations, an upper bound on the amount of resources that can be made available to the project, a project due date, the cost of resource utilization and the overhead cost; determine the execution date and mode for each task and the amount of resources to allocate to the project. The objective is to minimize the total project execution cost while respecting precedence constraints, project due date and not using more than the amount of resources that we decided to allocate to the project. Again we notice that this problem has never been the subject of any published research work. Chapter 5 presents an integer linear formulation of the problem, a neighborhood search solution heuristic, a genetic algorithm to solve it and an empirical experiment to evaluate the proposed heuristics showing the superiority of the proposed GA.

Finally, the conclusions of the thesis and some propositions for future research are given.

Le prophète, que la bénédiction et le salut soient sur lui, a dit «Lorsque le fils d'Adam meurt, toutes ses œuvres s'arrêtent hormis trois : une aumône, un savoir utile et un enfant pieux qui invoque Allah en sa faveur» (Rapporté par Mouslim)

Cette thèse est spécialement
dédiée à ma chère mère Abla.

Table des matières

Liste des tableaux

Liste des figures

Remerciements

Je tiens avant tout à exprimer ma profonde reconnaissance à mes directeurs de thèse Fayez Fouad Boctor et Jaques Renaud, qui ont assuré la direction de cette thèse. Leurs conseils judicieux, leur esprit critique et leur curiosité intellectuelle m'ont toujours inspirée et motivée pour mener les travaux de cette recherche. Qu'ils trouvent ici l'expression de mon plus profond respect.

J'exprime ma gratitude à Monsieur Adnène Hajji, du département d'opérations et systèmes de décision de l'Université Laval , Monsieur Daoud Ait-Kadi, du département de génie mécanique de l'Université Laval,et à Monsieur Rémy Glardon de l'École Polytechnique Fédérale de Lausanne en Suisse, qui ont accepté de faire partie du jury et pour l'honneur qu'ils me font en évaluant mes travaux.

Cette thèse a aussi été une épreuve personnelle, et j'exprime ma plus profonde reconnaissance a mes parents. Ils m'ont enseigné la droiture, le sérieux, la rigueur et l'ambition. Vos prières et vos conseils étaient pour moi une source d'inspiration, de réflexion et de motivation pour mener à terme cette thèse. J'ai voulu que vous soyez fiers de moi. Merci papa, merci maman, merci pour tout.

Je remercie également les membres de ma famille, ma chère sœur Amal et mes chers frères Doraid, Khalil et Chaouki, mes tantes et oncles, mes belles sœurs et mes beaux frères, mes neveux et nièces pour leur soutien indéfectible. Je remercie également ma belle mère pour ses multiples encouragements.

Je ne peux passer sous silence le soutien et l'aide de mes amies et amis, et mes collègues du département Opérations et systèmes de décision pour leurs encouragements. Je me garde de citer de nom afin de n'oublier personne.

J'exprime toute ma gratitude à mon cher époux Rachid et mes adorables enfants, Mayssa et Marouane. Je les remercie pour leur affection, leurs encouragements continuels et particulièrement pour le souffle d'optimisme avec lequel ils ont su me propulser pour que je puisse donner le meilleur de moi-même.

Avant-propos

Cette thèse de doctorat est la consolidation de mes travaux de recherche menés au sein du Centre interuniversitaire de recherche sur les réseaux d'entreprise, la logistique et le transport (CIRRELT) à la Faculté des sciences de l'administration de l'Université Laval. Les contributions de cette thèse sont présentées sous la forme de quatre articles scientifiques. Pour chacun des quatre articles qui composent cette thèse, je me suis impliquée en tant que chercheure principale. Plus précisément, mon rôle fut de proposer les problématiques de recherche, les approches de modélisation et les méthodes de résolution, le tout en concertation avec mon coauteur. Je me suis également occupée du travail de programmation, de l'expérimentation, de l'analyse des résultats et de la rédaction de la première version des articles.

Le premier article est un document de travail intitulé «*Revue de littérature du problème d'ordonnancement de projet à ressources limitées* (POPRL)». Il est soumis pour publication au Centre interuniversitaire de recherche sur les réseaux d'entreprise, la logistique et le transport (CIRRELT) à la Faculté des sciences de l'administration de l'Université Laval.

Le deuxième article intitulé «*An efficient genetic algorithm to solve the resource-constrained project scheduling problem with sequence dependent transfer times : The single mode case*» a été soumis pour publication à *European Journal of operational research* le 14 avril 2016. Deux versions préliminaires de cet article ont été arbitrées par les comités d'évaluation et publiées dans les actes des deux conférences internationales CIGI 2015 (*Conférence internationale de génie industriel*), 26 au 28 Octobre 2015, Québec et à PMS 2016 (*Project Management and Scheduling*), 19 au 22 avril 2016, Valence, Espagne.

Le troisième article s'intitule «*Multi-Mode, Resource-Constrained Project Scheduling with sequence dependent Transfer Times*». Une première version du papier a été acceptée et publié dans les actes de la conférence internationale PMS 2014 (Project Management and Scheduling), 30 mars au 2 avril 2014, Munich, Allemagne.

Le quatrième article intitulé «*The Generalized Resource Allocation and Leveling Problem in project scheduling*». La version incluse dans cette thèse a été arbitrée et acceptée par le comité d'évaluation et publiée dans les actes de la conférence internationale PMS 2016 (*Project Management and Scheduling*), Valence, Espagne.

Chapitre 1

Introduction

Le problème classique d'ordonnancement de projet à ressources limitées (POPRL) est devenu ces dernières années un problème de base dont plusieurs variantes font régulièrement l'objet de nouvelles publications. Il consiste en l'ordonnancement d'un ensemble de tâches, à l'aide d'un ou de plusieurs types de ressources renouvelables ou non renouvelables, disponibles en quantités limitées.

La résolution d'un POPRL a pour but la détermination des dates d'exécution des tâches en tenant compte des contraintes de préséance et de disponibilité des ressources et a comme objectif la minimisation de la durée totale du projet.

Le POPRL est un problème d'optimisation combinatoire de complexité NP-dur (Blazewicz et al. 1983). Il suscite beaucoup d'intérêt dans la communauté des chercheurs opérationnels, qui lui a consacré une littérature importante. Les contributions relatives à ce problème portent sur les méthodes exactes de résolution, la détermination de bornes inférieures sur la durée du projet et les méthodes heuristiques (approchées) de résolution.

Ces travaux s'appuient sur des outils décisionnels divers tels que la programmation linéaire en nombres entiers (PLNE), les méthodes d'évaluation et de séparation (en anglais « Branch-and-Bound »), les différents types de méthodes heuristiques incluant ce qui est appelé communément les méta-heuristiques. Des revues de la littérature scientifique sur les POPRL ont périodiquement été publiées, elles font le point sur les plus récents développements de la recherche. On y trouve des recommandations selon les tendances décelées et des propositions sur les avenues de recherche future.

Ce premier chapitre introduit cette thèse qui s'intitule Généralisations du problème d'ordonnancement de projet à ressources limitées. Pour mieux introduire le sujet, les définitions et la terminologie sont présentées dans la première section de ce chapitre. Ensuite les sections 1.2 et 1.3 discuteront la problématique, les objectifs et l'organisation de cette thèse.

1.1 Définition et terminologie

Dans cette section nous donnons la définition de certains termes se rapportant aux problèmes d'ordonnancement de projet à ressources limitées.

- **Projet** : Un projet est constitué d'un ensemble de tâches non répétitives, ayant des caractéristiques bien propres, qui sont exécutées à l'aide d'un ensemble de ressources.
- **Tâche** : Activité élémentaire faisant partie de l'ensemble des activités nécessaires à la réalisation du projet. Chaque tâche est définie par une durée d'exécution et les quantités de ressources requises à son exécution.
- **Période élémentaire** : C'est l'unité de mesure utilisée pour définir la durée des tâches et du projet. Elle peut être en heure, jours, mois etc.
- **Durée d'une tâche** : Temps nécessaire pour exécuter une tâche. Habituellement, elle est exprimée en nombre entier de périodes élémentaires.
- **Date de début** (fin) : Elle désigne un moment précis pour commencer (ou respectivement finir) une tâche. Ce moment correspondra habituellement au début (ou de fin) d'une période élémentaire.
- **Ordonnancement, plan ou calendrier d'exécution** : L'ensemble de toutes les informations nécessaires pour caractériser une solution du problème. Il peut être donné en spécifiant les dates de début des tâches, si cette information est suffisante pour définir la solution du problème.
- **Ressources** : Les moyens nécessaires pour l'exécution des tâches du projet. Dans la littérature, les trois catégories de ressources suivantes ont été définies :

  Ressources renouvelables : Une ressource est renouvelable si, après avoir été utilisée durant une période, elle redevient disponible pour la période suivante. On trouve dans cette catégorie des machines, des personnes, des ordinateurs, ...etc.

  Ressources non renouvelables : Une ressource est non renouvelable si la quantité disponible de cette ressource diminue au fur et à mesure de sa consommation pour l'exécution des tâches qu'elle participe à leur réalisation. C'est généralement le cas des matières premières, argent, etc.

  Ressources doublement restreintes : Une ressource est doublement restreinte, on dit aussi partiellement renouvelable, si sa disponibilité est limitée à la fois d'une période à l'autre et aussi dans sa quantité totale disponible pour le projet.

- **Contraintes de préséance** : Le terme «préséance» désigne un ordre entre deux tâches. Les contraintes de préséance sont définies par un ensemble P de paires d'indices des tâches telles que $(i, j) \in P$ signifie que la tâche j ne peut commencer qu'après la fin de la tâche i.

- **Contrainte de ressources** : L'exécution des tâches doit satisfaire différentes contraintes de ressources. Pour chaque type de ressources renouvelables $k \in K$, une quantité, désignée par R_k, spécifie la disponibilité de cette ressource renouvelable à chaque période t de la durée du projet. Un premier ensemble de contraintes s'assure que la quantité de ressources renouvelables utilisée par l'ensemble des tâches en cours d'exécution à chaque période ne dépasse pas la quantité disponible. De même, pour chaque type de ressources non renouvelables $w \in W$, une quantité totale maximale, désignée par N_w , est disponible et elle ne doit pas être dépassée pour toute la durée du projet.

1.2 Problématique de recherche

Le POPRL tel qu'il a été défini plus haut est une simplification des situations réelles, ce qui a incité les chercheurs ces dernières années à proposer plusieurs généralisations du problème classique. Les généralisations majeures du POPRL ont été essentiellement basées sur les éléments suivants : remplacer les contraintes de préséance telles que définies ci-haut par des contraintes de préséance plus générales, introduire des dates dues pour certaines tâches, permettre la préemption (l'arrêt puis la reprise de l'exécution) des tâches, considérer des tâches avec plusieurs modes d'exécution, prendre en considération des ressources autres que les ressources renouvelables classiques (ressources non renouvelables, ressources doublement restreintes, etc.), optimiser de nouvelles fonctions objectif autres que la durée totale du projet (maximisation de la valeur actuelle nette du projet, coût, robustesse de l'ordonnancement, nivellement de ressource, etc.), prendre en considération le temps de transfert des ressources entre les differents lieux d'exécution des tâches, prendre en considération le temps de mise en route avant l'exécution de certaines tâches, et considérer simultanément plusieurs projets.

On constate que malgré les efforts déployés pour définir des POPRL plus généraux, des contraintes continuent à être ignorées et forcent les gestionnaires dans la plus part des cas à se baser uniquement sur leur expérience pour ajuster manuellement les calendriers d'exécution, ou à utiliser des heuristiques conçues pour résoudre des versions simplifiées du problème.

L'objectif principal de cette thèse est de contribuer à modéliser et résoudre de nouvelles généralisations du problème d'ordonnancement de projet à ressources limitées. Les contributions suivantes sont proposées.

La première contribution consiste à developper de nouvelle méthodes pour résoudre le problème d'ordonnancement de projet à ressources limités dans l'objectif de minimiser la durée totale du projet, quand le transfert des ressources entre les sites d'exécution des tâches est non négligeable. Ce problème est désigné par le problème d'ordonnancement de projet à ressources limitées et des temps de transfert, la thèse traite à la fois le cas où chaque tâche à un seul mode d'exécution désigné par POPRLTT et la cas de plusieurs modes d'exécution sont possibles pour l'execution des tâches, designé par (POPRL/PMETT).

Le deuxième ensemble de contributions de cette thèse propose une nouvelle formulation mathématique et des méthodes de résolution au problème d'ordonnancement de projet avec plusieurs modes d'exécution par tâche dans l'objectif de minimiser le coût total du projet. Le coût du projet est la somme de deux coûts : le coût d'utilisation des ressources et les frais généraux associés à chacune des périodes de l'exécution du projet. Le coût de disponibilité d'une ressource est le produit de trois éléments: la quantité de cette ressource que nous allouons au projet, la durée du projet et le coût unitaire de la ressource. Ce problème sera appelé le Problème Général d'Allocation et de Nivellement des Ressources d'un Projet (PGANRP).

Pour résoudre ces problèmes complexes, nous proposons des algorithmes génétiques qui offrent un niveau de performance supérieur. Pour tester les méthodes proposées nous developpons des nouveaux ensembles de problèmes-tests pour le POPRLTT, POPRL/PMETT et le PGANRP, qui pourront être utilisés dans l'avenir pour stimuler la recherche dans ce domaine.

1.3 Structure du document

Suite à une introduction, le document inclue cinq autres chapitres. Le chapitre 2 présente une revue de littérature des POPRL. Les chapitres 3, 4 et 5 traitent respectivement les trois généralisations du POPRL que nous aborderons dans le cadre de cette thèse, soient le POPRLTT, le POPRL/PMETT et le PGANRP. Chaque chapitre situe le problème traité par rapport à la littérature déjà existante, présente une formulation mathématique du problème qui peut être utilisée pour résoudre les problèmes de petite taille, et propose des méthodes heuristiques appropriées au problème.

Le dernier chapitre conclut cette thèse. Il rappelle les grandes lignes de l'étude, précise les contributions de cette recherche ainsi que les avenues de recherches futures.

Chapitre 2

Document de travail : Revue de littérature du problème d'ordonnancement de projet à ressources limitées (POPRL)

Roubila Lilia Kadri et Fayez Fouad Boctor
Centre interuniversitaire de recherche sur les réseaux d'entreprise, la logistique et le transport (CIRRELT)
Département Opérations et Système de Décision, Université Laval, 2325 rue de la Terrasse, Québec, Canada, G1V0A6

Résumé

Ce document présente une revue de littérature du problème d'ordonnancement de projet à ressources limitées (POPRL) Il s'agit d'un problème d'optimisation combinatoire de complexité NP-dur (Blazewicz et al., 1983). La résolution d'un POPRL a pour but la détermination des dates d'exécution des tâches en tenant compte des contraintes de préséance et de disponibilité des ressources et ayant comme objectif la minimisation de la durée totale du projet.

L'aspect pratique de ce problème dans des contextes industriels divers a conduit à de nombreuses recherches. Les contributions relatives à ce problème portent sur les méthodes exactes de résolution, la détermination de bornes inférieures sur la durée du projet et les méthodes heuristiques (approchées) de résolution. Des généralisations du problème classique afin d'avoir une meilleure représentation des situations partiques sont également discutées.

L'objectif de cette revue est d'effectuer une mise à jour de la recherche disponible dans le domaine, consolider et classifier les travaux de recherche publiés. Aussi, elle étudie plus en détail le problème d'ordonnacement avec plusieurs modes d'exécution et enfin identifie des perspectives de recherches futures qui ont encore besoin d'être explorées.

2.1 Introduction

Le problème d'ordonnancement de projet à ressources limitées (POPRL) est devenu ces dernières années un problème de référence en gestion de projet (Hartmann (2012)).

Il s'agit de déterminer les dates d'exécution des tâches dans l'objectif de minimiser la durée totale du projet, tout en respectant les contraintes de disponibilité des ressources.
Trois types de ressources sont identifiés dans la littérature : les ressources renouvelables (où la quantité disponible est la même à chaque période), les ressources non renouvelables (où la quantité totale que le projet peut utiliser est limitée), et les ressources doublement restreintes (où il y a une limite à la fois sur la quantité utilisable dans chaque période et sur la quantité totale utilisable par le projet).

Le POPRL classique se caractérise habituellement par :

- Plusieurs types de ressources renouvelables qui sont disponibles à chaque période où pour chaque type $\forall k \in K$, une quantité de Q_k unités est disponible à chaque période.
- Un ensemble de tâches noté J, et chaque tâche ($\forall i \in J$) a une durée d_i et requiert pour son exécution un nombre r_{ik} d'unités de la ressource de type k; $\forall k \in K$.
- Un ensemble de relation de préséance P, où chaque relation $(i, j) \in P$ signifie que j ne peut commencer qu'après la fin de l'exécution de i.
- Une fois commencée, la tâche ne doit pas être interrompue (préemption non permise).

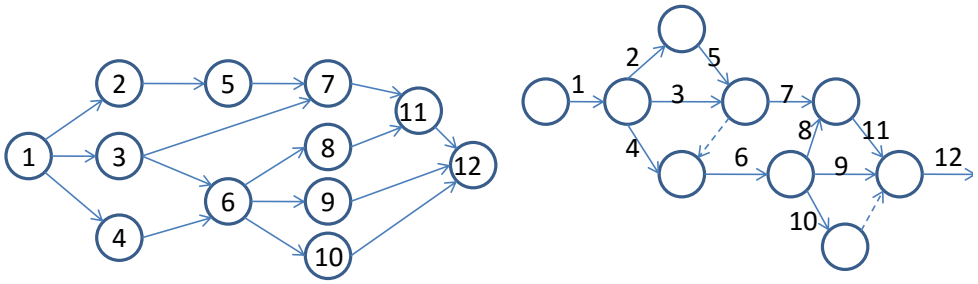
Deux types de graphes sont utilisés pour représenter un projet, ses tâches et les relations de préséance entre celles-ci. Dans le premier, appelé graphe sommet-tâche, les tâches sont représentées par les sommets du graphe et les arcs indiquent les relations de préséance. Par contre, dans le deuxième, appelé graphe arc-tâche, les arcs représentent les tâches et les sommets indiquent les relations de préséance de la façon suivante : toutes les tâches (arcs) dont le sommet final est le sommet i, sont des prédécesseurs immédiats de toutes les tâches dont i est le sommet initial. Le graphe sommet-tâche est le plus utilisé vu sa simplicité. En plus, représenter un projet par un graphe arc-tâche nécessite souvent l'ajout de tâches fictives.

Exemple : Klein (1999)

Les données de l'exemple de Klein (1999) sont présentées dans le Tableau 2.1 et la Figure 2.1 illustre les deux types de représentation graphique correspondantes à cet exemple. Les tâches fictives dans le graphe arc-tâche sont représentées par les arcs en pointillés.

Tâche	Prédécesseurs immédiats	Durée	Besoin en nombre d'unités de la ressource
1	-	0	0
2	1	3	2
3	1	5	3
4	1	1	3
5	2	3	1
6	3,4	2	1
7	3,5	4	2
8	6	5	3
9	6	6	1
10	6	4	1
11	7,8	4	1
12	9, 10,11	0	0

Tableau 2.1: Données de l'exemple de Klein (1999)



(a) Graphe sommet-tâche (b) Graphe arc-tâche

Figure 2.1: Les représentations graphiques de l'exemple de Klein (1999)

2.2 Modélisation

La résolution exacte du POPRL en faisant appel à la programmation linéaire en nombre entier (PLNE) a fait l'objet de plusieurs travaux. De nombreuses formulations mathématiques ont été développées, passant de la formulation en nombre entier de Pritsker et al. (1969), à la

formulation de Klein (1999). Voir également le chapitre six du livre de Demeulemeester and Herroelen (2006) et le chapitre trois de Klein (1999) pour plus de détails.

La principale difficulté dans la modélisation des POPRL par la PLNE réside dans la modélisation du problème au moyen d'inégalités linéaires des contraintes de ressources du problème. On distingue principalement les deux types de formulations suivantes :

- **Des formulations à temps continu** : ces formulations introduisent des variables de décision binaires indicées y_{ij} tel que $y_{ij} = 1$, si et seulement si la tâche i précède la tâche j, et 0 sinon. Parmi ces formulations on peut citer celles de Balas (1985), celle de Olaguíbel and Goerlich (1993), et celle d' Artigues et al. (2003).
- **Des formulations à temps discrétisé** : dans ce type de formulation, les variables de décision sont des variables binaires ou entières indicées à la fois par la tâche et la période, noté x_{it}. Parmi celles-ci on cite la formulation classique de Pritsker et al. (1969), de Mingozzi et al (1998), et de Klein (1999).

Deux formulations dans chaque classe sont détaillées dans les quatre paragraphes suivants. Il s'agit de la formulation à temps continu d' Olaguíbel and Goerlich (1993), et celle d' Artigues et al. (2003). Les formulations à temps discrétisé sont celles de Pritsker et al. (1969) et de Mingozzi et al. (1998).

2.2.1 Formulation à temps continu basée sur les ensembles critiques

Pour exprimer les contraintes de ressources dans un POPRL, Olaguíbel and Goerlich (1993) ont intégré la notion d'ensemble critique minimal à la formulation disjonctive de Balas (1985). Un sous-ensemble de tâches C est dit critique si la somme des consommations des tâches de l'ensemble C excède la quantité totale disponible pour au moins une ressource, et il est minimal si aucun sous-ensemble de C n'est critique. On note C_m l'ensemble des sous-ensembles critiques minimaux. Aussi, une paire de tâches i et j est dite incompatible si la somme des ressources requises pour cette paire dépassent la disponibilité pour au moins un type de ressource.

Pour cette formulation nous ajoutons deux tâches fictives notées 1 et F (c'est-à-dire, d'une durée nulle, et qui ne requièrent aucune ressource pour leur exécution) : la tâche 1 qui précède toutes les tâches sans prédécesseurs et la tâche F qui est le successeur immédiat de toutes les tâches sans successeurs.

Le modèle est défini par des variables binaires y_{ij} et des variables continues S_i où :

$y_{ij} = \begin{cases} 1, & \text{si } i \text{ précède } j \\ 0, & \text{sinon.} \end{cases}$

S_i : est la date de début de l'exécution de la tâche i, $\forall i \in J \cup \{1, F\}$, $S_i \geq 0$.

Le modèle est :

$$\text{Miminize } S_F \tag{2.1}$$

Sous les contraintes :

$$y_{ij} = 1 \qquad \forall (i,j) \in P \tag{2.2}$$

$$y_{ij} + y_{ji} \leq 1 \qquad \forall i,j \in J \cup \{1\} \tag{2.3}$$

$$y_{ih} \geq y_{ij} + y_{jh} - 1 \qquad \forall i,j,h \in J \cup \{1\} \tag{2.4}$$

$$y_{ij} + y_{ji} = 1 \qquad \forall (i,j) \text{ une paire de tâches incompatible} \tag{2.5}$$

$$S_j - S_i \geq -M + (d_i + M) y_{ij} \qquad \forall i,j \in J \cup \{1\} \tag{2.6}$$

$$\sum_{(i,j) \in C \text{x} C} y_{ij} \geq 1 \qquad \forall C \in C_m \tag{2.7}$$

$$y_{ij} \in \{0,1\} \qquad \forall i,j \in J \cup \{1\} \tag{2.8}$$

$$S_i \geq 0 \qquad \forall i \in J \cup \{1, F\} \tag{2.9}$$

La minimisation de la durée totale du projet est donnée par l'expression (2.1), les contraintes (2.2) expriment les relations de préséance initiales du problème. Les contraintes (2.3) imposent que toutes les paires de tâches (i,j), soient tel que i précède j, ou bien j précède i, ou i et j s'exécutent en parallèle. Les contraintes (2.4) sont des contraintes de transitivité qui conjointement avec les contraintes (2.3), éliminent les cycles dans le graphe. Les contraintes (2.5) imposent que la relation entre chaque paire de tâches incompatibles (i,j) soit tel que i précède j ou l'inverse. Les contraintes (2.6) lient les deux types de variables du modèle : pour une valeur suffisamment grande de M, les contraintes imposent que si $y_{ij} = 1$, l'exécution de la tâche j doit commencer après la date de fin de la tâche i. L'ensemble des contraintes (2.7) imposent qu'au moins une contrainte de préséance soit créee dans chaque ensemble critique minimal.

2.2.2 Formulation à temps continu basée sur les flux

Toujours inspiré par les travaux de Balas (1985) et sur la base de la formulation d' Olaguíbel and Goerlich (1993), Artigues et al. (2003) ont opté pour l'utilisation du concept des flux dans les réseaux pour la modélisation des besoins en ressources au lieu du concept des ensembles critiques. On note qu'ils ont obtenu ainsi un modèle plus compact. Ce modèle définit les trois types de variables suivants :

- Variables continues usuelles de date de début S_i (pour définir la date de début de chaque tâche du projet)
- Variables séquentielles binaires y_{ij} (permettant de définir un ordre complet sur l'ensemble des tâches)
- Variables continues dites variables de flux f_{ijk}, où f_{ijk} représente la quantité de la ressource de type k directement transmise d'une tâche i (à la fin de son exécution) vers une tâche j (au début de son exécution).

On suppose qu'au début du projet les ressources sont toutes disponibles et stockées au niveau de la tâche fictive de début du projet. Par la suite, l'exécution des premières tâches débute par l'utilisation des ressources disponibles, et une fois terminées, elles transfèrent leurs ressources aux tâches qui les succèdent. Cette opération se répète jusqu'à ce que les dernières tâches à exécuter les transmettent à leur tour à la tâche fictive F de la fin du projet; cela se traduit par $r_{1k} = r_{Fk} = Q_k, \forall k \in K$.

Le modèle est donné par :

$$\text{Miminize } S_F \tag{2.10}$$

Sous les contraintes :

$$y_{ij} = 1 \qquad \forall (i,j) \in P \tag{2.11}$$

$$S_j - S_i - M.y_{ij} \geq d_i - M \qquad \forall i \in J \cup \{1\}, \forall j \in J \cup \{F\} \tag{2.12}$$

$$f_{ijk} \leq \min(r_{ik}, r_{jk}).y_{ij} \qquad \forall i \in J \cup \{1\}, \forall j \in J \cup \{F\}, \forall k \in K \tag{2.13}$$

$$\sum_{j \in J \cup \{F\}} f_{ijk} = r_{ik} \qquad \forall i \in J \cup \{1\}, \forall k \in K \tag{2.14}$$

$$\sum_{i \in J \cup \{1\}} f_{ijk} = r_{jk} \qquad \forall j \in J \cup \{F\}, \forall k \in K \tag{2.15}$$

$$f_{ijk} \geq 0 \qquad \forall i \in J \cup \{1\}, \forall j \in J \cup \{F\}, \forall k \in K \tag{2.16}$$

$$S_i \geq 0 \qquad \forall i \in J \cup \{1, F\} \tag{2.17}$$

$$y_{ij} \in \{0,1\} \qquad \forall i \in J \cup \{1\}, \forall j \in J \cup \{F\} \tag{2.18}$$

La fonction objectif donnée par l'expression (2.10) est la minimisation de la date de début de la dernière tâche (tâche F) du projet. Les équations (2.11) expriment les relations de préséance entre les tâches du projet. Les contraintes (2.12) permettent de lier les deux variables y_{ij} et S_i pour chaque paire de tâches i et j. Les contraintes (2.13) lient les variables de flux f_{ijk} aux variables séquentielles y_{ij}, via la disponibilité des ressources. Ainsi, si i précède j, alors le flux maximal de ressources envoyé de i vers j correspond au minimum de la consommation r_{ik} de la tâche prédécesseur i et de la consommation r_{jk} de la tâche successeur j, ceci, quel

que soit la ressource k. Les contraintes (2.14) et (2.15) assurent la conservation des flux. Les contraintes (2.16) et (2.17) imposent que les variables de flux et les dates de début des tâches soient des variables continues. Les contraintes (2.18) quant à elles imposent que les variables y_{ij} soient des variables binaires.

2.2.3 Formulation à temps discrétisé de Pritsker et al. (1969)

La première formulation à temps discrétisé a été proposée par Pritsker et al. (1969). Dans ce type de formulation, les variables de décision sont des variables binaires indexées sur un horizon de temps discrétisé de T périodes, telle que :

$x_{it} = \begin{cases} 1, & \text{si l'exécution de la tâche } i \text{ commence au début de la période } t \\ 0, & \text{sinon.} \end{cases}$

L'ensemble des tâches en cours d'exécution à la période t se définit alors par :

$$A_t = \{\forall i \in J \cup \{1, F\} \setminus \sum_{s=t}^{t+d_i-1} x_{is} = 1\}$$

Le modèle est donc donné par :

$$\text{Miminize} \sum_{t=1}^{T} t x_{Ft} \tag{2.19}$$

Sous les contraintes :

$$\sum_{t=1}^{T} x_{it} = 1 \qquad \forall J \cup \{1, F\} \tag{2.20}$$

$$\sum_{t=1}^{T} t\,(x_{jt} - x_{it}) \geq d_i \qquad \forall (i,j) \in P \tag{2.21}$$

$$\sum_{i \in J} r_{ik} \sum_{\tau=t-d_i+1}^{t} (x_{i\tau}) \leq Q_k \;\; t = 1, ..., T \qquad \forall k \in K \tag{2.22}$$

$$x_{it} \in \{0, 1\} \qquad \forall i \in J \cup \{1, F\}, t = 1, ..., T \tag{2.23}$$

La fonction objectif donnée par (2.19) représente la minimisation de la durée totale du projet. Les contraintes (2.20) assurent que chaque tâche doit avoir une seule date de début (la préemption est non permise). Les contraintes (2.21) expriment les relations de préséances entre les différentes tâches, en stipulant que si la paire de tâches (i, j) appartient à l'ensemble P, alors la date de début de la tâche j sera ultérieure ou égale à la date de fin de la tâche j. Les contraintes (2.22) représentent les contraintes de ressources; elles assurent que pour chaque type de ressource et à chaque période de l'horizon temporel, la somme des consommations des tâches en cours d'exécution ne dépasse pas la quantité des ressources disponibles. Les contraintes (2.23) assurent que les variables x_{it} prennent des valeurs binaires.

2.2.4 Formulation en temps discrétisé basée sur les ensembles admissibles

Cette formulation a été proposée par Mingozzi et al. (1998) elle est basée sur le principe que toute solution réalisable peut être caractérisée par une séquence, qu'on peut noter $s = (Adm_{l_1}, \ldots, Adm_{l_T})$, où Adm_{l_t} est un ensemble admissible de tâches ($Adm_{l_t} \subseteq V$), incluant toutes les tâches exécutées à la période t. Il est dit admissible, car les tâches qui le composent peuvent s'exécuter simultanément en regard des contraintes de préséance et de ressources.

Avant de présenter le modèle, on définit les deux ensembles $IndA$ et $IndA_i$ comme suit :

- $IndA$: Ensemble des indices l de tous les ensembles admissibles de J, (toutes les tâches incluses dans un ensemble admissible doivent vérifier les contraintes de préséance et de ressource).
- $IndA_i \subseteq IndA$: Ensemble de tous les indices des ensembles admissibles incluant la tache i.

Le modèle définit les deux ensembles de variables de décision binaires suivants : x_{it} et z_{lt}. Les variables x_{it} sont definies comme dans la formulation de (Pritsker et al., 1969). Le deuxième ensemble de variable z_{lt} est définie pour tout ensemble admissible Adm, et sur toutes les périodes de l'horizon temporel T tel que :

$$z_{lt} = \begin{cases} 1, & \text{si toutes les tâches du sous ensembe admissible } l \text{ sont exécutées à la période } t \\ 0, & \text{sinon.} \end{cases}$$

Le modèle est identique au précédent, excepté les contraintes de ressource, qui sont remplacées par les contraintes suivantes :

Sous les contraintes :

$$\sum_{l \in IndA_i} \sum_{t \in I_i} z_{lt} = d_i \qquad \forall J \cup \{1, F\} \qquad (2.24)$$

$$\sum_{l \in IndA_i} z_{lt} \leq 1 \qquad t = 1, ..., T \qquad (2.25)$$

$$x_{it} \geq \sum_{l \in IndA_i} z_{lt} + \sum_{l \in IndA_i} z_{l(t+1)} \qquad \forall i \in J \cup \{1, F\}, t = 1, ..., T \qquad (2.26)$$

$$z_{lt} \in \{0, 1\} \qquad \forall l \in IndA, t = 1, ..., T \qquad (2.27)$$

où I_i est la fenêtre de de temps où la tâche i peut etre exécutée.
Les contraintes (2.24) garantissent que toute tâche i s'exécute pendant d_i périodes situées entre la date de début au plus tôt et au plus tard (ces dates sont définies par la méthode du chemin critique). Les contraintes (2.25) imposent qu'au plus un seul ensemble admissible soit

en cours d'exécution à chaque période t. Les contraintes (2.26) imposent que la variable de décision x_{it}, prenne la valeur 1 si une tâche i est en exécution à la période t, et qu'elle n'est pas dans l'ensemble admissible en exécution à la période t-1. Les contraintes (2.27) quant à elles, imposent que les variables z_{lt} prennent des valeurs binaires.

Les différentes formulations mathématiques du POPRL souffrent soit de leur taille (nombre de variables ou de contraintes), soit de la faiblesse de leurs relaxations. L'inconvénient de la formulation de Mingozzi et al. (1998) est qu'elle génère un nombre exponentiel de variables binaires; son utilisation se limite donc à des instances de très petite taille. Mais à partir de celle-ci, les auteurs ont été capables de définir des bornes inférieures au POPRL d'une assez bonne qualité.

La section suivante du présent document propose un survol des différentes techniques développées dans la littérature pour définir des bornes inférieures au POPRL.

2.3 Bornes inférieures pour le POPRL

Les bornes inférieures sont d'une très grande utilité dans les méthodes de séparation et d'évaluation ainsi que dans les méthodes de résolution approchées. Mais le choix d'une méthode pour obtenir une borne implique un compromis entre la qualité de la borne et son temps de calcul, ce qui a incité les chercheurs à développer plusieurs techniques pour définir des bornes inférieures pour le POPRL. Ces techniques consistent à autoriser la violation de certaines contraintes du problème ou à la réduction du problème initial de façon à ce que la solution donne une borne inférieure pour la valeur optimale du problème initial. Le Tableau 2.2 ci-dessous classifie les différentes approches trouvées dans la littérature en deux grandes classes (voir Klein (1999) et Demeulemeester and Herroelen (2006), celles dites directes, elles peuvent être simples ou complexes, et celles dites indirectes.

Dans les trois sections qui suivent, nous présenterons respectivement des bornes inférieures issues d'une technique directe simple, d'une technique directe complexe et d'une technique indirecte.

2.3.1 Borne inférieure du chemin critique

La borne inférieure la plus simple et la plus utilisée est la borne du chemin critique. Elle consiste à relaxer les contraintes de ressources du POPRL. La résolution du problème obtenu se résume tout simplement à chercher le chemin le plus long dans un graphe $G = (J \cup \{1, F\}, P)$. Ce chemin porte l'appellation de chemin critique, ainsi sa longueur définie une borne inférieure sur la solution optimale du POPRL.

Différentes généralisations sur le chemin critique ont permis d'améliorer cette borne. Ainsi, Stinson et al. (1978) proposent la borne définie par la fonction (2.28) ci-dessous:

$$CPM + \max_i(d_i - e_i) \tag{2.28}$$

où CPM est la longueur du chemin critique et $(d_i - e_i)$ est la longueur d'une partie de la tâche i qui ne peut pas être exécutée en parallèle avec le chemin critique sans violer la contrainte de disponibilité des ressources.

2.3.2 Borne inférieure de Mingozzi et al (1998)

Mingozzi et al. (1998) ont proposé cinq bornes inférieures pour le POPRL. Celles-ci dominent la borne du chemin critique, et elles sont plus serrées que la borne qui a été définie par Stinson et al. (1978). Elles ont toutes été obtenues à partir de la formulation du POPRL basée sur les ensembles admissibles, et par l'utilisation de différents types de relaxations (relaxation linéaire, relaxation des contraintes de préséance, ...etc.). L'idée de base est de trouver un sous ensemble de tâches dont aucune paire ne peut être exécutée en parallèle. La somme des durées de ces tâches constitue une borne inférieure sur la durée du projet. Les résultats expérimentaux démontrent que les bornes obtenues présentent un bon compromis qualité/temps de calcul.

2.3.3 Borne d'amélioration destructive de Klein and Scholl (1999)

Dans l'approche dite d'amélioration destructive, la borne inférieure est la plus petite valeur de T pour laquelle on ne peut plus prouver que le problème est réalisable.

2.4 Méthode exactes de résolution du POPRL

Les problèmes d'ordonnancement de projet à ressources limitées sont des problèmes NP-durs (voir Blazewicz et al. (1983)). Les méthodes exactes de résolution de ce problème peuvent se définir en trois classes (voir Kolisch (2000)) :

1. Méthodes basées sur la programmation en nombres entiers;
2. Méthodes basées sur la programmation dynamique;
3. Méthodes de recherche arborescente.

Patterson (1984) a souligné dans une étude comparative de ces trois méthodes, que les méthodes de recherche arborescente sont les méthodes les plus performantes et les plus efficaces.

Bornes directes
Simples
1. Borne basée sur le chemin critique - le chemin critique : CPM - séquence critique : Stinson et al. (1978) - séquences étendues : Demeulemeester and Herroelen (1992) 2. Borne basée sur les ressources - la quantité de travail maximale - la quantité de travail étendue 3. Borne définie par les n-machine : Carlier and Latapie (1991) 4. Borne définit par la résolution d'un problème du "node packing" de base Mingozzi et al. (1998) 5. Borne définie par une version améliorée du problème "node packing" Mingozzi et al. (1998) 6. Borne définie par une version des n-machine : Mingozzi et al. (1998) 7. Borne basée sur les sous projet : Klein and Scholl (1999)
Complexes
1. Borne définie par la relaxation linéaire Christofides et al. (1987) 2. Bornes définie par relaxation Lagrangienne Christofides et al. (1987) 3. Borne définie par une approche de "set covering" :Mingozzi et al. (1998)
Bornes indirectes
Klein and Scholl (1999)

Tableau 2.2: Classification des techniques pour trouver des bornes inférieures

Le principe général de ces méthodes est de construire un arbre de recherche afin d'explorer implicitement l'espace de recherche, ce qui revient à partitionner le problème en sous problèmes pour pouvoir ensuite éliminer ceux qui sont moins intéressants en se basant sur soit des bornes inférieures soit des règles de dominance. L'opération de partitionnement s'appelle la séparation ou le branchement et l'élimination est la résultante d'une évaluation.

Dans la littérature, plusieurs techniques d'évaluation et de séparation ont été proposées pour le POPRL. Elles se distinguent principalement par le choix du schéma de branchement (la séparation de l'espace de recherche) et le choix de la méthode d'évaluation (bornes inférieures, règle de dominance). On distingue principalement les trois types de branchement suivants :

- Branchement d'une seule tâche par nœud;
- Branchement d'un sous-ensemble de tâches par nœud;
- Branchement basé sur la fixation des paires de disjonctions et des tâches pouvant être exécutées en parallèle.

Les trois sections qui suivent présentent les grandes lignes dans chaque type de branchement. Il s'agit respectivement de l'algorithme de Patterson et al. (1989) basé sur le branchement d'une seule tâche par nœud, de l'algorithme de Demeulemeester and Herroelen (1992) basé sur un ensemble de tâches par nœud, et enfin, l'algorithme basé sur les schémas d'ordonnancement de Brucker et al. (1998).

2.4.1 Algorithme de Patterson et al. (1989)

Cet algorithme a été proposé par Patterson et al. (1989). Il a ensuite été révisé et généralisé dans d'autres travaux pour traiter d'autres variantes du problème. L'idée de base de cet algorithme de séparation et d'évaluation est l'énumération (implicite) de toutes les séquences réalisables du problème. Une séquence, appelée aussi liste, est dite réalisable si l'ordre attribué aux tâches respecte les contraintes de préséance, c'est-à-dire, qu'aucune tâche n'est listée avant l'un de ses prédécesseurs.

À l'étape initiale de l'algorithme, la tâche fictive 1 du début du projet est planifiée à une date de début égale à zéro. Ensuite à chaque niveau dans l'arbre, l'algorithme détermine l'ensemble des tâches déjà planifiées et l'ensemble des tâches éligibles (ceux dont les prédécesseurs ont déjà été planifiés). Une tâche éligible est ensuite sélectionnée, et sa date de début au plus tôt est calculée en tenant compte des contraintes de ressources et de préséance. L'algorithme passe ensuite au niveau suivant. Si la dernière tâche fictive du projet F est planifiée, alors un ordonnancement réalisable est complété. Dans ce cas, un retour vers les niveaux précédents est

prévu et la prochaine tâche éligible non encore testée est choisie. Si toutes les tâches éligibles ont été testées, l'algorithme remonte encore d'un niveau dans l'arbre.

2.4.2 Algorithme de Demeulemeester and Herroelen (1992)

Cet algorithme a été initié par Christofides et al. (1987) et il a été par la suite révisé et amélioré par Demeulemeester and Herroelen (1992). Il est basé sur la notion de « l'alternative de retard minimal (Minimal Delaying Alternative) » où un MDA est lié au concept d'ensemble réalisable maximal (ensembles maximal de tâches qui peuvent être exécutées simultanément) et au concept d'ensemble interdit IN (ensembles des tâches qui ne sont pas reliées par des relations de préséance mais ne pouvant pas être exécutées simultanément sans provoquer un conflit de ressource). Un MDA est un sous ensemble réalisable maximal d'un ensemble interdit.

À l'étape initiale de l'algorithme, la tâche fictive 1 du début du projet est planifiée à une date de début égale à zéro. Ensuite chaque niveau de l'arbre est associé à un instant t_g (dit instant de décision) auquel toutes les tâches éligibles (dont tous leurs prédécesseurs sont déjà planifiés et se terminent tous avant t_g) sont planifiées si elles ne provoquent pas un conflit de ressources, et ainsi un seul nœud est rajouté à l'arbre. Dans le cas contraire, l'algorithme énumère tous les MDA possibles (une branche pour chaque MDA).

Demeulemeester and Herroelen (1992) ont utilisé cet algorithme pour résoudre les instances de Patterson (1984). Les résultats expérimentaux obtenus restent compétitifs jusqu'à ce jour.

2.4.3 Algorithme de Brucker et al. (1998)

Ce type d'algorithme a été développé par Brucker et al. (1998). Il est basé sur une représentation assez particulière, puisqu'à chaque nœud de l'arbre correspond un schéma d'ordonnancement, noté (C, D, N, L), où C est l'ensemble des paires de tâches en conjonction (si $(i, j) \in C$, alors i précède j), D est l'ensemble des paires en disjonction ($(i, j) \in D$, donc la somme des consommations des tâches i et j excède la quantité totale disponible pour au moins une ressource, mais l'ordre d'exécution des deux tâches est encore inconnu), N inclut toutes les tâches qui peuvent s'exécuter en parallèle pour au moins une période de l'horizon temporel, et L est l'ensemble des tâches flexibles.

L'algorithme est initialisé par un schéma initial $(C_0, D_0, \emptyset, L)$ tel que C_0 est défini par toutes les contraintes de préséance du départ du problème, D_0 est l'ensemble des paires de tâches (i, j) tel que $(r_{ik} + r_{jk} \geq Q_k)$ pour au moins une ressource $\forall k \in K$, et L groupe toutes les paires de tâches qui ne sont pas liées par une relation de disjonction ou de conjonction.

L'objectif du branchement est de remplacer les relations de flexibilité par des relations de conjonction ou des relations parallèles. Et à partir du schéma exprimé par $(C, D, N, \emptyset)$, il est

possible de trouver en un temps polynomial le meilleur ordonnancement correspondant définit par $(C, \emptyset, N, \emptyset)$. L'algorithme effectue ensuite des retours en arrière pour améliorer la solution courante.

2.5 Méthodes heuristiques

Les méthodes heuristiques pour la résolution des problèmes combinatoires de type NP-dur sont d'une grande utilité puisqu'elles permettent de donner de bonnes solutions dans des temps d'exécution acceptables. C'est pour cette raison qu'elles ont été d'une grande popularité dans la résolution des POPRL. Elles ont été classées par Hartmann (2012) comme suit :

1. Les heuristiques constructives basées sur des règles de priorité;
2. Les méthodes de recherche dans le voisinage;
3. Les techniques d'énumération tronquée;
4. Les approches basées sur les arcs disjonctifs.

Ces méthodes sont abordées dans chacune des sections suivantes.

2.5.1 Heuristiques constructives basées sur des règles de priorité

Ce type de méthodes a été d'une grande popularité dans le domaine de l'ordonnancement de projet. Elles sont des méthodes intuitives, faciles à mettre en œuvre et très rapides. Elles sont constructives parce que la solution est obtenue d'une façon progressive, en prenant une décision à la fois. À chaque étape la tâche ayant la plus grande priorité est planifiée; si deux tâches ou plus sont à égalité, une deuxième règle de priorité doit être utilisée.

Ces heuristiques se divisent en deux groupes : les heuristiques sérielles et les heuristiques parallèles. Pour les heuristiques sérielles, les indices de priorité sont fixés avant de commencer à planifier les tâches tandis que pour les heuristiques parallèles, l'indice de priorité est recalculé après chaque décision. Les pseudo codes de ces deux types d'heuristiques sont donnés dans la Figure 2.2 et 2.3 respectivement (Kolisch (1996)).

On distingue aussi les heuristiques à passe unique et celles à passes multiples. Les heuristiques à passes multiples construisent à chaque passe une nouvelle solution soit en utilisant une autre règle de priorité soit en inversant le sens d'exécution de l'heuristique tel que décrit dans les trois paragraphes suivants.

Établir l'ordre de priorité de toutes les tâches à planifier

Tant qu'il reste encore des tâches non planifiées :

1. Planifier chaque tâche selon son ordre de priorité à la date la plus proche de sa date de début au plus tôt et qui respecte la disponibilité des ressources (vérifier la disponibilité des ressources à chaque périodes de sa durée d'exécution)
2. Mettre à jour la disponibilité des ressources en conséquence et mettre à jour la liste des tâches non planifiées.

Figure 2.2: Pseudo code des heuristiques dites sérielles

1. **Heuristiques de construction « en avant »** : dans ce type de construction, les tâches sont planifiées après leurs prédécesseurs et on détermine leur dates de début en fonction des dates de fin de ces prédécesseurs. On commence donc par l'ordonnancement des tâches sans prédécesseurs en leur accordant la date de début au plus tôt du projet et on continu la procédure jusqu'à ce qu'on atteigne les tâches sans successeurs.

2. **Heuristiques de construction « en reculant »** : dans ce cas, on commence par planifier les tâches sans successeurs en leur donnant une date de fin quelconque. Leurs dates de début sont ensuite calculées et utilisées comme date de fin pour leurs prédécesseurs. À la fin, toutes les dates sont décalées de façon à commencer le projet à sa date de début au plus tôt.

3. **Heuristiques de construction bidirectionnelle** : les deux techniques précédentes sont alternativement exécutées pour donner deux ordonnancement (un ordonnancement en avant et un en reculant). La meilleure des deux solutions est ensuite retenue.

   Dans la littérature on constate qu'il existe une grande panoplie de règles (voir Boctor (1990),Olaguibel and Goerlich (1993) et Kolisch (1996) pour plus de détail). Le Tableau 2.3 ci-dessous résume quelques unes d'entre elles selon la classification qui a été proposée par Klein (1999) :

   a. Des règles basées sur les caractéristiques des tâches,

   b. Des règles basées sur le chemin critique,

   c. Des règles basées sur les ressources,

   d. Des règles hybrides.

Tant qu'il reste encore des tâches non planifiées :

1. Allez à la période la plus proche où il y a des ressources encore disponibles;
2. Etablir la liste des tâches qui sont éligibles à cette période (c'est à dire ceux pour lesquels toutes les ressources encore disponibles sont suffisantes pour pouvoir les exécuter et tous leurs prédécesseurs immédiats sont déjà planifiées et déjà terminées à cette période.
3. Établir l'ordre de priorité des tâches éligibles.
4. Si la liste est vide revenir à l'étape 1
5. Tant qu'il reste encore des ressources disponibles, planifier les tâches éligibles selon leurs ordre de priorité.
6. Mettre à jour la disponibilité des ressources en conséquence et mettre à jour la liste des tâches éligibles et les tâches déjà planifiées à la période de temps considérée.
7. Retournez à l'étape 4.

Figure 2.3: Pseudocode des heuristiques dites parallèles

2.5.2 Méthodes de recherche dans le voisinage

Ces méthodes peuvent donner de meilleurs résultats que les heuristiques basées sur des règles de priorité puisque à partir d'une solution de départ (donnée par une heuristique constructive par exemple), il est possible d'améliorer cette solution en explorant ses voisinages. Les deux ingrédients essentiels pour définir un voisinage sont le schéma de représentation (le codage de la solution) et les opérateurs de voisinage.

1. **Schéma de de représentation des solutions** : En faisant un survol de la littérature sur les POPRL, on remarque que cinq différents schémas ont été utilisés pour représenter une solution. On trouve :

   a. **Représentation par une liste ou une séquence** : L'ordonnancement (solution) peut être ensuite obtenu en attribuant à chaque tâche la date de début au plus tôt en se basant sur les relations de préséance et la disponibilité des ressources, voir Boctor (1996b).

   b. **Représentation par une liste ordonnée de règles de priorité** : pour ce type de représentation, la séquence est exprimée par un vecteur, chaque composante du

Classe	Acronyme	Signification
Des règles basées sur les caractéristiques des tâches	SPT	Plus petite durée
	MIS	Le plus grand nombre de successeurs immédiats
	MTS	Le plus grand nombre de successeurs
	RPW	La grande somme des durées de la tâche avec tous ses successeurs immédiats
	RPW*	La grande somme des durées de la tâche avec tous ses successeurs
Des règles basées sur le chemin critique	EST	Plus petite date de début au plus tôt
	EFT	Plus petite date de fin au plus tôt
	ESTD	Plus petite date de début au plus tôt (dynamique)
	LST	Plus petite date de début au plus tard
	LFT	Plus petite date de début au plus tard
	MSL	Plus petite marge totale
	MSLD	Plus petite marge totale (dynamique)
Des règles basées sur les ressources	WRUP	Le poids le plus élevé de l'utilisation des ressources et les règles de préséance

Tableau 2.3: Classification des règles de priorité pour le POPRL

vecteur étant une règle de priorité. L'ordonnancement peut être ensuite déduit en appliquant ces règles dans l'ordre préétabli, voir Artigues et al. (2003).

c. **Représentation par un vecteur de clés**: pour ce type de représentation, à chaque tâche est assigné un nombre réel défini sur un intervalle quelconque (par exemple l'intervalle [0 1]). En classant les tâches selon l'ordre croissant ou décroissant, on obtient une représentation en liste et à partir de celle-ci nous pouvons déduire un ordonnancement, voir Gonçalves et al. (2011).

d. **Représentation par vecteur décalé** : la quatrième représentation a été proposée par Sampson and Weiss (1993). Chaque solution est représentée par un vecteur, où chaque composante du vecteur est un entier non négatif, indiquant le nombre de périodes d'exécution qu'une tâche donnée doit être décalée (retardée) par rapport à la date au plus tôt. Un ordonnancement est ensuite obtenu en utilisant les dates ainsi décalées. Évidemment, cette représentation permet de visiter des voisins non réalisables puisque ces dates peuvent mener à des violations des limites de ressources. Par conséquent, une pénalité doit être appliquée à la valeur de la fonction objectif des solutions non réalisables.

e. **Représentation par schéma d'ordonnancement** : Cette représentation a été utilisée par Brucker et al. (1998). Elle se base sur le principe d'exprimer la relation qui existe entre chaque paire de tâches du projet, en fonction de quatre types de

relations : Conjonction, Disjonction, Parallèle ou Flexible. Ces relations définissent respectivement les quatre ensembles disjoints (C, D, P, L). L'ordonnancement est ensuite obtenue par l'heuristique de Baar et al. (1999).

2. **Opérateurs de recherche dans le voisinage** : ce sont des opérations élémentaires appliquées sur une ou deux solutions pour aboutir à une ou plusieurs solutions voisines. Parmi les opérateurs de voisinage, on trouve :

   a. **Opérateurs pouvant être appliqués sur une seule solution** : Partant d'une solution dite courante, une solution voisine peut être obtenue par un :

      i. Opérateur de déplacement où une tâche est déplacé à une nouvelle position dans la liste définissant la solution

      ii. Opérateur de permutations où deux ou plusieurs tâches échangent leur position,

      iii. Opérateur de décalage où une partie de la liste est déplacée (décalée).

   b. **Opérateurs applicables sur deux ou plusieurs solutions** À partir de deux ou plusieurs solutions, d'autres solutions voisines peuvent être obtenues par :

      i. Un croisement des solutions (croisement à une seule position, croisement à plusieurs positions, croisement uniforme),

      ii. Une combinaison (linéaire ou non) de ces solutions.

Plusieurs approches de recherche dans le voisinage ont été proposées dans la littérature. Le Tableau 2.4 présente une classification de quelques-unes de ces méthodes en se limitant aux trois heuristiques de recherche dans le voisinage les plus utilisées (recherche tabou, algorithme génétique et l'algorithme de recuit simulé). Pour chacun de ces travaux, le tableau donne la méthode employée, le code de représentation de la solution, l'heuristique utilisée pour déduire un ordonnancement du code de présentation, et les opérateurs de recherche dans le voisinage utilisé.

Kolisch and Hartmann (2006) ont comparé plusieurs de ces méthodes, notamment, des méthodes de recherche tabou, des algorithmes génétiques, des algorithmes de recuit simulé ainsi que plusieurs méthodes hybrides. Leurs résultats expérimentaux démontrent que les méthodes hybrides sont les plus prometteuses, comme celle de Kochetov and Stolyar (2003), où l'algorithme génétique est hybridé avec une méthode de recherche tabou.

Des méthodes aussi efficaces que celle de Kochetov and Stolyar (2003) ont vu le jour ultérieurement. Debels et al. (2006) ont utilisé la nouvelle technique dite de recherche dispersée (Scatter Search). Chen (2011) et Koulinas et al. (2014) ont proposé la méthode dite Particle swam optimisation. Valls et al. (2008), Mendes et al. (2009), Montoya-Torres et al. (2010) et Zamani (2013) quant à eux ont utilisé des algorithmes génétiques.

	Auteurs	Code de représentation de la solution	Méthode de construction de la solution	Opérateurs de recherche dans le voisinage
Recherche Tabou	Lee and Kim (1996)	Liste aléatoire	parallèle	Permutation
	Baar et al. (1999)	Liste de priorité	sériel	décalage
	Thomas and Salhi (1998)	Schéma d'ordonnancement	Autre	
	Klein and Scholl (1999)	Ordonnancement	-	trois mouvements
	Artigues et al. (2003)	Liste de priorité	sériel	décalage
Algorithme Génétique	Lee and Kim (1996)	Clé aléatoire	parallèle	croisement à 1 points
	Hartmann (1998)	Règle de priorité	sériel	croisement à 2 points
		Clé aléatoire	sériel	croisement à 2 points
	Kohlmorgen et al. (1999)	Clé aléatoire	sériel	croisement à 2 points
	Hartmann (2002)	Liste de priorité	sériel/parallèle	croisement à 2 points
	Hindi et al. (2002)	Liste de priorité	sériel	croisement à 2 points
	Toklu (2002)	Ordonnancement	-	-
	Coelho and Valadares (2002)	Liste de priorité	sériel	croisement à 2 points
	Valls et al. (2008)	Liste de priorité	sériel	croisement à 2 points
	Mendes et al. (2009)	Clé aléatoire	Procédure adaptée	croisement à 2 points
	Montoya-Torres et al. (2010)	Liste de priorité	sériel	croisement à 2 points
	Zamani (2013)	Liste de priorité	sériel/parallèle	croisement à 2 points modifié
Recuit simulé	Sampson and Weiss (1993)	Vecteur de décalage	-	-
	Boctor (1996b)	Liste de priorité	sériel	Permutation
	Lee and Kim (1996)	Clé aléatoire	parallèle	Permutation
	Bouleimen and Lecocq (2003)	Liste de priorité	sériel	Permutation
	Valls et al. (2004)	Clé aléatoire	sériel/parallèle	Permutation

Tableau 2.4: Approches de recherche dans le voisinage proposées dans la littérature pour le POPRL.

Le Tableau 2.5 propose une classification des résultats obtenus par ces méthodes quand elles ont été testées sur les trois ensembles de problème test de Kolisch and Sprecher (1997) : les ensembles appelés J30 (avec 30 tâches), J60 (60 tâches) et J120 (120 tâches). Les résultats sont obtenus en limitant à 5000 le nombre de solutions générées.

Auteurs	Méthode de recherche dans le voisinage	Déviation moyenne par rapport		
		l'optimum	à longueur du chemin critique	
		J30	J60	J120
Kochetov and Stolyar (2003)	Hybride	0.04	11.17	32.06
Debels et al. (2006)	Scatter Search	0.11	11.10	33.10
Valls et al. (2008)	Algorithme Génétique	0.06	11.10	**32.54**
Mendes et al. (2009)	Algorithme Génétique	**0.02**	11.04	33.03
Chen (2011)	Particle swam optimisation	0.14	11.43	33.88
Zamani (2013)	Algorithme Génétique	0.04	**10.94**	32.89
Koulinas et al. (2014)	Particle swam optimisation	0.04	11.13	32.59

Tableau 2.5: Comparaison des résultats de quelques travaux récents

2.5.3 Techniques d'énumération tronquée

Pour obtenir des solutions de bonne qualité pour le POPRL, en des temps de calcul raisonnables, quelques auteurs ont opté pour des méthodes d'énumération telles que la méthode de séparation et évaluation, mais en appliquant une règle qui permet d'arrêter le déroulement de la méthode après un temps prédéterminé ou en limitant le nombre de nœuds visités dans l'arbre d'énumération (voir Pollack-Johnson (1995), Demeulemeester and Herroelen (1997), Sprecher (2000), et Demeulemeester and Herroelen (2006), pour plus de détails).

2.5.4 Approches basées sur l'ajout des arcs disjonctifs

Cette approche a trouvé son application dans les travaux de Alvarez-Valdes and Tamarit (1989) et Bell and Han (1991). L'idée de base de cette approche est l'ajout de nouvelles relations de préséance au problème original, dans l'objectif de rendre le calendrier au plus tôt réalisable avec les ressources disponibles.

2.6 Généralisation du POPRL

Le POPRL tel qu'il a été défini précédemment est une simplification des situations réelles, ce qui a incité les chercheurs ces dernières années à proposer plusieurs généralisations du problème standard. Hartmann and Briskorn (2010) ont proposé une revue bibliographique dans laquelle

ils ont rapporté quelque cent quatre-vingt-dix-huit références traitant des variantes ainsi que des généralisations du problème.

Dans la sous-section suivante, on classifie quelques variantes et généralisation du POPRL trouvées dans la littérature.

2.6.1 Classification de quelques variantes et généralisation du POPRL

Dans le Tableau 2.6, on propose un bref survol de quelques-unes de ces variantes divisées en trois groupes:

1. Variantes avec des objectifs alternatifs,
2. Généralisations avec des contraintes de préséance généralisées,
3. Variantes avec de nouvelles hypothèses.

Dans les sections qui suivent, la littérature relative à trois de ces généralisations va être présentée. La section 2.7, présente la littérature traitant le problème où les tâches peuvent avoir plusieurs modes d'exécution. Cette version plus réaliste du problème a fait l'objet d'un nombre important de publications. La section 2.8 présente la littérature relative au problème avec temps de transfert des ressources entre les sites d'exécution des tâches. On constate qu'il existe très peu de travaux consacrée a ce problème. Afin de remédier à cette lacune, le chapitre 4 et le chapitre 5 de cette thèse sont dédiés à cet aspect très pratique du problème.

La section 2.9 est consacrée au problème d'ordonnancement de projet dans l'objectif de minimiser le coût d'exécution du projet. Nous verons que la plupart des publications ne prennent pas en considération tous les éléments du coût. C'est la question de recherche à laquelle nous nous intéréssons au chapitre 5 de la présente thèse.

2.7 Problème d'ordonnamcement de projet à ressources limitées avec plusieurs modes d'exécution-POPRL/PME

Le problème d'ordonnancement de projet sous contrainte de ressources limitées où les tâches ont plusieurs modes d'exécution sera noté POPRL/PME. Ce problème est une généralisation du problème standard permettant que chaque tâche ait un nombre limité de modes d'exécution

Variantes avec d'autres objectifs	Objectif basé sur le temps	Nudtasomboon and Randhawa (1997); Kolisch (2000); Viana and de Sousa (2000)
	Objectif basé sur la robustesse de la cédule	Icmeli-Tukel and Rom (1997); Al-Fawzan and Haouari (2005) Abbasi et al. (2006);Kobylański and Kuchta (2007); Chtourou and Haouari (2008)
	Objectif pour le réordonnancement	El Sakkout and Wallace (2000); Vanhoucke et al. (2001) Calhoun et al. (2002); Van de Vonder et al. (2007)
	Objectif basé sur les coûts	Maniezzo and Mingozzi (1999); Möhring et al. (2001) Khoshjahan et al. (2013);Gomes et al. (2014)
	Maximiser la valeur actuelle nette	Vanhoucke et al. (2001); Mika et al. (2005); Kim et al. (2005); Padman and Zhu (2006); Vanhoucke (2010) Leyman and Vanhoucke (2015);Leyman and Vanhoucke (2016)
	Objectifs multiples	Nudtasomboon and Randhawa (1997); Hapke et al. (1998) Al-Fawzan and Haouari (2005); Doerner et al. (2008) Xiao et al. (2016)
Contraintes de temps généralisées	Décalage minimal et maximal entre la fin de la tâche et le début d'un successeur	Bartusch et al. (1988); Neumann and Zhan (1995) Herroelen et al. (1998); De Reyck et al. (1998) Kreter et al. (2016)
Généralisations avec des hypothèses différentes	Preemption permise	Franck et al. (2001); Buddhakulsomsiri and Kim (2006) Damay et al. (2007); Ballestín et al. (2008)
	Temps de préparation avant l'exécution de tâches	Van de Vonder et al. (2007); Mika et al. (2008)
	Tâches ayant plusieurs modes d'exécution	Talbot (1982); (Boctor, 1993, 1996b,a); Mori and Tseng (1997) Boctor and d'Avignon (2005); Zhu et al. (2006); Zhu et al. (2006) Vanhoucke et al. (2008); Van Peteghem and Vanhoucke (2010) Coelho and Vanhoucke (2011); Van Peteghem and Vanhoucke (2014) Messelis and De Causmaecker (2014)
	Temps de transfert des ressources	(Krüger and Scholl, 2009; Krüger, 2009) Poppenborg and Knust (2014)

Tableau 2.6: Classification de quelques variantes et généralisations du POPRL.

où chaque mode est caractérisé par les quantités des ressources renouvelables et non renouvelables requises et la durée d'exécution que l'on peut obtenir si ces quantités de ressources sont utilisées pour exécuter la tâche. Dans un POPRL/PME, il s'agit non seulement de trouver les dates de début ou de fin des tâches, mais aussi de choisir le mode d'exécution à utiliser pour chaque tâche avec l'objectif de minimiser la durée du projet. Le POPRL/PME a été d'une grande popularité ces dernières années, une revue de littérature est proposé par Węglarz et al. (2011).

Dans la section suivante, on définit le problème d'une façon un peu plus formelle, la deuxième et la troisième sous-section présenteront un survol des approches de résolution exactes et heuristiques respectivement.

2.7.1 Modélisation

Pour modéliser un POPRL/PME, Talbot (1982) a proposé une généralisation de la formulation à temps discrétisée de Pritsker et al. (1969). Il définit les variables de décision comme suit :

$$x_{imt} = \begin{cases} 1, & \text{si l'exécution de la tâche } i \text{ commence au début de la période } t \\ & \text{en utilisant le mode } m \\ 0, & \text{sinon.} \end{cases}$$

$$\text{Minimiser} \sum_{t \in I_i} t x_{F1t} \tag{2.29}$$

Sous les contraintes :

$$\sum_{m \in M_i} \sum_{t \in I_i} x_{imt} = 1 \qquad \forall i \in J \cup \{1, F\} \tag{2.30}$$

$$\sum_{m' \in M_j} \sum_{t \in I_j} t\, x_{jm't} - \sum_{m \in M_i} \sum_{t \in I_i} (t + d_{im})\, x_{imt} \geq 0 \qquad \forall (i,j) \in P \tag{2.31}$$

$$\sum_{i \in J \cup \{1,F\}} \sum_{m \in M_i} r_{imk} \sum_{\tau = max(t-d_i, EST_i)}^{min(t-1, LST_i)} x_{im\tau} \leq Q_k \qquad t = 1, ..., T \in K \forall k \in K \tag{2.32}$$

$$\sum_{i \in J \cup \{1,F\}} \sum_{m \in M_i} \sum_{t=I_i} n_{imw} x_{im\tau} \leq N_w \qquad \forall w \in W \tag{2.33}$$

$$x_{imt} \in \{0, 1\} \qquad \forall i \in J \cup \{1, F\}, \forall m \in M_i \forall t \in I_i \tag{2.34}$$

L'objectif est la minimisation de la durée du projet, il est donné par l'expression (2.29). Les contraintes (2.30) imposent qu'à chaque tâche soit attribué un seul mode d'exécution et une seule date de début (préemption non permise). Les contraintes de préséances sont exprimées par (2.31), quant aux contraintes de ressources renouvelables et non renouvelables elles sont données respectivement par les contraintes (2.32) et (2.33). Les contraintes (2.34) assurent que toutes les variables de décision du modèle prennent des valeurs binaires.

Une nouvelle formulation du POPRL/PME a été proposée par Maniezzo and Mingozzi (1999). Les auteurs ont proposé une généralisation de la formulation basée sur les ensembles admissibles de Mingozzi et al. (1998).

2.7.2 Méthodes de résolution exactes

Le PORPL/PME est un problème de complexité NP-dur. Les approches de résolution exactes qui ont été développées pour la résolution de ce type de problème sont des méthodes d'énumération implicite :

- Algorithmes basés sur le branchement d'une seule tâche par nœud : ils ont été proposés par : Patterson et al. (1989), Sprecher et al. (1997) et par Hartmann and Drexl (1998).
- Algorithme basé sur les alternatives de retard : il a été proposé par Sprecher et al. (1997).
- Algorithme basé sur les alternatives d'extension : il a été proposé par Sprecher et al. (1997), inspiré de l'algorithme de Demeulemeester and Herroelen (1992).

Hartmann and Drexl (1998) ont mené une étude comparative entre ces trois types d'algorithmes pour la résolution des instances de 10,12, 14 et 16 tâches de Kolisch and Sprecher (1997). Ils ont conclu que :

- Algorithme basé sur l'arbre de préséance proposé par Sprecher et al. (1997) est le plus facile à implémenter et le mieux adapté pour les grandes instances.
- Résolution exacte des problèmes de 20 tâches et plus par des méthodes exactes dans des temps acceptables reste toujours une tâche très ardue.

Une nouvelle méthode a été proposée par Zhu et al. (2006). C'est une méthode qui combine la génération d'inégalités valides et la méthode de séparation et coupe (ou Branch and Cut). Cette méthode a été testée sur les problèmes de la librairie PSLIB de Kolisch and Sprecher (1997). L'algorithme, contrairement au précédent, a réussi de trouver la solution optimale de tous les problèmes test de 20 tâches. Pour les problèmes à 30 tâches, il a trouvé l'optimum pour 506 problèmes et a trouvé la meilleure solution connue pour 529 problèmes parmi les 552 de ces ensembles de problèmes tests.

2.7.3 Méthodes heuristiques de résolution

Les heuristiques proposées dans la littérature pour résoudre le POPRL/PME se divisent en deux groupes : celles conçues pour la version avec des ressources renouvelables uniquement et celles qui traitent les problèmes avec ressources renouvelables et non renouvelables. Parmi les heuristiques pour la version avec des ressources renouvelables, on peut citer (Boctor, 1993, 1996a,b; Gagnon et al., 2005). Le Tableau 2.7 présente une classification de ces travaux selon l'approche de résolution adoptée.

Approche de résolution	Auteurs	Instances de test	Nombre tâche	Nombre de modes d'exécution	Nombre de types de ressources
Constructive	Boctor (1993)	Boctor	50/00	1 à 4	1, 2 ou 4
	Boctor (1996a)	Boctor	50/100	1 à 4	1, 2 ou 4
	Lova et al. (2006)	Boctor	50/100	1 à 4	1, 2 ou 4
Algorithme génétique	Mori and Tseng (1997)	Autuers	20 : 70	2 à 4	4
Recherche tabou	Gagnon et al. (2005)	Boctor	50/100	1 à 4	1, 2 ou 4
Recuit simulé	Boctor (1996b)	Boctor	50/100	1 à 4	1, 2 ou 4

Tableau 2.7: Heuristiques de résolution du POPRL/PME avec ressources renouvelables.

Les heuristiques développées pour résoudre le POPRL/PME avec des ressources renouvelables et non renouvelables se devise en trois groupes (voir Tableau 2.8):

- Des heuristiques constructives basées sur les règles de priorité
- Des méthodes de recherche dans le voisinage
- Des méthodes d'évaluation et de séparation tronquées.

Van Peteghem and Vanhoucke (2014) ont effectué une étude comparative entre les meilleures heuristiques développées dans la littérature pour la résolution des POPRL/PME en utilisant les instances de PSPLIB de (Kolisch and Sprecher, 1997) avec 10, 12, 14, 16, 18, 20 et 30 tâches; les problèmes-test de Boctor (1993) à 50 et 100 tâches et d'autre problèmes-test proposés par les auteurs, només MMLIB50, MMLIB100 et MMLIB+ incluant 50, 100 et entre 50 et 100 tâches respectivement. Les resultats obenus sont résumés dans les deux tableaux Tableau 2.9 et Tableau 2.10. Où les déviations pour les instances de Kolisch et al. (1996) de 10 à 20 tâches sont mesurées par rapport à l'optimum. Pour les instances de grande taille, c'est à dire les instances J30 de Kolisch et al. (1996) et celles de 50 et 100 tâches de Boctor (1996b) et Van Peteghem and Vanhoucke (2014) les déviations sont mesurés par rapport à la longueur du chemin critique.

Approche	Auteurs	Instances	Nombre de de tâche	Nombre de mode tâche	Ressources renouvelables/non renouvelables
Recherche locale	Kolisch and Drexl (1997)	PSPLIB	30	3	2/2
Algorithme génétique	Hartmann (2001)	PSPLIB	30	3	2/2
	Alcaraz et al. (2003)	PSPLIB Boctor	30 100	3 1 à 4	2/2 1 à 4/0
	Lova et al. (2009)	PSPLIB Boctor	30 100	3 1 à 4	2/2 1 à 4/0
	Tseng and Chen (2009)	PSPLIB	30	3	2/2
	Van Peteghem and Vanhoucke (2010)	PSPLIB Boctor	30 100	3 1 à 4	2/2 1 à 4/0
Recherche tabou	Nonobe and Ibaraki (2002)	PSPLIB	30	3	2/2
Recuit simulé	Józefowska et al. (2001)	PSPLIB	30	3	2/2
	Bouleimen and Lecocq (2003)	PSPLIB Boctor	30 100	3 1 à 4	2/2 1 à 4/0
Colonies de fourmis	Chyu et al. (2005)	PSPLIB	30	3	2/2
	Chiang et al. (2008)	PSPLIB	30	3	2/2
Algorithme évolutionnaire	Elloumi et al. (2006)	PSPLIB	30	3	2/2
	Damak et al. (2009)	PSPLIB	30	3	2/2
PSO	Zhang et al. (2006)	PSPLIB	30	3	2/2
	Jarboui et al. (2008)	PSPLIB	30	3	2/2
PLA	Jedrzejowicz and Ratajczak (2006)	PSPLIB	30	3	2/2
Décomposition de Benders	Maniezzo and Mingozzi (1999)	PSPLIB	30	3	2/2
	Boschetti and Maniezzo (2009)	PSPLIB	30	3	2/2
recherche dispersée	Van Peteghem and Vanhoucke (2011)	PSPLIB	30	3	2/2

Tableau 2.8: Heuristiques de résolution du POPRL/PME avec ressources non renouvelables.

Récemment, Messelis and De Causmaecker (2014) ont exploré une nouvelle alternative intéressante, c'est à dire la possibilité de construire un outil de sélection automatique d'algorithme pour le POPRL/PME basé sur les caractéristiques des problèmes tests. Leurs idée repose sur le concept de combiner différents algorithmes dans un super-algorithme afin d'asuurer une meilleure performance. Cette approche a prouvé son efficacité pour les problèmes-tests MMLIB50, MMLIB100 et MMLIB+ de Van Peteghem and Vanhoucke (2014)

Méthode	Instances						
	J10	J12	J14	J16	J18	J20	J30
Van Peteghem and Vanhoucke (2011)	0.00	0.02	0.08	0.15	0.23	0.32	13.66
Van Peteghem and Vanhoucke (2010)	0.01	0.09	0.22	0.32	0.42	0.57	13.75
Lova et al. (2009)	0.06	0.17	0.32	0.44	0.63	0.87	14.58
Jarboui et al. (2008)	0.03	0.09	0.36	0.44	0.89	1.1	18.14
Ranjbar and Kianfar (2009)	0.18	0.65	0.89	0.95	1.21	1.64	16.21
Alcaraz et al. (2003)	0.24	0.73	1.00	1.12	1.43	1.91	21.83
Józefowska et al. (2001)	1.16	1.73	2.6	4.07	5.52	6.74	18.64

Tableau 2.9: La déviation moyenne par rapport à l'optimum/à la longueur du chemin critique pour les instances de Kolisch et al. (1996).

	Instances				
	Boctor 50	MMLIB 50	Boctor 100	MMLIB 100	MMLIB+
Van Peteghem and Vanhoucke (2011)	23.79	25.45	25.11	26.51	101.45
Van Peteghem and Vanhoucke (2010)	23.41	27.12	24.67	29.55	97.59
Lova et al. (2009)	23.65	28.59	24,63	31.10	114.07
Boctor (1996b)	25.13	54.48	26.82	66.67	54.48
Olaguibel and Goerlich (1993)	26.52	-	29,16	-	-

Tableau 2.10: La déviation moyenne par rapport à la longueur du chemin critique pour les instances de Boctor (1996b) et Van Peteghem and Vanhoucke (2014)

2.8 Problème d'ordonnancement de projet avec temps de transfert des ressources

Le POPRL classique, tel que défini précédemment, suppose qu'aucun temps de transfert n'est nécessaires pour transférer des ressources entre les différents lieux ou sites d'exécution des différentes tâches. En pratique, il arrive que le déplacement des ressources entre ces sites peut prendre un temps non-négligeable. Par exemple, le matériel de construction lourd pourrait être demandé à différents sites de construction et de les déplacer d'un site à l'autre peut nécessiter plusieurs heures.

Dans un tel cas, la modélisation et la résolution du problème d'ordonnancement de projet sans prendre en considération les temps de transfert pourrait produire des ordonnancements irréalisables. Récemment Krüger and Scholl (2009) ont développé des méthodes pour résoudre le problème d'ordonnancement de projet avec ressources limitées avec des temps de transfert (POPRLTT) dans un contexte d'un projet unique et pour un ensemble de projets. Les auteurs ont formulés les deux problèmes en faisant appel à la programmation linéaire. Des heuristiques constructives ont ensuite été proposées. Ensuite, Krüger (2009) a développé un algorithme génétique pour ce problème. Récemment, un algorithme de recherche tabu basé sur les flux a été développé par Poppenborg and Knust (2014). Kadri and Boctor (2014) ont étudié le problème quand les tâches peuvent avoir plusieurs modes d'exécution POPRL/PMETT.

2.9 Problème de minimisation des coûts du projet

Dans la littérature certains chercheurs ont étudié le problème d'ordonnancemnt de projet dans l'objectif de minimiser le coût du projet. Ce problème est appelé problème de coût de disponibilité des ressources ou en anglais «Resource availability cost problem (RACP)». Dans ce problème, l'objectif est de déterminer les quantités de ressources à allouer au projet afin de minimiser une fonction de coût des ressources qui ne dépend pas de la durée du projet. Möhring (1984) a été le premier à introduire le RACP. Il a developé une procédure de résolution exacte sous l'hypothèse que les tâches ont un seul mode d'exécution. Drexl and Kimms (2001) ont développé deux bornes inférieures pour le problème en utilisant des techniques de relaxation Lagrangienne et de génération de colonne. Le problème qu'ils considéraient et appelé le problème d'investissement, ou en anglais «Resource Investement Problem (RIP)», est légèrement différent de la RACP car ils n'imposent pas une date due au projet. Des approches heuristiques sont également proposées pour résoudre le RACP. Yamashita et al. (2006) a proposé une heursitique de recherche dispersée (Scatter search). Quant à Shadrokh and Kianfar (2007) ont proposé un algorithme génétique.

On constate que la plupart des approches de résolution développées sont conçues pour résoudre

le problème dans un contexte de mode unique d'exécution des tâches. La seule heuristique conçue pour résoudre la version avec plusieurs modes d'exécution est celle de Hsu and Kim (2005). Les auteurs proposent une heursitique basée sur des règles de priorité. Cependant, pour résoudre ce problème, les chercheurs font une hypothèse très discutable. Ils supposent que le coût d'utilisation d'une unité de ressource, disponible à partir du début à la fin d'exécution du projet reste constant quelle que soit la durée du projet. Cette hypothèse simplifie considérablement la modélisation du problème et réduit la complexité des méthodes de résolution. Mais cette hypothèse est loin des situations pratiques où le coût de disponibilité des ressources est plus souvent proportionnel à la durée du projet. Aussi, nous constatons que dans les deux versions du RACP ou RIP, les frais généraux ne sont pas pris en compte. Encore une fois, dans la pratique, les frais généraux représente une partie importante du coût d'exécution du projet et ne doivent pas être négligés.

2.10 Conclusion

Dans ce chapitre, on a revu le problème d'ordonnancement de projet à ressources limitées POPRL. Ce problème est devenu ces dernières années un problème de référence en gestion de projet (Hartmann (2012)). Il s'agit de déterminer les dates d'exécution des tâches dans l'objectif de minimiser la durée totale du projet, tout en respectant les contraintes de préséance entre les tâches du projet et les limites sur les disponibilités des ressources.

Le POPRL tel qu'il a été défini précédemment est une simplification des situations réelles, ce qui a incité les chercheurs ces dernières années à proposer plusieurs généralisations du problème standard, tel que le problème d'ordonnancement de projet à ressources limitées où les tâches ont plusieurs modes d'exécution (POPRL/PME). Dans ce dernier chaque tâche a un nombre limité de modes d'exécution où chaque mode est caractérisé par les quantités des ressources renouvelables et non renouvelables requises et la durée d'exécution que l'on peut obtenir si ces quantités de ressources sont utilisées pour exécuter la tâche. Il s'agit non seulement de trouver les dates de début ou de fin des tâches, mais aussi de choisir le mode d'exécution à utiliser pour chaque tâche avec l'objectif de minimiser la durée du projet. Le POPRL/PME a été d'une grande popularité ces dernières années, une revue de littérature a récemment été proposé par Węglarz et al. (2011).

Ces problèmes ont fait l'objet de plusieurs travaux de recherche; des méthodes aussi bien exactes pour les petites instances, que heuristiques ont été proposées.

Les algorithmes génétiques se distinguent parmi les techniques qui performent le mieux pour la résolution du POPRL et du POPRL/PME.

On constate que malgré les efforts déployés pour définir des POPRL plus généraux, des con-

traintes continuent à être ignorées. Cela force les gestionnaires à se baser uniquement sur leur expérience pour réaliser ou ajuster manuellement les ordonnancements produits par des heuristiques conçues pour résoudre des versions simplifiées du problème. L'objectif de cette thèse est de combler partiellement ces lacunes en analysant des versions du problèmes qui prennent en considération d'autres contraintes pratiques ou qui tente d'optimiser des objectifs plus réalistes.

Ainsi, la thèse propose dans le prochain chapitre une nouvelle approche pour résoudre le problème d'ordonnancement des tâches d'un projet en considération le temps nécessaire pour transférer les ressources entre les sites d'exécution des différentes tâches qui requièrent ces ressources. Le quatrième chapitre propose une nouvelle généralisation du problème précédent pour traiter le cas ou plusieurs modes sont disponibles pour executer chaque tâche, et quant des ressources renouvelables et non renouvelables sont nécéssaires pour realiser le projet.

Le cinquième chapitre présente une nouvelle version du problème d'ordonnancement de projet qui n'a pas encore été résolue par le passé. Il s'agit du problème qui vise à la fois à déterminer les ressources à allouer au projet et d'établir un calendrier d'exécution qui minimise le coût de ces ressources et les frais généraux du projet.

2.11 Références

Abbasi, B., Shadrokh, S., Arkat, J., 2006. Bi-objective resource-constrained project scheduling with robustness and makespan criteria. Applied mathematics and computation 180 (1), 146–152.

Al-Fawzan, M. A., Haouari, M., 2005. A bi-objective model for robust resource-constrained project scheduling. International Journal of production economics 96 (2), 175–187.

Alcaraz, J., M, C., R, R., 2003. Solving the multi-mode resource-constrained project scheduling problem with genetic algorithms. Journal of the Operational Research Society 54 (6), 614–626.

Alvarez-Valdes, R., Tamarit, J. M., 1989. Heuristic algorithms for resource-constrained project scheduling: A review and an empirical analysis. In: Advances in project scheduling. Vol. 9. Elsevier Amsterdam, pp. 113–134.

Artigues, C., Michelon, P., Reusser, S., 2003. Insertion techniques for static and dynamic resource-constrained project scheduling. European Journal of Operational Research 149 (2), 249–267.

Baar, T., Brucker, P., S, K., 1999. Tabu search algorithm and lower bounds for the resource-constrained project scheduling problem. In: Voss et al. (eds) Meta heuristics : Advances and trends in local search paradigms for optimization. Springer, pp. 1–19.

Balas, E., 1985. On the facial structure of scheduling polyhedra. Mathematical programming studies, 179–218.

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

Bartusch, M., Möhring, R. H., Radermacher, F. J., 1988. Scheduling project networks with resource constraints and time windows. Annals of operations Research 16 (1), 199–240.

Bell, C. E., Han, J., 1991. A new heuristic solution method in resource-constrained project scheduling. Naval Research Logistics (NRL) 38 (3), 315–331.

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

Boctor, F. F., 1990. Some efficient multi-heuristic procedures for resource-constrained project scheduling. European Journal of Operational Research 49 (1), 3–13.

Boctor, F. F., 1993. Heuristics for scheduling projects with resource restrictions and several resource-duration modes. International Journal of Production Research 31 (11), 2547–2558.

Boctor, F. F., 1996a. A new and efficient heuristic for scheduling projects with resource restrictions and multiple execution modes. European Journal of Operational Research 90 (2), 349–361.

Boctor, F. F., 1996b. Resource-constrained project scheduling by simulated annealing. International Journal of Production Research 34 (8), 2335–2351.

Boctor, F. F., d'Avignon, G., 2005. A tabu search algorithm for the multiple mode resource-constrained project scheduling problem. Gestion des Opérations et Production, 28–89.

Boschetti, M., Maniezzo, V., 2009. Benders decomposition, lagrangean relaxation and metaheuristic design. Journal of Heuristics 15 (3), 283–312.

Bouleimen, K., Lecocq, H., 2003. A new efficient simulated annealing algorithm for the resource-constrained project scheduling problem and its multiple mode version. European Journal of Operational Research 149 (2), 268–281.

Brucker, P., Knust, S., Schoo, A., Thiele, O., 1998. A branch and bound algorithm for the resource-constrained project scheduling problem. European Journal of Operational Research 107 (2), 272–288.

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

Calhoun, K. M., Deckro, R. F., Moore, J. T., Chrissis, J. W., Van Hove, J. C., 2002. Planning and re-planning in project and production scheduling. Omega 30 (3), 155–170.

Carlier, J., Latapie, B., 1991. Une méthode arborescente pour résoudre les problèmes cumulatifs. Revue française d'automatique, d'informatique et de recherche opérationnelle. Recherche opérationnelle 25 (3), 311–340.

Chen, R.-M., 2011. Particle swarm optimization with justification and designed mechanisms for resource-constrained project scheduling problem. Expert Systems with Applications 38 (6), 7102 – 7111.

Chiang, C.-W., Huang, Y.-Q., Wang, W.-Y., 2008. Ant colony optimization with parameter adaptation for multi-mode resource-constrained project scheduling. Journal of Intelligent & Fuzzy Systems 19 (4, 5), 345–358.

Christofides, N., Alvarez-Valdés, R., Tamarit, J. M., 1987. Project scheduling with resource constraints: A branch and bound approach. European Journal of Operational Research 29 (3), 262–273.

Chtourou, H., Haouari, M., 2008. A two-stage-priority-rule-based algorithm for robust resource-constrained project scheduling. Computers & industrial engineering 55 (1), 183–194.

Chyu, C.-C., Chen, A. H., Lin, X.-H., 2005. A hybrid ant colony approach to multi-mode resource-constrained project scheduling problems with non-renewable types. In: Proceedings of First International Conference on Operations and Supply Chain Management, Bali.

Coelho, J., Valadares, L. T., 2002. Comparative analysis on approximation algorithms for the resource constrained project scheduling problem. In: Eighth International Workshop on Project Management and Schduling. EURO Working Group on Project Management and Scheduling.

Coelho, J., Vanhoucke, M., 2011. Multi-mode resource-constrained project scheduling using rcpsp and sat solvers. European Journal of Operational Research 213 (1), 73–82.

Damak, N., Jarboui, B., Siarry, P., Loukil, T., 2009. Differential evolution for solving multi-mode resource-constrained project scheduling problems. Computers & Operations Research 36 (9), 2653–2659.

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

De Reyck, B., et al., 1998. A branch-and-bound procedure for the resource-constrained project scheduling problem with generalized precedence relations. European Journal of Operational Research 111 (1), 152–174.

Debels, D., De Reyck, B., Leus, R., Vanhoucke, M., 2006. A hybrid scatter search/electromagnetism meta-heuristic for project scheduling. European Journal of Operational Research 169 (2), 638–653.

Demeulemeester, E., Herroelen, W., 1992. A branch-and-bound procedure for the multiple resource-constrained project scheduling problem. Management Science 38 (12), 1803–1818.

Demeulemeester, E. L., Herroelen, W. S., 1997. New benchmark results for the resource-constrained project scheduling problem. Management Science 43 (11), 1485–1492.

Demeulemeester, E. L., Herroelen, W. S., 2006. Project scheduling: a research handbook. Vol. 49. Springer Science & Business Media.

Doerner, K. F., Gutjahr, W. J., Hartl, R. F., Strauss, C., Stummer, C., 2008. Nature-inspired metaheuristics for multiobjective activity crashing. Omega 36 (6), 1019–1037.

Drexl, A., Kimms, A., 2001. Optimization guided lower and upper bounds for the resource investment problem. Journal of the Operational Research Society, 340–351.

El Sakkout, H., Wallace, M., 2000. Probe backtrack search for minimal perturbation in dynamic scheduling. Constraints 5 (4), 359–388.

Elloumi, S., Fortemps, P., Teghem, J., Loukil, T., 2006. A new bi-objective algorithm using clustering heuristics to solve the multi-mode resource-constrained project scheduling problem. In: Proceedings of the 25th workshop of the UK planning and scheduling special interest group (PlanSIG 2006), Nottingham. pp. 113–120.

Franck, B., Neumann, K., Schwindt, C., 2001. Project scheduling with calendars. OR-Spektrum 23 (3), 325–334.

Gagnon, M., Boctor, F. F., Davignon, G., 2005. A tabu search algorithm for multiple mode resource-constrained project scheduling problem.

Gomes, H. C., de Assis das Neves, F., Souza, M. J. F., 2014. Multi-objective metaheuristic algorithms for the resource-constrained project scheduling problem with precedence relations. Computers & Operations Research 44, 92 – 104.

Gonçalves, J. F., Resende, M. G. C., Mendes, J. J. M., 2011. A biased random-key genetic algorithm with forward-backward improvement for the resource constrained project scheduling problem. Journal of Heuristics 17 (5), 467–486.

Hapke, M., Jaszkiewicz, A., Słowiński, R., 1998. Interactive analysis of multiple-criteria project scheduling problems. European Journal of Operational Research 107 (2), 315–324.

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

Hartmann, S., 2002. A self-adapting genetic algorithm for project scheduling under resource constraints. Naval Research Logistics (NRL) 49 (5), 433–448.

Hartmann, S., 2012. Project scheduling under limited resources: models, methods, and applications. Vol. 478. Springer Science & Business Media.

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

Hartmann, S., Drexl, A., 1998. Project scheduling with multiple modes: a comparison of exact algorithms. Networks 32 (4), 283–297.

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

Hindi, K. S., Yang, H., Fleszar, K., 2002. An evolutionary algorithm for resource-constrained project scheduling. Evolutionary Computation, IEEE Transactions on 6 (5), 512–518.

Hsu, C., Kim, D. S., 2005. A new heuristic for the multi-mode resource investment problem. Journal of the Operational Research Society 56 (4), pp. 406–413.

Icmeli-Tukel, O., Rom, W. O., 1997. Ensuring quality in resource constrained project scheduling. European Journal of Operational Research 103 (3), 483–496.

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

Jedrzejowicz, P., Ratajczak, E., 2006. Population learning algorithm for the resource-constrained project scheduling. Springer.

Józefowska, J., Mika, M., Różycki, R., Waligóra, G., Węglarz, J., 2001. Simulated annealing for multi-mode resource-constrained project scheduling. Annals of Operations Research 102 (1-4), 137–155.

Kadri, R. L., Boctor, F. F., 2014. Multi-mode resource constrained project scheduling with sequence dependent transfer times. In: Proceedings of the 14th International Conference on Project Management and Scheduling. pp. 106 – 109.

Khoshjahan, Y., Najafi, A. A., Afshar-Nadjafi, B., 2013. Resource constrained project scheduling problem with discounted earliness–tardiness penalties: Mathematical modeling and solving procedure. Computers & Industrial Engineering 66 (2), 293 – 300.

Kim, K., Yun, Y., Yoon, J., Gen, M., Yamazaki, G., 2005. Hybrid genetic algorithm with adaptive abilities for resource-constrained multiple project scheduling. Computers in industry 56 (2), 143–160.

Klein, R., 1999. Scheduling of resource-constrained projects. Vol. 10. Springer Science & Business Media.

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

Kobylański, P., Kuchta, D., 2007. A note on the paper by ma al-fawzan and m. haouari about a bi-objective problem for robust resource-constrained project scheduling. International Journal of Production Economics 107 (2), 496–501.

Kochetov, Y., Stolyar, A., 2003. Evolutionary local search with variable neighborhood for the resource constrained project scheduling problem. In: Proceedings of the 3rd international workshop of computer science and information technologies. Vol. 132.

Kohlmorgen, U., Schmeck, H., Haase, K., 1999. Experiences with fine-grainedparallel genetic algorithms. Annals of Operations Research 90, 203–219.

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

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

Kolisch, R., Drexl, A., 1997. Local search for nonpreemptive multi-mode resource-constrained project scheduling. IIE transactions 29 (11), 987–999.

Kolisch, R., 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., 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.

Koulinas, G., Kotsikas, L., Anagnostopoulos, K., 2014. A particle swarm optimization based hyper-heuristic algorithm for the classic resource constrained project scheduling problem. Information Sciences 277, 680 – 693.

Kreter, S., Rieck, J., Zimmermann, J., 2016. Models and solution procedures for the resource-constrained project scheduling problem with general temporal constraints and calendars. European Journal of Operational Research 251 (2), 387 – 403.

Krüger, D., 2009. Multi-project scheduling with transfers. Ph.D. thesis, University of Jena, Germany.

Krüger, D., 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.

Lee, J.-K., Kim, Y.-D., 1996. Search heuristics for resource constrained project scheduling. Journal of the Operational Research Society, 678–689.

Leyman, P., Vanhoucke, M., 2015. A new scheduling technique for the resource–constrained project scheduling problem with discounted cash flows. International Journal of Production Research 53 (9), 2771–2786.

Leyman, P., Vanhoucke, M., 2016. Payment models and net present value optimization for resource-constrained project scheduling. Computers & Industrial Engineering 91, 139–153.

Lova, A., Tormos, P., Barber, F., 2006. Multi-mode resource constrained project scheduling: scheduling schemes, priority rules and mode selection rules. Inteligencia Artificial 30 (10), 69–86.

Lova, A., Tormos, P., Cervantes, M., Barber, F., 2009. An efficient hybrid genetic algorithm for scheduling projects with resource constraints and multiple execution modes. International Journal of Production Economics 117 (2), 302–316.

Maniezzo, V., Mingozzi, A., 1999. The project scheduling problem with irregular starting time costs. Operations Research Letters 25 (4), 175–182.

Mendes, J. J. d. M., Gonçalves, J. F., Resende, M. G., 2009. A random key based genetic algorithm for the resource constrained project scheduling problem. Computers & Operations Research 36 (1), 92–109.

Messelis, T., De Causmaecker, P., 2014. An automatic algorithm selection approach for the multi-mode resource-constrained project scheduling problem. European Journal of Operational Research 233 (3), 511–528.

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

Mika, M., Waligora, G., Węglarz, 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.

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

Möhring, R. H., 1984. Minimizing costs of resource requirements in project networks subject to a fixed completion time. Operations Research 32 (1), 89–120.

Möhring, R. H., Schulz, A. S., Stork, F., Uetz, M., 2001. On project scheduling with irregular starting time costs. Operations Research Letters 28 (4), 149–154.

Montoya-Torres, J. R., Gutierrez-Franco, E., Pirachicán-Mayorga, C., 2010. Project scheduling with limited resources using a genetic algorithm. International Journal of Project Management 28 (6), 619–628.

Mori, M., Tseng, C. C., 1997. A genetic algorithm for multi-mode resource constrained project scheduling problem. European Journal of Operational Research 100 (1), 134–141.

Neumann, K., Zhan, J., 1995. Heuristics for the minimum project-duration problem with minimal and maximal time lags under fixed resource constraints. Journal of Intelligent Manufacturing 6 (2), 145–154.

Nonobe, K., Ibaraki, T., 2002. Formulation and tabu search algorithm for the resource constrained project scheduling problem. In: Essays and surveys in metaheuristics. Springer, pp. 557–588.

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

Olaguíbel, R. A.-V., Goerlich, J. T., 1993. The project scheduling polyhedron: Dimension, facets and lifting theorems. European Journal of Operational Research 67 (2), 204 – 220.

Olaguibel, R. A.-V., Goerlich, J. T., 1993. The project scheduling polyhedron: dimension, facets and lifting theorems. European Journal of Operational Research 67 (2), 204–220.

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

Patterson, J., Slowinski, R., Talbot, F., Weglarz, J., 1989. An algorithm for a general class of precedence and resource constrained scheduling problems. Advances in project scheduling 187, 3–28.

Patterson, J. H., 1984. A comparison of exact approaches for solving the multiple constrained resource, project scheduling problem. Management Science 30 (7), 854–867.

Pollack-Johnson, B., 1995. Hybrid structures and improving forecasting and scheduling in project management. Journal of Operations Management 12 (2), 101–117.

Poppenborg, J., Knust, S., 2014. A flow-based tabu search algorithm for the rcpsp with transfer times. In: Proceedings of the 14th International Conference on Project Management and Scheduling, March 30th– April 2nd 2014. pp. 181 – 184.

Pritsker, A. A. B., Waiters, L. J., Wolfe, P. M., 1969. Multiproject scheduling with limited resources: A zero-one programming approach. Management Science 16 (1), 93–108.

Ranjbar, M., Kianfar, F., 2009. A hybrid scatter search for the rcpsp. Scientia Iranica. Transaction E, Industrial Engineering 16 (1), 11.

Sampson, S. E., Weiss, E. N., 1993. Local search techniques for the generalized resource constrained project scheduling problem. Naval Research Logistics (NRL) 40 (5), 665–675.

Shadrokh, S., Kianfar, F., 2007. A genetic algorithm for resource investment project scheduling problem, tardiness permitted with penalty. European Journal of Operational Research 181 (1), 86 – 101.

Sprecher, A., 2000. Scheduling resource-constrained projects competitively at modest memory requirements. Management Science 46 (5), 710–723.

Sprecher, A., Hartmann, S., Drexl, A., 1997. An exact algorithm for project scheduling with multiple modes. Operations-Research-Spektrum 19 (3), 195–203.

Stinson, J. P., Davis, E. W., Khumawala, B. M., 1978. Multiple resource–constrained scheduling using branch and bound. AIIE Transactions 10 (3), 252–259.

Talbot, F. B., 1982. Resource-constrained project scheduling with time-resource tradeoffs: The nonpreemptive case. Management Science 28 (10), 1197–1210.

Thomas, P. R., Salhi, S., 1998. A tabu search approach for the resource constrained project scheduling problem. Journal of Heuristics 4 (2), 123–139.

Toklu, Y. C., 2002. Application of genetic algorithms to construction scheduling with or without resource constraints. Canadian Journal of Civil Engineering 29 (3), 421–429.

Tseng, L.-Y., Chen, S.-C., 2009. Two-phase genetic local search algorithm for the multimode resource-constrained project scheduling problem. Evolutionary Computation, IEEE Transactions on 13 (4), 848–857.

Valls, V., Ballestín, F., Quintanilla, S., 2004. A population-based approach to the resource-constrained project scheduling problem. Annals of Operations Research 131 (1-4), 305–324.

Valls, V., Ballestin, F., Quintanilla, S., 2008. A hybrid genetic algorithm for the resource-constrained project scheduling problem. European Journal of Operational Research 185 (2), 495–508.

Van de Vonder, S., Demeulemeester, E., Herroelen, W., 2007. A classification of predictive-reactive project scheduling procedures. Journal of Scheduling 10 (3), 195–207.

Van Peteghem, V., Vanhoucke, M., 2010. A genetic algorithm for the preemptive and non-preemptive multi-mode resource-constrained project scheduling problem. European Journal of Operational Research 201 (2), 409–418.

Van Peteghem, V., Vanhoucke, M., 2011. Using resource scarceness characteristics to solve the multi-mode resource-constrained project scheduling problem. Journal of Heuristics 17 (6), 705–728.

Van Peteghem, V., Vanhoucke, M., 2014. An experimental investigation of metaheuristics for the multi-mode resource-constrained project scheduling problem on new dataset instances. European Journal of Operational Research 235 (1), 62–72.

Vanhoucke, M., 2010. A scatter search heuristic for maximising the net present value of a resource-constrained project with fixed activity cash flows. International Journal of Production Research 48 (7), 1983–2001.

Vanhoucke, M., Coelho, J., Debels, D., Maenhout, B., Tavares, L. V., 2008. An evaluation of the adequacy of project network generators with systematically sampled networks. European Journal of Operational Research 187 (2), 511–524.

Vanhoucke, M., Demeulemeester, E., Herroelen, W., 2001. On maximizing the net present value of a project under renewable resource constraints. Management Science 47 (8), 1113–1121.

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

Węglarz, J., Józefowska, J., Mika, M., Waligóra, G., 2011. Project scheduling with finite or infinite number of activity processing modes–a survey. European Journal of Operational Research 208 (3), 177–205.

Xiao, J., Wu, Z., Hong, X.-X., Tang, J.-C., Tang, Y., 2016. Integration of electromagnetism with multi-objective evolutionary algorithms for rcpsp. European Journal of Operational Research 251 (1), 22–35.

Yamashita, D. S., Armentano, V. A., Laguna, M., 2006. Scatter search for project scheduling with resource availability cost. European Journal of Operational Research 169 (2), 623 – 637.

Zamani, R., 2013. A competitive magnet-based genetic algorithm for solving the resource-constrained project scheduling problem. European Journal of Operational Research 229 (2), 552 – 559.

Zhang, H., Tam, C., Li, H., 2006. Multimode project scheduling based on particle swarm optimization. Computer-Aided Civil and Infrastructure Engineering 21 (2), 93–103.

Zhu, G., Bard, J. F., Yu, G., 2006. A branch-and-cut procedure for the multimode resource-constrained project-scheduling problem. INFORMS Journal on Computing 18 (3), 377–390.

Chapitre 3

Problème d'ordonnancement de projet à ressources limitées et des temps de transfert des ressources (POPRLTT)

Dans ce chapitre on traite le problème d'ordonnancement de projet à ressources limitées et des temps de transfert des ressources (POPRLTT). On suppose que la préemption n'est pas permise et les contraintes de préséances sont dit de type début-fin sans décalage. On suppose aussi que les durées des tâches et les temps de transfert des ressources sont connues. Le but dans un POPRLTT est la détermination des dates d'exécution des tâches en tenant compte des contraintes de préséance et de disponibilité des ressources et les temps de transfert des ressources. L'objectif est la minimisation de la durée totale du projet.
Nous proposons un nouvel algorithme génétique basé sur un opérateur de croisement à deux positions. L'étude expérimentale menée sur un grand nombre de problèmes-tests prouve la performance de l'algorithme proposé par rapport aux méthodes deja existantes dans la littérature.
Notons qu'une version de cet article intitulé «*An efficient genetic algorithm to solve the resource-constrained project scheduling problem with sequence dependent transfer times : The single mode case*» a été coécrit avec Fayez Fouad Boctor et soumis pour publication à *European Journal of Operational Research* le 14 avril 2016. Une version préliminaire a été arbitrée par le comité d'évaluation et publiée dans les actes de la conférence internationale PMS 2016 (*Project Management and Scheduling*), 19 au 22 avril 2016, Valence, Espagne.

Article 1 : An efficient genetic algorithm to solve the resource-constrained project scheduling problem with sequence dependent transfer times : The single mode case

Abstract

In this article, we address the Resource-Constrained Project Scheduling Problem with sequence dependent Transfer Times (RCPSPTT). We assume that pre-emption is not allowed and precedence relations are zero-lag finish-to-start relations. Also, we assume that activities durations and resource transfer times are known and deterministic. The objective is to choose a start time for each activity of the project such as the project duration is minimized while satisfying precedence relations, resource availabilities and resource-transfer time constraints. We propose a new genetic algorithm using a two-point crossover operator. Experiment conducted on a large number of instances shows that the proposed algorithm performs better than the solution methods previously published.

3.1 Introduction

The standard version of the *Resource-constrained, project scheduling problem* (RCPSP) consists of scheduling a set of activities (j =1,...,J) linked by zero-lag, finish-to-start precedence relations while pre-emption is not allowed. Resources required to perform activities are renewable and available in limited amounts at each period. Each activity has a unique and known duration and uses a constant amount of the resources for each period of its execution. The objective of the RCPSP is to minimize the project duration (make-span) while respecting precedence and resource constraints. This problem is known to be NP-hard as shown by Blazewicz et al. (1983).

To solve the RCPSP to optimality, various branch and bound procedures have been developed (Patterson et al., 1990; Carlier and Latapie, 1991; Demeulemeester and Herroelen, 1996;

Brucker et al., 1998). However, none of this technique is capable of solving large size realistic problem in reasonable computational time.

Hence, several authors propose heuristic approaches to solve the RCPSP. Most of these methods belong to two categories, priority rule based heuristics and neighbourhood search heuristics. The first category are construction heuristics which start with an empty schedule and add activities one by one until all activities are scheduled. This operation is controlled by a scheduling scheme and a priority rule. Two different scheduling schemes called serial and parallel (Kelley, 1963; Brooks and White, 1965) are widely used to develop most heuristic techniques published in the literature (Boctor, 1990; Kolisch, 1996a,b; Kolisch and Drexl, 1997). In order to improve a feasible schedule obtained by some constructive heuristic, various neighbourhood search methods have been developped like Simulated annealing (Boctor, 1996; Bouleimen and Lecocq, 2003), Tabu search (Baar et al., 1999; Nonobe and Ibaraki, 2002), Scatter search (Debels et al., 2006; Mobini et al., 2009) and Genetic algorithm (Alcaraz et al., 2003; Hartmann, 2001; Valls et al., 2004; Mendes et al., 2009; Zamani, 2013).

The standard RCPSP, as defined above, assumes that no transfer times are needed to transfer resources between the sites of execution of different tasks. In practice, however, it happens that moving resources between these sites may take a significant time. For example, heavy construction equipment might be requested at different construction sites and moving them from one site to another may require several hours.

In such a case, modelling and solving the project scheduling problem without taking into consideration transfer times might produce unfeasible schedules. Recently Krüger and Scholl (2009) developed methods to solve *the resource-constrained project scheduling problem with transfer times* (RCPSPTT) for the single and multi-project cases. The authors formulated both problems as integer linear models and developed a priority rule based heuristic solution approach. Then Krüger (2009) developed a genetic algorithm for this problem. Recently, a flow based tabu search algorithm for the RCPSP with transfer times was developed by Poppenborg and Knust (2014). Hereafter we propose a new and efficient genetic algorithm to solve the problem.

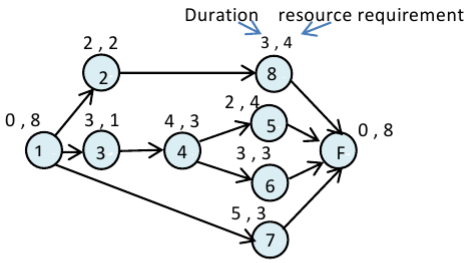
The remainder of this paper is organized as follows. Section 3.2 describes the Resource constraint project scheduling problem with transfer times and section 3.3 presents a mathematical formulation for it. Section 3.4 describes the proposed genetic approach and section 3.5 reports and discusses the results of the conducted evaluation experiment. Section 3.6 concludes this research work and opens on possible extensions.

3.2 Problem description

In the considered RCPSPTT we have a set J of activities, a set P of zero-lag, finish-to-start precedence relations between these activities, a set K of renewable resources and the limits on resources availability are denoted $Q_k; k \in K$. Each activity i has a set of immediate predecessors $P_i \subseteq J$, a set of immediate and indirect predecessors, denoted $\pi_i \subseteq J$, a set of immediate successors $Z_i \subseteq J$ and a set of immediate and indirect successors, denoted $\sigma_i \subseteq J$. Each activity has a unique and known duration d_i and uses a constant amount of resources $q_{ik}; \forall k \in K$, for each period of its execution. We also define $I_i = [EST_i, LST_i]$, the time window between the earliest and the latest start times of activity i. These times are calculated using the critical path method while using a feasible project duration denoted T. We assume that the transfer of a unit or several units of renewable resource k from the execution location of activity i to the execution location of activity j requires a transfer time denoted Δ_{ijk}. All transfer times are assumed to fulfil the triangular inequality; i.e., $\Delta_{ijk} \leq \Delta_{ilk} + \Delta_{ljk}. \forall i, j, l \in J$.

Without loss of generality, we assume that project activities include a dummy start activity (activity 1) and a dummy finish activity (activity F). These two activities have a zero duration and require the entire available amount Q_k of every resource k; (i.e., $q_{1k} = q_{Fk} = Q_k; \forall k \in K$). This is because the dummy start activity should provide resources to other activities while the dummy finish activity should collect them back. Transfer times from the dummy start activity and to the dummy finish activity are assumed nil (i.e., $\Delta_{1ik} = \Delta_{iFk} = 0; \forall i \in J, k \in K$). The objective in RCPSPTT is to construct a schedule that minimizes the project duration while satisfying precedence constraints, resource constraints and transfer time constraints.

Figure 3.1 shows an example of a project comprising 9 activities to be scheduled using one resource type with eight units available. The transfer time matrix Δ_{ij} is also given in Figure 3.1. The optimal schedule of this example with a make-span of 13 time-periods is shown in Figure 3.2 where arrows indicate transfer of resource units.



$$\Delta = \begin{Bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 1 & 1 & 1 & 2 & 2 & 0 \\ 0 & 2 & 0 & 1 & 2 & 1 & 2 & 1 & 0 \\ 0 & 1 & 1 & 0 & 2 & 1 & 2 & 2 & 0 \\ 0 & 1 & 2 & 2 & 0 & 1 & 2 & 2 & 0 \\ 0 & 1 & 1 & 1 & 1 & 0 & 2 & 2 & 0 \\ 0 & 2 & 2 & 2 & 2 & 2 & 0 & 2 & 0 \\ 0 & 2 & 1 & 2 & 2 & 2 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{Bmatrix}$$

Figure 3.1: a RCPSPTT example.

3.3 Mathematical formulation

In addition to the notation presented above, we need to use three decision variables, first we use a binary variables x_{it} that take the value 1 if and only if activity i starts at the beginning of time period t. we also use another binary variable, y_{ijk}, that takes the value 1 if and only if at least one unit of resource k is transferred from activity i to activity j. Finally, f_{ijk} gives the number of units of resource k to transfer from activity i to activity j.

Using this notation the mathematical model of the RCPSPTT can be written as follows (also see Krüger and Scholl (2009)):

$$\text{Miminize} \sum_{t \in I_i} t.x_{Ft} \tag{3.1}$$

$$\sum_{t \in I_i} x_{it} = 1 \qquad \forall i \in J \tag{3.2}$$

$$\sum_{t \in I_j} t x_{jt} - \sum_{t \in I_i} (t + d_i) x_{it} \geq 0 \qquad \forall (i,j) \in P \tag{3.3}$$

$$\sum_{t \in I_j} t x_{jt} - \sum_{t \in I_i} (t + d_i) - (T + \triangle_{ijk}) y_{ijk} \geq -T \qquad \forall i \in J - F, j \in J - \pi_i \tag{3.4}$$

$$f_{ijk} \leq \min(q_{ik}, q_{jk}) y_{ijk} \qquad \forall i \in J - F, j \in J - \pi_i \tag{3.5}$$

$$\sum_{i \in J-\sigma_i} f_{ijk} = q_{jk} \qquad \forall j \in J - \pi_i \tag{3.6}$$

$$\sum_{j \in J-\pi_i} f_{ijk} = q_{ik} \qquad \forall i \in J - F \tag{3.7}$$

$$x_{it} \in \{0,1\} \qquad \forall i \in J, t \in I_i \tag{3.8}$$

$$y_{ijk} \in \{0,1\} \qquad \forall i \in J - F, j \in J - \pi_i \tag{3.9}$$

$$f_{ijk} \geq 0 \qquad \forall i \in J - F, j \in J - \pi_i \tag{3.10}$$

The objective function (3.1) minimizes the project duration. Constraints (3.2) ensure that each activity is executed without pre-emption and start within the time window between its earliest and latest start time. Constraints (3.3) are the precedence constraints and constraints (3.4) ensure that the transfer time of the resource k is taken into consideration when some of its units are transferred from activity i to activity j. Constraint (3.5) imply that if $y_{ijk} = 0$ then $f_{ijk} = 0$.

Otherwise f_{ijk} takes a value less than or equal to both the number of units required to execute the activity j and the number of units available at i. Constraints (3.6) and (3.7) are the resources flow conservation constraints. Constraints (3.8); (3.9) and (3.10) define the decision variables of the model.

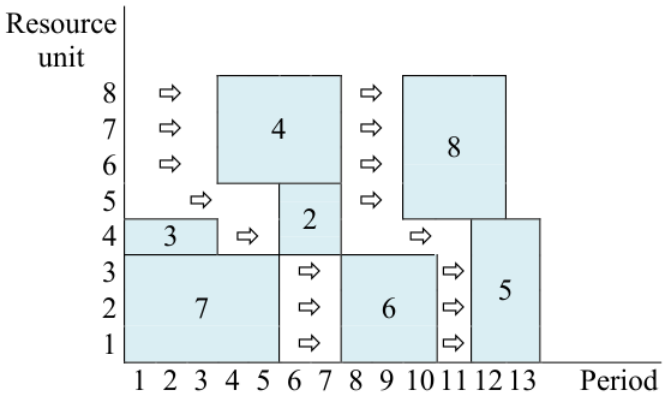


Figure 3.2: Optimal solution of the RCPSPTT example.

3.4 The proposed genetic algorithm

In this section we develop a new genetic algorithm GA to solve the RCPSPTT. Our GA starts with an initial population generated through a modified version of the regret-based biased random sampling mechanism. The original regret-based biased random sampling procedure was proposed by Kolisch and Drexl (1997). The size of the population is constant and equal to

G. The genetic algorithm then determines the fitness value of each individual, randomly selects a number of pairs of individuals (solutions), and uses a Two-point Crossover operator (TPCO) to generate two new individuals (offspring solutions) from each selected pair of parent solutions. This crossover operator can be seen as a modification of the two-point operator proposed by Zamani (2013) who called it the Magnet-Based Crossover (MBX) operator. Afterwards a mutation operation is applied to some of the newly generated individuals.

After computing the fitness of each new individual, we add offspring individuals to the current population. Then, among the G parents and the G offspring solutions, we select the best G individuals and place them in the next generation. This process is repeated until the total number of solutions generated by the algorithm reaches a pre-selected number L. The pseudo-code of the algorithm is presented in Figure 3.3 below.

1: Generate an initial population of G individuals using the modified regret-based biased random sampling
2: $N_{solutions} = G$
3: **WHILE** ($N_{solutions} \leq L$)
- Generate G offspring individuals from current population using the proposed TPCO.
- Apply mutation to some of the generated individuals.
- Determine the fitness (project duration) of each offspring solution.
- $N_{solutions} = N_{solutions} + G$
- Add the new G solutions to the previous generation of G solutions.
- Select the best G individuals to be the next generation.

END WHILE
4: Retain the best obtained solution

Figure 3.3: The developed GA.

3.4.1 Solution coding

The genetic algorithm described in this paper uses an extended activity list representation of solutions, where each individual is represented (coded) by $|J| + 2$ genes as shown in Figure 3.4. The first $|J|$ genes give a precedence-feasible activity list (sequence) meaning that each activity should be placed after all its predecessors. After that, two extra binary genes indexed $|J|+1$ and $|J|+2$ are added. The first one indicates the scheduling generation scheme to use in order to generate the corresponding feasible schedule. In this work the serial (S) and parallel (P) scheduling schemes, as adapted by Krüger and Scholl (2009) to the RCPSP with transfer times, are used to construct the schedule (these two algorithms are presented in subsection

3.4.3). The second binary gene indicates the scheduling direction (forward or backward F/B) to be used while generating the schedule.

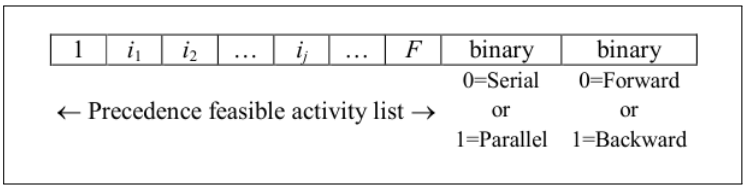


Figure 3.4: Solution coding (representation).

3.4.2 Generation of the initial population

The first step of the genetic algorithm consists of generating an initial population of G individuals. As mentioned before, each individual is composed of an ordered activity list (sequence of activities), a binary indicating the type of scheduling scheme (serial or parallel) to be used to translate the list into a feasible schedule, and another binary indicating the direction (forward or backward) to be followed while building this schedule. The ordered activity list is obtained by a modified version of the regret-based biased sampling heuristic described in Figure 3.5 below. Within this heuristic, three priority rules are used: the LFT (latest finish time), LST (latest start time) and the MINSLK (minimum total slack) rules. This biased sampling method is an iterative heuristic that iteratively determines the set of eligible activities (whose predecessors are already included in the sequence under construction), randomly selects one of them and adds it to the sequence. The probability of selecting an activity j equals $\Psi_j = r_j / \sum_{i \in S} r_i$, where r_j, called the regret value of activity j, depends on its priority and S is the set of eligible activities. This process is repeated until a complete precedence-feasible sequence is obtained. Afterwards, two binaries are randomly generated and added to complete the code representing the individual.

3.4.3 Transforming a sequence into a schedule

Once an individual solution (coded by a sequence of activities and the two additional binaries as shown in Figure 3.4) is generated, we need to construct the corresponding feasible schedule and to calculate its fitness value. The fitness value of a solution is given by the duration of the corresponding schedule.

To construct a feasible schedule, we use an adaptation of either the serial or the parallel scheduling scheme (Kelley, 1963; Krüger and Scholl, 2009). Before we present our adaptation of these two scheduling schemes, let us introduce the following notation:

x_{ijk} : Quantity of resource k transferred from activity i to activity j

1: Initialize the activity sequence : $\sigma = \{1\}$
2: Randomly choose a priority rule among the three rules (LFT, LST and MINSLK)
3: Let S is the set of activities $S = \{\forall i \in J, P_i = \{1\}\}$
4: **WHILE** $(S \neq \emptyset)$
- Calculate $p_j, \forall j \in S$, the priority of eligible activity according to the rule selected p_j.
- For each eligible activity j, calculate r_j, its regret value, and ψ_j its selection probability, where: $r_j = 1 - (p_i - \max_{i \in S} p_i)$ and $\psi_j = r_j / \sum_{i \in S} r_j$.
- Based on the probability ψ_j and using the simple roulette method, select an activity among the eligible ones. Let k be the selected activity. Add k to the sequence; $\sigma = \sigma + \{k\}$.
- Up date S, the set of eligible activities S, $S = \{\forall \in J \; P_i \subseteq \sigma\}$

END WHILE

Figure 3.5: Modified-regret based biased random sampling procedure.

y_{ik} : Number of units of k available at the site of activity i

D : Set of activities scheduled to start by the beginning of t

C : Set of activities scheduled to finish by the beginning of t

A : Set of activities in progress at period t

FT_i : Finish time of activity i

Z_{jk} : Set of activities that can provide units of k to activity j

Γ_{jk} : Set of pair of activities (i,l) such that i provide units of k to l and that activity j can be placed in between and uses some of these units of k before passing them to l

Seriel scheduling scheme

Figure 3.6 presents the forward serial scheduling scheme. The sequence representing the solution is an input to this procedure. The procedure sequentially adds activities to the schedule until a complete feasible schedule is obtained. At each iteration, the next activity j in the sequence is selected and we determine t its earliest possible start time.

All activities i which precede activity j in the list and can deliver at least one unit of resource type k to activity j at time t or earlier are considered and added to a set denoted Z_{jk}. Also, we identify all pairs of scheduled activities i and l such that activity j can be inserted between them and we can break the resource flow between these two activities to deliver some resource units from i to j which transfer them later to l without causing any delay for activity l. All such breakable flows are identified and the corresponding pairs of activities i and l are added to a set denoted Γ_{jk}. Thus providing the required units of resource k to j at time t can be

either from activities belonging to Z_{jk} or by breaking some flows between pairs of activities belonging to Γ_{jk}. The amount of resource k that can be provided to j at period t, denoted s^{jk}, is given by $s^{jk} = \sum_{i \in Z_{jk}} y_{ik} + \sum_{(i,l) \in \Gamma_{jk}} x_{ilk}$.

If enough resource units are available to schedule activity j at t; i.e., if $s^{ik} \leq q_{jk}; \forall k \in K$, then, activity j can be scheduled to start at the beginning of period t. To provide activity j with resource k, first, we use the breakable flows between pairs of activities (i,l) belonging to Γ_{jk} starting with the pair holding the largest amount of k denoted u_{ilk}. If all such flows are used and still more units of k are needed to perform task j, we use the units available at the sites of activities i belonging to Z_{jk} arranged in the ascending order of their transfer time $\triangle_{ijk}$.

On the other hand, if s^{jk} the transferable amount of resource k is not sufficient to allow starting activity j at the beginning of time period t, then t is increased until sufficient resources become available. The procedure stops when a complete schedule is constructed.

Parallel scheduling scheme

The parallel scheduling scheme is presented in Figure 3.7. It is a straightforward adaptation of parallel scheduling scheme for the standard resource-constrained project scheduling problem proposed by Kelley (1963) and presented by Brooks and White (1965). This iterative procedure uses as input the sequence that we seek to transform into a schedule. In each iteration, the procedure identifies the candidate (schedulable) activities, arrange them according to their order in the sequence, and schedule the activity having the highest priority that can be performed with the available amount of resources. Then it updates the list of candidate activities and if the list is not empty, proceeds to schedule activity in top of the list. In case the candidates list is empty but still there are some non-scheduled activities, the procedure goes to the earliest date where certain activities can be scheduled and select one of them. The procedure ends once all activities are scheduled.

3.4.4 The proposed two-point crossover operator

The uniform crossover and the k-point crossover operators (Djerid et al., 1996) were suggested to handle scheduling problems. However, the one point crossover (Davis (1985)) and two-point crossover (Hartmann, 1998) are usually used in solving the standard RCPSP.

Two types of two-point crossover are suggested and in both types we start by selecting a contiguous block of genes from one of the parent solutions. In the first type, to create an offspring solution, we modify the sequence of activities within the selected block (see Hartmann (2002) and Lova et al. (2009)) and in the second type, we keep the block unchanged and modify the sequence of activities outside this block (see Zamani (2013)). The crossover operator

Initialize: $D = \{1\}$
$FT_i = 0;\ \forall i \in J$
$u_{ijk} = 0;\ \forall i \in J - \{F\}, \forall j \in J - \pi_i$
$v_{1k} = Q_k;\ \forall k \in K : v_{ik} = \Phi;\ \forall i \in J - \{1\},\ \forall k \in K$
Let σ be the sequence to transform into a schedule
FOR λ=2 to $|J|$
Let j be the activity in position λ in the sequence σ and $t = max_{i \in P_j} FT_i$
WHILE $j \notin D$
FOR k =1 **to** $|K|$
$Z_{jk} = \varnothing$
FOR l =1 **to** λ-1
Let i be the activity in position l in the sequence σ
$\Gamma_{jk} = \varnothing$
IF $v_{ik} > 0$ **AND** $FT_i + \Delta_{ijk} \le t$ **THEN** $Z_{jk} = Z_{jk} \cup \{i\}$
FOR e = 1 to λ-1
Let m be the activity in position e in the sequence σ
IF $m \neq i$ **AND** $u_{imk} > 0$ **AND** $FT_i + \Delta_{ijk} \le t \le FT_m - d_j - d_m - \Delta_{jmk}$ **THEN** $\Gamma_{jk} = \Gamma_{jk} \cup (i,m)$
NEXT e
NEXT l
$s^{jk} = \sum_{i \in Z_{jk}} v_{ik} + \sum_{(i,m) \in \Gamma_{jk}} u_{imk}$
NEXT k
IF $s^{jk} \ge q_{jk},\ \forall k \in K$ **THEN**
For k =1 **to** $|K|$
Arrange all breakable-flow activity pairs $(i,m) \in \Gamma_{jk}$ in the descending order of u_{imk}
Arrange all potential delivering activities $i \in Z_{jk}$ in the ascending order of transfer time Δ_{ijk}
$l = 1$
WHILE $v_{jk} < q_{jk}$ **AND** $l \le |\Gamma_{jk}|$
Let (i,m) be the pair in position l in the set Γ_{jk}
$x = min\ (u_{imk},\ q_{jk} - v_{jk})$
$u_{imk} = u_{imk} - x$
$u_{ijk} = u_{ijk} + x : u_{jmk} = u_{jmk} + x : v_{jk} = v_{jk} + x$
$l = l + 1$
END WHILE
$e = 1$
WHILE $v_{jk} < q_{jk}$
Let activity i be the activity in position e in the set Z_{jk}
$x = min\ (v_{ik},\ q_{jk} - v_{jk})$
$v_{ik} = v_{ik} - x$
$u_{ijk} = u_{ijk} + x : v_{jk} = v_{jk} + x$
$e = e + 1$
END WHILE
NEXT k
$D = D \cup \{ j \}$
$FT_j = t + d_j$,
ELSE
$t = t + 1$
END IF
END WHILE
NEXT λ

Figure 3.6: Serial Scheduling scheme for the RCPSPTT.

suggested by Zamani and called Magnet-Based Crossover operator or MBX works as follows. First, a contiguous block of genes is selected from the first parent (called the donor parent) by randomly drawing two numbers n_1 and n_2 ($1 < n_1 < n_2 < |J|$) which become the bounding positions in the donor sequence specifying the required contiguous block. Then, it determines the position of the first and last elements of this block in the sequence of the second parent (called the receiver). Let these position be denoted n_0 and n_3 respectively. The offspring sequence is built by concatenating:

- activities in positions 1 to n_0-1 from the sequence of the receiver;
- activities in between positions n_0 and n_3 in the receiver sequence which are predecessors to any activity in the block;
- activities of the block;
- activities in positions between n_0 and n_3 in the receiver sequence who are successors to any activity in the block; and
- activities in positions between n_3+1 and $|J|$ in the receiver sequence.

However, some of the activities in the positions between n_0 and n_3 in the receiver sequence may neither be a predecessor or a successor to any activity in the block. These activities are called free activities and are randomly placed either before or after the block in the offspring sequence. Zamani (2013) suggest that whenever a free activity is encountered, it is placed before the block with a chance of p to overturn this decision. But once this decision is overturned, the remaining free activities are placed after the block. Having q free activities, Zamini fixes p=0.5 for q=1 and p=2/(q+2) for q>1. Afterwards, the two additional binaries (indicating if a serial or parallel scheme is to be used and the direction of applying it; Forward or backward) are copied from the code of the donor solution.

Our proposed two-point crossover operator is similar to Zamani's operator. However it differs in that free activities are randomly placed either before or after the activities of the block, as long as this does not violate any precedence relation. This is implemented as follows. For every free activity, we determine all possible positions before or after the block, where it can be placed without violating precedence relations. Then we randomly chose one of these positions. This makes our operator more disruptive than Zamani's operator as it introduces more diversification in the process. Our numerical experiment shows that our operator produces better results.

To illustrate the application of the proposed two-point crossover operator described here, we use the numerical example presented in Figure 3.1. Two feasible solution codes are presented in top of Figure 3.8 and are considered as the donor and receiver parents respectively. Assuming that we randomly draw n_1=6 and n_2=8 then the contiguous block to transfer from the donor

Initialize: $t = 1$
$C = \{1\}$ (set of activities completed by time t)
$FT_i = 0$; $\forall i \in J$
$v_{1k} = Q_k$; $\forall k \in K : v_{ik} = \Phi$; $\forall i \in J - \{1\}$, $\forall k \in K$
$\sigma = \{1, i_1, i_2, .., F\}$ the sequence to transform into a schedule
$S = \{\forall i \in J \mid P_i = C\}$ (set of candidate activities at time t)
$D = \{1\}$ (set of scheduled activities)
$a_{kt} = Q_k$; $\forall k \in K$, $t = 1, .., T$ (units of k still available at t)
Arrange activities in the list S according to their order in σ

WHILE $|D| < |J|$
  WHILE $S \neq \varnothing$
    Let j be the first activity in the ordered list S
    FOR $k = 1$ **to** $|K|$
      $Z_{jk} = \varnothing$
      For $l = 1$ **to** $|C|$
        Let i be the activity in position l in the set C
        IF $v_{ik} > 0$ **AND** $FT_i + \Delta_{ijk} \leq t$ **THEN** $Z_{jk} = Z_{jk} + \{i\}$
      NEXT l
    NEXT k
    IF $\sum_{i \in Z_{jk}} v_{ik} \geq q_{jk}$, $\forall k \in K$ **THEN**
      FOR $k = 1$ **to** $|K|$
        Arrange activities $i \in Z_{jk}$ in the ascending order of their transfer time Δ_{ijk}
        $e = 1$
        WHILE $v_{ik} < q_{jk}$
          Let activity i be the activity in position e in the set Z_{jk}
          $x = min\,(v_{ik}\,,\, q_{jk} - v_{jk}) : v_{ik} = v_{ik} - x : u_{ijk} = u_{ijk} + x : v_{jk} = v_{jk} + x$
          $e = e + 1$
        END WHILE
      NEXT k
      $D = D \cup \{j\}$
      $FT_j = t + d_j$,
      FOR $\tau = t$ to FT_j
        FOR $k = 1$ to $|K|$
          $a_{kt} = a_{kt} - q_{jk}$
        NEXT k
      NEXT τ
    END IF
      $S = S - \{j\}$
  END WHILE
    $t = t + 1$
    Update C the list of activities completed by time t; $C = \{\forall j \in J \mid FT_j < t\}$
    $S = \{\forall j \in J - D \mid P_j \subseteq C;\ q_{jk} \leq a_{kt};\ \forall k \in K\}$ (the list of candidate activities at time t)
  IF $S \neq \varnothing$ **THEN**
    Arrange activities in the list S according to their order in σ
  END IF
END WHILE

Figure 3.7: Parallel Scheduling scheme for the RCPSPTT.

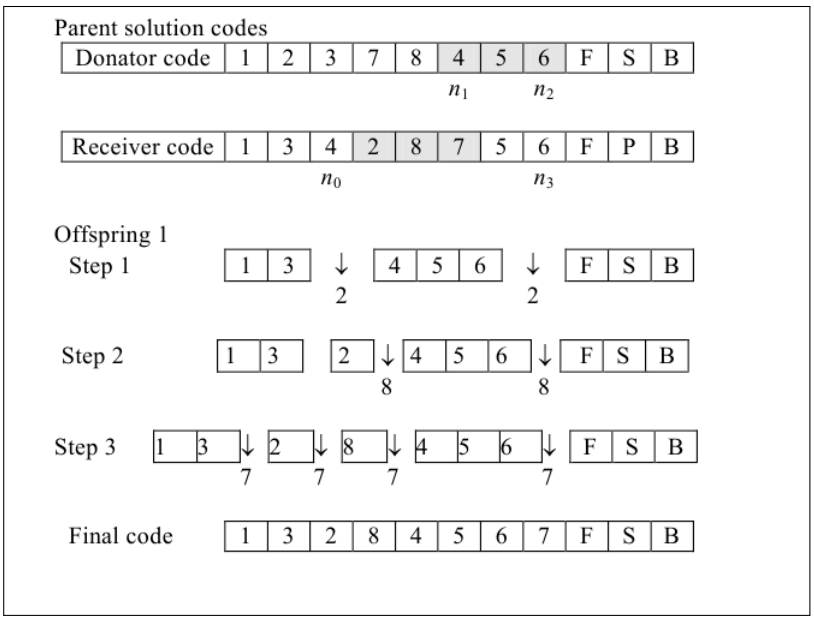


Figure 3.8: Application of the proposed two-position crossover.

parent to the offspring is composed respectively from activities 4, 5 and 6 (activities in positions 6, 7 and 8 in the donor sequence). Consequently, the positions n_0 and n_3 in the receiver parent are respectively position 3 (where we have activity 4) and 8 (where we have activity 6).

Now we can start building the activity sequence for the offspring as follows. We copy from the receiver all activities between positions 1 and n_0-1 (i.e., between 1 and position 2) as well as the activity in position n_3+1 (i.e., position 9 which is also the last position in the sequence). Activities of the block are then placed in between. The three remaining activities 2, 8 and 7 are free activities as they are neither predecessors nor successors to any activity in the block. These free activities are then added to the offspring sequence one by one as follows. Activity 2 can be placed in the offspring sequence either between activity 3 and activity 4 (just before the block) or between activity 6 and activity F (just after the block). Assume that we (randomly) decided to place it between activities 3 and 4. Then, as activity 8 is an immediate successor of activity 2 and immediate predecessor of activity F, it can be inserted either between activities 2 and 4 or between 6 and F. Suppose that we (randomly) placed it between activities 2 and 4, then four positions are now possible for activity 7 (see Figure 3.8). We randomly place activity 7 between activities 6 and F. Finally, we copy the two last binaries from the donor and we obtain the offspring code 1-3-2-8-4-5-6-7-F-0-0 as shown in the figure.

Notice that from every pair of parent solutions we generate two offspring solutions. To generate the first offspring, we use the first parent solution as donor and the second as receiver while to generate the second offspring we consider the second parent as donor and the first one as

receiver.

3.4.5 Mutation

Once all offspring solutions are generated, a mutation operator is applied to every new individual as follows. We randomly select a position, say i, in the activity sequence ($i.e., 1 < i < |J| - 2$) and draw a random number x_1 from a uniform distribution between 0 and 1. Then we permute the jobs in positions i and $i+1$ if the activity in position i is not an immediate predecessor of the activity in position $i+1$ and $x_1 < \mu$, the mutation probability. In our implementation of the GA we used $\mu = 0.05$. We may also mute one or both of the last two binaries in the code with a mutation probability equals μ. Thus we draw a random number x_2 and flip the binary indicating whether we use the serial or the parallel scheme to construct the project schedule if $x2 < \mu$ also, we draw another random variable x_3 and change the last binary of the code if $x_3 < \mu$.

3.5 Computational results

3.5.1 Test instances

Two sets of instances were used to assess the performance of the proposed genetic algorithm. The instances of the first set are generated by the instance generator PROGEN (Kolisch and Sprecher, 1997)with no transfer times. These instances were then solved to optimality and transfer times were added in such way that optimal solutions without transfer times remain optimal after adding these times. Notice that the test instances used to test the genetic algorithm developed by Krüger (2009), Krüger and Scholl (2009) and those used to test the Tabu search algorithm developed by Poppenborg and Knust (2014)), were generated in the same way. These researchers used this special scheme to add transfer times as they expected that it should be difficult to solve the RCPSP with transfer times to optimality. For the second set, transfer times are drawn randomly from a uniform distribution either between 0 and 5 or between 0 and 10 time periods. This second set is composed of instances that we were able to solve to optimality.

Instances of the first set

The first set, composed of a total of 869 instances, is divided into two sub-sets: a subset composed of the 480 J30-instances and a second subset containing 389 instances among the 480 instances of the set J60-instances generated by the instance generator PROGEN (see Kolisch and Sprecher (1997)). These instances are in the project scheduling problem library PSPLIB (http://www.om-db.wi.tum.de/psplib/main.html).

To these instances we added transfer times in a way that makes optimal solutions of these instances with and without transfer times remain the same. This way we used to generate transfer times is similar to what has been done by Krüger (2009) and Poppenborg and Knust (2014). To add transfer times we first solve to optimality the instance without transfer times. To obtain these optimal solutions, we used the mathematical formulation of Pritsker et al. (1969) and the commercial MIP code GUROBI 5.0.1. Then, in the obtained optimal solutions, we determined for each activity the set of activities from which we can transfer the necessary resources. Then we added randomly generated transfer times that are less than or equal to the difference between the finish time of the activities providing resources needed and the start time of the activity requiring these resources.

The instances we used in our numerical experiment are those for which we were able to obtain their optimal solution. The used MIP code was able to solve all the J30 instances but not all the J60 instances. So, we used only 389 of the J60 instances for which the optimal solution was found. Krüger (2009) indicated that they were able to get the optimal solution for 400 among the 480 instances of the J60 set. However, their solutions are not provided anywhere.

Table 3.1 gives the characteristics of these instances. For all these instances the network complexity (NC) is between 1.5 and 2.1 while the resource factor (RF) is between 0.25 and 1 and the resource strength (RS) is between 0.25 and 1. The network complexity NC is measured by the average number of immediate predecessors in the network while the resource factor RF indicates the portion of resources consumed. The table also indicates the resources strength which is a scaling parameter expressing the resource availability as a convex combination of the sum of the minimum amount of the resources that allows to schedule the project and the amount required to schedule activities at their earliest start time.

Instances with randomly generated transfer times

This second set is composed of 705 instances generated by adding random transfer times to some of the J30 instances provided by the project scheduling library PSPLIB mentioned above. Added transfer times are drawn from a uniform distribution between either 0 and 5 time units or between 0 and 10 time units. All of these instances involve 4 types of renewable resources ($K = 4$) with a network complexity (NC) between 1.5 and 2.1 and the resource factor (RF) of 0.25. The resource strength (RS) is 0.5, 0.7 or 1.0. Table 3.2 gives the characteristics of these instances. As the table indicates, the optimal solution of 584 instances (83% of the 705 instance) is not the same as the one with no transfer times. Notice that we generated many other instances with random transfer times but we were only able to solve these instances to optimality. To obtain the optimal solution of these instances with random transfer times we use the MIP commercial code GUROBI 5.0.1 and the mathematical formulation given in

Set	Network Complexity (NC)	Resource Factor (RF)	Resource Strength (RS)	Total number of instances
$J30$	[1.5 , 2.1]	[0.25 , 1]	0.25	120
	[1.5 , 2.1]	[0.25 , 1]	0.50	120
	[1.5 , 2.1]	[0.25 , 1]	0.75	120
	[1.5 , 2.1]	[0.25 , 1]	1.0	120
All				480
$J60$	[1.5 , 2.1]	[0.25 , 1]	0.25	35
	[1.5 , 2.1]	[0.25 , 1]	0.50	114
	[1.5 , 2.1]	[0.25 , 1]	0.75	120
	[1.5 , 2.1]	[0.25 , 1]	1.0	120
All				389

Table 3.1: Some characteristics of instances of the first set

section 3.3.

Transfer times range	NC	RF	RS	Total number of instances	Number of instance where the optimal solution with random transfer time is different of that without transfer times
0 to 5	1.5 to 2.1	0.25	0.5	112	112
			0.7	120	103
			1.0	120	60
			ALL	352	275
0 to 10	1.5 to 2.1	0.25	0.5	113	113
			0.7	120	116
			1.0	120	80
			ALL	**353**	**309**
ALL				**705**	**584**

Table 3.2: Some characteristics of instances of the second set

3.5.2 Results obtained by the proposed genetic algorithm

All instances are solved using a computer with i7 core processor, 2 GHz and 6 GB of internal memory. The proposed genetic algorithm is implemented in MATLAB release 2014 with a limit of 5000 visited solutions.

Results for the first set of instances

Table 3.3 gives the percentage deviation from the optimal solution for the J30 and J60 instances of the first set. From this table we can see that the proposed GA obtained an average deviation from the optimum of 0.12% and 0.36% for the J30 and J60 instances respectively. From this table, it seems that the percentage deviation from the optimum depends mainly on two factors: the number of activities and the resource strength coefficient RS. First, the percentage deviation increases when the number of activities increases. This is expected as the number of feasible solutions also increases when the number of activities increases and consequently it is more difficult to find the best one among this larger number of feasible solutions. Second, the percentage deviation increases when the resource strength decreases. This can be explained as the number of feasible solutions increases when the resource strength decreases.

Set	*NC*	*RF*	*RS*	Number of instances	Average deviation from optimum	Number of times the optimum is found
J30	[1.5 , 2.1]	[0.25 , 1]	0.25	120	0.39	105 (87.5%)
	[1.5 , 2.1]	[0.25 , 1]	0.50	120	0.11	114 (95%)
	[1.5 , 2.1]	[0.25 , 1]	0.75	120	0.00	120 (100%)
	[1.5 , 2.1]	[0.25 , 1]	1.00	120	0.00	120 (100%)
All J30 instances				**480**	**0.12**	**459 (95.6%)**
J60	[1.5 , 2.1]	[0.25 , 1]	0.25	35	1.13	22 (62.8%)
	[1.5 , 2.1]	[0.25 , 1]	0.50	114	0.887	68 (59.6%)
	[1.5 , 2.1]	[0.25 , 1]	0.75	120	0.00	120 (100%)
	[1.5 , 2.1]	[0.25 , 1]	1.00	120	0.00	120 (100%)
All J60 instances				**389**	**0.36**	**330 (84.8%)**
All instances				**869**	**0.23**	**789 (90.8 %)**

Table 3.3: Average percentage deviation from the optimum as obtained by the proposed genetic algorithm

Table 3.4 compare the results of the proposed GA with those of the genetic algorithm of Krüger (2009) and of the Tabu search algorithm of Poppenborg and Knust (2014). We notice that Krüger's genetic algorithm has a limit of 5000 visited solutions while Poppenborg and Knust's tabu search algorithm has a limit of 10 000 visited solutions. Our GA is limited to 5000 visited solutions. A corresponding average computational time of 153 sec and 320 sec was obtained for the J30 and J60 instances respectively. The results in Table 3.4 show that the genetic algorithm developed in this paper gives the best results outperforming the two other solution heuristics.

Procedure	J30	J60
GA Proposed in this work	**0.12**	**0.36**
Krüger's GA (2009)	0.16	0.38
Popenborg and Knust's TS (2014)	1.27	0.98

Table 3.4: Average percentage deviation from the optimum of instances: a comparison with the published solution methods.

Results for the instances with random transfer times

The proposed genetic algorithm (GA) was able to find the optimal solution for all the test instances with random transfer times. However, these instances are probably less difficult to solve. It is not sure that the proposed GA will always obtain the optimal solution for instances with a resource factor (RF) higher than 0.25, or instances with tighter resource strength (e.g. $RS = 0.25$), or those with a larger number of activities. Unfortunately, we were not able to obtain optimal solutions for instances with $RF > 0.25$ or with $RS = 0.25$.

3.6 Conclusion

In this paper, the Resource Constrained Project Scheduling Problem with sequence dependent Transfer Times (RCPSPTT) is studied. A genetic algorithm using a two-point crossover operator is proposed and experimental results show that this GA is able to efficiently solve the problem. This GA has been compared to methods published in the literature using a set of instances constructed by the PROGEN project generator to which we added transfer times. The obtained results show that the proposed genetic algorithm outperforms the genetic algorithm developed by Krüger (2009) as well as the Tabou search algorithm of Poppenborg and Knust (2014). Further research is obviously needed to develop more efficient optimal solution procedures for the RCPSPTT. The development of an optimal solution procedure capable of solving larger and more difficult test instances will allow us to provide a more complete assessment of the performance of the proposed GA algorithm.

3.7 References

Alcaraz, J., M, C., R, R., 2003. Solving the multi-mode resource-constrained project scheduling problem with genetic algorithms. Journal of the Operational Research Society 54 (6), 614–626.

Baar, T., Brucker, P., S, K., 1999. Tabu search algorithm and lower bounds for the resource-constrained project scheduling problem. In: Voss et al. (eds) Meta heuristics : Advances and trends in local search paradigms for optimization. Springer, pp. 1–19.

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

Boctor, F. F., 1990. Some efficient multi-heuristic procedures for resource-constrained project scheduling. European Journal of Operational Research 49 (1), 3–13.

Boctor, F. F., 1996. Resource-constrained project scheduling by simulated annealing. International Journal of Production Research 34 (8), 2335–2351.

Bouleimen, K., Lecocq, H., 2003. A new efficient simulated annealing algorithm for the resource-constrained project scheduling problem and its multiple mode version. European Journal of Operational Research 149 (2), 268–281.

Brooks, G., White, C., 1965. An algorithm for finding optimal or near optimal solution to the production scheduling problem. Journal of Industrial Engineering 16 (1), 34 – 40.

Brucker, P., Knust, S., Schoo, A., Thiele, O., 1998. A branch and bound algorithm for the resource-constrained project scheduling problem. European Journal of Operational Research 107 (2), 272–288.

Carlier, J., Latapie, B., 1991. Une méthode arborescente pour résoudre les problèmes cumulatifs. Revue française d'automatique, d'informatique et de recherche opérationnelle. Recherche opérationnelle 25 (3), 311–340.

Davis, L., 1985. Job shop scheduling with genetic algorithms. In: Proceedings of the 1st International Conference on Genetic Algorithms. pp. 136–140.

Debels, D., De Reyck, B., Leus, R., Vanhoucke, M., 2006. A hybrid scatter search/electromagnetism meta-heuristic for project scheduling. European Journal of Operational Research 169 (2), 638–653.

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

Djerid, L., Portmann, M.-C., Villon, P., 1996. Performance analysis of permutation cross—over genetic operators. Journal of Decision Systems 5 (1-2), 157–177.

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

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

Hartmann, S., 2002. A self-adapting genetic algorithm for project scheduling under resource constraints. Naval Research Logistics (NRL) 49 (5), 433–448.

Kelley, J., 1963. The critical-path method : Resources planning and scheduling. In: Muth, J. F., Thomson, G. L. E. (Eds.), Industrial Planning Scheduling. Prentice-Hall, New Jersey, pp. 347 – 365.

Kolisch, R., 1996a. Efficient priority rules for the resource-constrained project scheduling problem. Journal of Operations Management 14 (3), 179 – 192.

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

Kolisch, R., Drexl, A., 1997. Local search for nonpreemptive multi-mode resource-constrained project scheduling. IIE transactions 29 (11), 987–999.

Kolisch, R., 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., 2009. Multi-project scheduling with transfers. Ph.D. thesis, University of Jena, Germany.

Krüger, D., 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.

Lova, A., Tormos, P., Cervantes, M., Barber, F., 2009. An efficient hybrid genetic algorithm for scheduling projects with resource constraints and multiple execution modes. International Journal of Production Economics 117 (2), 302–316.

Mendes, J. J. d. M., Gonçalves, J. F., Resende, M. G., 2009. A random key based genetic algorithm for the resource constrained project scheduling problem. Computers & Operations Research 36 (1), 92–109.

Mobini, M. M., Rabbani, M., Amalnik, M., Razmi, J., Rahimi-Vahed, A., 2009. Using an enhanced scatter search algorithm for a resource-constrained project scheduling problem. Soft Computing 13 (6), 597–610.

Nonobe, K., Ibaraki, T., 2002. Formulation and tabu search algorithm for the resource constrained project scheduling problem. In: Essays and surveys in metaheuristics. Springer, pp. 557–588.

Patterson, J. H., Brian Talbot, F., Slowinski, R., Węglarz, J., 1990. Computational experience with a backtracking algorithm for solving a general class of precedence and resource-constrained scheduling problems. European Journal of Operational Research 49 (1), 68–79.

Poppenborg, J., Knust, S., 2014. A flow-based tabu search algorithm for the rcpsp with transfer times. In: Proceedings of the 14th International Conference on Project Management and Scheduling, March 30th– April 2nd 2014. pp. 181 – 184.

Pritsker, A. A. B., Waiters, L. J., Wolfe, P. M., 1969. Multiproject scheduling with limited resources: A zero-one programming approach. Management Science 16 (1), 93–108.

Valls, V., Ballestín, F., Quintanilla, S., 2004. A population-based approach to the resource-constrained project scheduling problem. Annals of Operations Research 131 (1-4), 305–324.

Zamani, R., 2013. A competitive magnet-based genetic algorithm for solving the resource-constrained project scheduling problem. European Journal of Operational Research 229 (2), 552 – 559.

Chapitre 4

Problème d'ordonnancement de projet avec pusieurs modes d'execuion et des temps de transfert des ressources

Cet article traite le problème d'ordonnacement de projet avec plusieurs modes d'exécution et temps de transfert des ressources. Un temps de transfert est le temps nécessaire pour transférer une ressource entre les différents lieux d'exécution des tâches du projet. Ainsi, le temps de transfert d'une ressource dépend de l'emplacement des tâches à exécuter, ainsi que des caractéristiques de la ressource à transférer.

Dans le problème étudié, nous supposons que la préemption est non autorisée, les ressources utilisées sont renouvelable et non renouvelable, chaque activité a plusieurs modes d'exécution, et les relations de préséance sont de type début-fin sans décalage. Aussi, nous supposons que les durées des activités et les temps de transfert des ressources sont connues et déterministes.

L'objectif est de choisir un temps de début et un mode d'exécution pour chaque tâche du projet, tels que la durée du projet est minimisée tout en respectant les contraintes de préséance, de disponibilité de ressources et les temps de transfert.

Récemment, Krüger and Scholl (2009), Krüger (2009) and Poppenborg and Knust (2014) ont étudié ce problème quand les tâches peuvent avoir un seul mode d'exécution. Ils ont proposé des procédures de résolution et ils ont présenté les résultats numériques pour évaluer la performance des approches proposées.

Ci-après, nous abordons ce problème dans un contexte multi mode. Au meilleur de notre connaissance, cette version du problème n'a jamais été abordé auparavant. D'abord, nous décrivons le problème considéré et nous proposons une formulation mathématique de celui-ci.

Ensuite, nous présentons un algorithme génétique que nous avons conçu pour résoudre les instances de grandes tailles. Enfin, nous donnons les résultats d'une expérience menée pour évaluer la performance de l'approche génétique proposée.

Une version préliminaire de cet article intitulé *Multi-Mode Resource Constrained Project Scheduling with sequence dependent Transfer Times* a été acceptée et publié dans les actes de la conférence internationale PMS 2014 (Project Management and Scheduling), 30 mars au 2 avril 2014, Munich, Allemagne.

Article 2 : Multi-Mode Resource Constrained Project Scheduling with sequence dependent Transfer Times

Abstract

This paper deals with the Multi-Mode Resource-Constrained Project Scheduling Problem with sequence dependent Transfer Times (MRCPSPTT). A sequence dependent transfer time is the time needed to transfer a required resource from the site of the preceding activity to the site of the succeeding one. Thus transfer time of a resource depends on the locations of the activities to execute as well as on the characteristics of the resource to transfer. In the studied problem we assume that pre-emption is not allowed, the used resources are either renewable or non-renewable, each activity has a known and discrete number of execution modes, and precedence relations are zero-lag finish-to-start relations. Also we assume that activity durations and resource transfer times are known and deterministic. The objective is to choose a start time and an execution mode for each activity of the project such that the project duration is minimized. Recently, Krüger and Scholl (2009), Krüger (2009) and Poppenborg and Knust (2014) considered the single-mode resource-constrained project scheduling problem with transfer times (RCPSPTT). They proposed some solution procedures and presented numerical results to assess their performance. Hereafter we address the Multi-Mode Resource-Constrained Project Scheduling Problem with sequence dependent Transfer Times (MRCPSPTT). To the best of our knowledge, this version of the problem has never been addressed before. First we describe the considered problem and present a mathematical formulation of it. Then we introduce a genetic algorithm we designed to solve large scale instances. Finally, we give the results of an experiment conducted to assess the performance of the proposed genetic approach.

4.1 Introduction and literature review

The Resource Constrained Project Scheduling Problem (RCPSP) has received a lot of research attention during the last fifty years. This scheduling problem can be found in diverse practical settings such as road construction, software development, research and development projects, ship building, and many other real-life situations. The standard version of the RCPSP consists

of determining start times for a set of activities so that the project duration is minimized. Activities are linked by zero-lag, finish-to- start precedence relations. Resources required to perform activities are available in discrete and limited amounts at each period. Each activity has a unique and known duration and uses a constant amount of the resources for each period of its execution. Also, non-pre-emption is generally assumed meaning that once an activity is started it cannot be interrupted. This problem is known to be NP-hard (Blazewicz et al., 1983).

In order to model more practical situations, the standard RCPSP has been generalized in several ways. One of the most important generalizations is to allow multiple execution modes for all or some of the project activities. This version of the problem is called the Multi-Mode Resource Constrained Project Scheduling Problem (MRCPSP). In this problem, each activity can be accomplished in one of several modes and each mode is characterized by a known duration and given resource requirements (Elmaghraby, 1995). The objective in MRCPSP is to choose an execution mode to each activity and to determine its start time such that to minimize the overall project duration while satisfying precedence relations and resource availability constraints. Two main categories of resources are considered: renewable and non-renewable resources. Other types of resources can also be considered like double constrained resources and partially renewable resources. As an example of renewable resources we have machines, machine operators and transportation equipment, which have a limited per-period availability. On the other hand, non-renewable resources, such as budget money, are available in limited amounts over the entire project regardless of its duration (Słowinski, 1981).

The MRCPSP is also NP-hard as its special case, the RCPSP, is NP-hard. Moreover, as it is shown by (Kolisch and Drexl, 1997), even answering the question of whether there exists a feasible solution of the MRCPSP with more than one limited non-renewable resource, is an NP-complete problem. Because of its practical importance the MRCPSP has attracted a lot of attention and several authors have developed exact as well as heuristic approaches for solving it.

Talbot (1982) proposed a generalization of the linear integer formulation of the RCPSP of (Pritsker et al., 1969) to model the MRCPSP using the usual time discretization representation. Maniezzo and Mingozzi (1999) proposed a new mathematical formulation for the MRCPSP, which is a generalization of the formulation proposed by Mingozzi et al. (1998) for the RCPSP. The new formulation requires an exponential number of variables, corresponding to all feasible subsets of activities that can be simultaneously executed without violating resource and precedence constraints. More recently, three different formulations were developed by Zapata et al. (2008). The proposed models differ in how time is modeled. The first formulation uses the standard discrete time representation, the second one uses a sequence decision variable in order to eliminate time from the model, and the third formulation uses

a set of events and its decision variables indicate if activities are started or finished at these events. Experimental results reported by the authors indicate that all these formulations are, in general, capable to solve problems of the same size range as the conventional MIP model (Talbot, 1982).

Exact algorithms of the branch and bound type were developed by Talbot (1982), Patterson et al. (1990), Sprecher et al. (1997) and Hartmann and Drexl (1998). As shown by Hartmann and Drexl (1998), none of these procedures are able to find an optimal solution of large instance in a reasonable computation time. Recently, a Branch-and-cut procedure was proposed by Zhu et al. (2006). The developed procedure uses problem-specific cuts derived from resource conflicts and precedence relations, as well as specially designed bounds tightening schemes and branching rules to accelerate convergence. The Branch and Cut procedure was tested on instances from the PSPLIB and was able to find the optimal solution for all instances with 20 activities and for 506 out of 552 instances with 30 activities. Thus, there still exists 30-activity instances for which the optimal solution has not been found (Węglarz et al., 2011).

Heuristic procedure for solving the MRCPSP based on priority rules have been proposed by Boctor (1993), Drexl and Gruenewald (1993) and Boctor (1996a). Drexl and Gruenewald (1993) presented a stochastic scheduling method. This heuristic may generate feasible or infeasible solution and they suggested that it should be applied several times and to retain the best feasible solution. Unfortunately, although the heuristic was applied ten thousand times by Boctor (1993), it failed to solve any of Boctor's 240 test problems.

Boctor (1993) used several parallel and serial scheduling procedures to solve the MRCPSP where only renewable resources are considered. He studied the performance of twenty-one solution procedures and all of them were able to generate feasible solutions whenever such solutions exist. All these heuristics are forward scheduling heuristics as they start by scheduling the activities with no predecessors and do not try to schedule any activity before scheduling all its predecessors. Finally, Boctor (1996a) developed a parallel heuristic that at each step enumerates some schedulable combinations of activities and chooses from them the one having the best value for a proposed criterion.

Many neighborhood search methods are developed to solve the MRCPSP. Kolisch and Drexl (1997) proposed a local search method that first chooses for each activity an execution mode and then performs a neighborhood search to modify the selected modes. Finally a sampling procedure is used to construct a number of solutions and the best one is retained. Simulated annealing approaches are proposed by Słowiński et al. (1994), Boctor (1996b), Józefowska et al. (2001) and Bouleimen and Lecocq (2003). Tabu search procedures are proposed, among others, by Thomas and Salhi (1998), by Baar et al. (1999), and by Nonobe and Ibaraki (2002). Genetic algorithms have been also developed by Mori and Tseng (1997), Özdamar (1999),

Hartmann (2001), Alcaraz et al. (2003), Lova et al. (2009), Van Peteghem and Vanhoucke (2010) and Zamani (2013). Other local search methods like particle swarm optimization have been presented by Zhang et al. (2006) and Jarboui et al. (2008) whereas Merkle et al. (2002) and Chiang et al. (2008) who developed approaches based on ant colony optimization. Finally, Messelis and De Causmaecker (2014) investigate the construction of an automatic selection algorithm to choose the heuristic to use based on the characteristics of the MRCPSP (multi-mode resource-constrained project scheduling problem) to solve.

In many practical situations, activities have to be performed in different locations (sites). Consequently, the required resources have to be transferred between different execution locations. For example, heavy equipment in construction projects might be requested at different construction sites and moving them from one site to other may take a significant transfer time. In such a case, modeling and solving the project scheduling problem as a MRCPSP might produce schedules that do not take into consideration these transfer times. Recently, Krüger and Scholl (2009) developed methods to solve the single-mode, resource-constrained project scheduling problem with transfer times (SRCPSPTT) for the single and multi-project cases. The authors formulate both problems as integer linear models and developed a priority rule-based heuristic solution approach. Recently a genetic algorithm is proposed by Krüger (2009) and Kadri and Boctor (2016) for this problem. Also, Poppenborg and Knust (2014) developed a tabu search solution algorithm for it.

Setup times, which are a special case of transfer times, have already been considered in RCPSP, where generally, a setup time is defined as the time necessary to prepare resource for processing an activity. The most setup types considered in the literature are: sequence-independent setup times, sequence-dependent setup times and schedule dependent setup times. In the first case, setup time depends only on the activity and the resource the activity will use but do not depend on the sequence of activities (Kolisch et al., 1995). In the second case setup times depend not only on the executed activity and the used resource, but also on the sequence of activities processed by this resource. In schedule-dependent setup situations, the setup time depends on the sequence of activities, the required resource as well as other characteristics of the schedule to execute. For example, Mika et al. (2008) considered the problem of scheduling workflow on a grid of computational resources. They formulate the problem as an extension of the multi-mode resource constrained MRCPSP where computers resources are placed in different locations which make setup times schedule-dependent. The objective in such a case is to determine activity mode and execution location as well as a feasible starting time for all activities such that the project duration is minimized. For a comprehensive survey on setup times and their classification in project scheduling see Mika et al. (2006).

This paper makes the following contributions. First, in section 4.2, introduce the Multi-Mode Resource-Constrained Project Scheduling Problem with Sequence dependent Transfer Times

(MRCPSPTT). In section 4.3, we model this problem as a mixed integer program and propose a genetic algorithm to solve the problem in section 4.4. In section 4.5, we present the results of an extensive computational experiment undertaken to assess the performance of several versions of the proposed solution approach. Finally, section 4.6 presents some conclusions and future research.

4.2 Problem definition

In the MRCPSPTT we have a set J of activities, a set P of finish-to-start precedence relations between activities, a set of renewable resource K and a set of non-renewable resources W. The limits on renewable resources are denoted Q_k; $k \in K$ and the availability over the entire project of the non-renewable resources is given by N_w; $w \in W$. Each activity i has a set of immediate predecessors $P_i \subseteq J$, a set of immediate and indirect predecessors, denoted $\pi_i, (\pi_i \subseteq J)$.

In addition, every activity i has a set of possible execution modes M_i. A mode $m \in M_i$ is characterized by q_{imk}, the required number of units of the renewable resource k, $k \in K$, n_{imk}, the required number of units of the non-renewable resource w, $w \in W$, and the corresponding duration d_{im}. We also define I_i as the time window between the earliest start time and the latest start time of activity i. These times are calculated using the critical path method using the shortest mode of each activity and a feasible project duration T.

We assume that activities are to be performed in a number of different locations (sites). The transfer of a renewable resource k from the execution location of activity i to the execution location of activity j requires a transfer time denoted $\triangle_{ijk}$. These transfer times are assumed to fulfil the triangular inequality. Notice that there is no transfer of non-renewable resources between activities as they are consumed when used.

Without loss of generality, we assume that project activities include a dummy start activity, activity 1, and a dummy finish activity, activity F. These two activities have only one execution mode of duration zero, require no non-renewable resources and requires the entire available amount Q_k of the every renewable resource k; $k \in K$ (i.e., q_{11k}=q_{F11k}=Q_k). This is because the dummy start activity should provide renewable resources to the other activities while the dummy finish activity should collect them back. Transfer times from the dummy start activity and to the dummy finish activity are nil i.e., $(\Delta_{1ik} = \Delta_{iFk} = 0; \forall i \in J,\, k \in K)$. The objective in MRCPPTT is to choose an execution mode to each activity and to construct a schedule that minimizes the project duration while satisfying precedence, resource and transfer constraints.

4.3 Mathematical formulation

To formulate the MRCPSPTT we extend the RCPSPTT mixed-integer linear program of Krüger and Scholl (2009). The following decision variables are used:

$f_{imjm'k}$: Number of units of resource k directly transferred from activity i executed using mode m, to activity j executed using mode $m^{'}$.

$y_{imjm'k}$: A binary that takes the value 1 if and only if a unit or more than one unit of renewable resource k are transferred from activity i executed using mode m to activity j executed using mode $m^{'}$.

x_{imt} : A binary that takes the value 1 if and only if activity i is executed using mode m and starts at the beginning of time period t.

The mathematical model is as follows:

$$\text{Miminize} \sum_{t \in I_F} t x_{F1t} \tag{4.1}$$

Subject to :

$$\sum_{t \in I_i} \sum_{m \in M_i} x_{imt} = 1 \qquad \forall i \in J, m \in M_i \tag{4.2}$$

$$\sum_{t \in I_j} \sum_{m' \in M_j} t x_{jm't} - \sum_{t \in I_j} \sum_{m \in M_i} (t + d_i) \cdot x_{imt} \geq 0 \qquad \forall (i,j) \in P \tag{4.3}$$

$$\sum_{t \in I_j} \sum_{m' \in M_j} t x_{jm't} - \sum_{t \in I_i} \sum_{m \in M_j} (t + d_i) x_{imt} - T \sum_{m \in M_i} \sum_{m' \in M_j} y_{imjm'k} \geq -T + \triangle_{ijk} \qquad \forall i \in J - F, j \in J - \pi_i \forall k \in K \tag{4.4}$$

$$2 y_{jm'imk} \leq \sum_{t \in I_j} x_{jm't} + \sum_{t \in I_i} x_{imt} \qquad \forall m \in M_i, \forall m' \in M_j, \qquad \forall i \in J - F, \forall j \in J - \pi_i, \forall k \in K \tag{4.5}$$

$$f_{imjm'k} \leq \min(q_{imk}, q_{jm'k}) \cdot y_{imjm'k}; \qquad \forall i \in J - F, j \in J - \pi_i \tag{4.6}$$

$$\sum_{i \in J - \sigma_i} \sum_{m \in M_i} \sum_{m' \in M_j} f_{imjm'k} = \sum_{t \in I_j} \sum_{m' \in M_j} q_{jm'k} x_{jm't} \qquad \forall j \in J - \pi_i \tag{4.7}$$

$$\sum_{i\in J-\sigma_i}\sum_{i\in J-\sigma_i}\sum_{j\in J-\pi_i} f_{imjmk'} = \sum_{t\in I_i}\sum_{m\in M_i} q_{imk}x_{jmt} \qquad \forall i \in J-F \qquad (4.8)$$

$$\sum_{t\in I_j}\sum_{i\in J}\sum_{m\in M_i} n_{imt} \leq N_w \qquad \forall w \in_W \qquad (4.9)$$

$$x_{imt} \in \{0,1\} \qquad \forall m \in M_i, \forall i \in J, \forall t \in I_i \qquad (4.10)$$

$$y_{imjm'k} \in \{0,1\} \qquad \forall m \in M_i, \forall m' \in M_j, \qquad \forall k \in K, \forall i \in J-F, \forall j \in J-\pi_i \qquad (4.11)$$

$$f_{imjm'k} \geq 0 \qquad \forall m \in M_i, \forall m' \in M_j \qquad \forall k \in K, \forall i \in J-F, \forall j \in J-\pi_i \qquad (4.12)$$

The objective function (4.1) minimizes the project duration. Constraints (4.2) ensure that each activity is performed in only one mode and is started within its time window. Constraints (4.3) are the precedence constraints. Constraints (4.4) ensure that the transfer time of the renewable resource k is taken into consideration when transferred from activity i performed using mode m to activity j performed using mode m'. Constraints (4.5) make sure that $y_{imjm'k}$ take the value 0 if the modes of i and j are not respectively m and m'. Constraints (4.6) imply that if $y_{imjm'k} = 0$ then $f_{imjm'k} = 0$; Otherwise $f_{imjm'k}$ takes a value equal the less between the requirement of j and what is available at i. Constraints (4.7) and (4.8) represent the flow conservation constraint of renewable resources. Constraints (4.9) ensure that the used quantities of non-renewable resources do not exceed the imposed limits. Constraints (4.10), (4.11) and (4.12) define the decision variable of the model.

Consider the project instance given in Figure 4.1 (which will be used throughout the remainder of this paper) with 10 non-dummy activities, each with two modes. For each mode, the requirements of one renewable resources and one nonrenewable resource are indicated. The availability of the renewable resource and the nonrenewable resources are 6 and 15 respectively. We assume that activities are to be performed in a number of different locations (sites). The transfer time of the renewable resource from the execution location of activity i to the execution location of activity j is denoted Δ_{ij} and given by the matrix Δ. We assume that this matrix is symmetric. Figure 4.2 depicts one of the optimal solutions of this example with a makespan of 19 time-periods where shaded areas indicate transfer times of resource units between different activities.

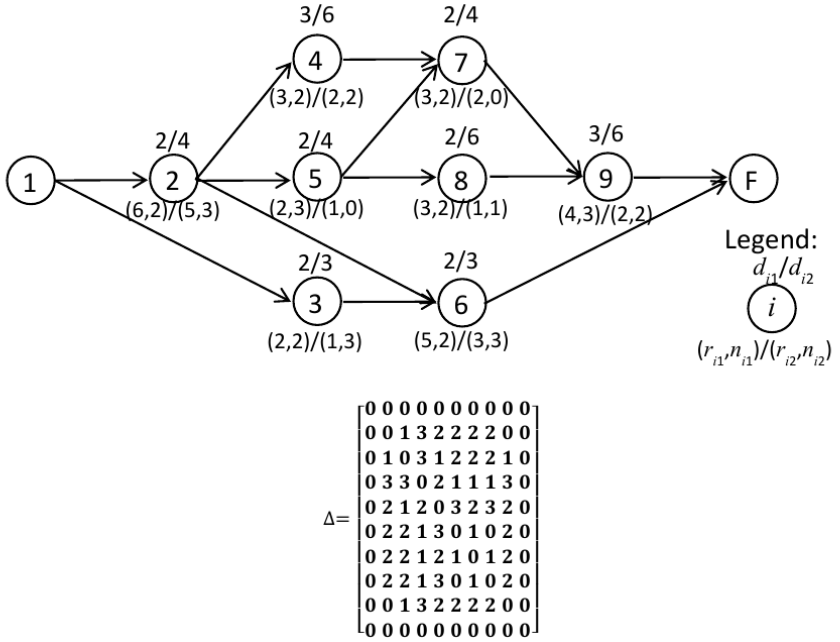


Figure 4.1: a MRCPSPTT example.

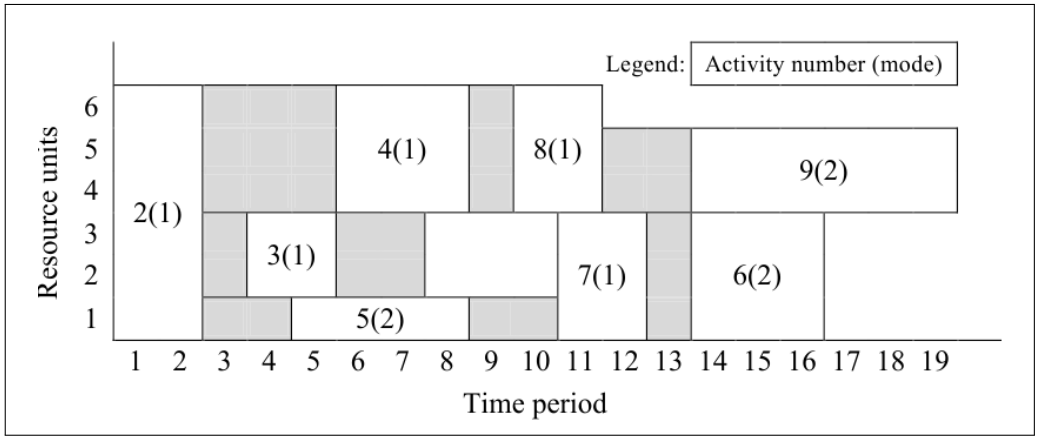


Figure 4.2: One of the optimal solutions of the example.

4.4 The proposed genetic algorithm

The proposed genetic algorithm is given in Figure 4.3. In order to reduce the search space we apply a preprocessing procedure similar to the one used by Sprecher et al. (1997) prior to applying the proposed genetic algorithm. This preprocessing procedure eliminates all inefficient or infeasible modes as well as all abundant (not constraining) non-renewable resources. A mode is inefficient if there is another mode for the same activity having the same or smaller

duration and requires the same amount of resources or less. A mode is infeasible if its usage would violate the resource constraints in any schedule. In addition, a non-renewable resource is said to be abundant (not constraining) if the sum of the maximal requirements for that resource does not exceed its availability. Excluding an abundant non-renewable resources, inefficient and infeasible modes does not eliminate any feasible or optimal schedule.

After applying the preprocessing procedure, the genetic algorithm starts with generating an initial population of G individuals. Then each individual of the population is evaluated and assigned a fitness value. After that, the population is randomly partitioned into pairs of individuals and, to each resulting pair, a crossover operator is applied to produce two new solutions.

Afterwords, a mutation operator is applied to modify some new offspring individuals. Then, among the G parent individuals and G new solutions we choose the best G ones to become the next generation. This process is repeated until a terminating condition is reached. Specifically, the proposed genetic algorithm stops once the total number of generated solutions reaches a predetermined value.

1: Preprocessing
2: Generate an initial population of G
3: $N_{solutions} = G$
4: **WHILE** ($N_{solutions} \leq L$)
- Generate G offspring individuals from current population using the proposed two-position crossover operator proposed hereafter..
- Apply mutation to some of the generated offspring solutions.
- For feasible offspring solutions, apply a local search procedure to improve it.
- Determine the fitness of each offspring solution.
- $N_{solutions} = N_{solutions} + G$
- Add the new G solutions to the previous generation of G solutions.
- Select the best G individuals to be the next generation.

END WHILE
5: Retain the best obtained solution

Figure 4.3: The proposed genetic algorithm.

4.4.1 Solution coding

Each individual is represented by a vector composed of two lists and an extra binary gene (see Figure 4.4). The fist list, named AL, is a precedence-feasible activity list, i.e.; a list where each activity is placed in a position after the position of each of its predecessors. The second is a mode assignment list, named ML, which assigns to each activity in AL one of its possible modes. After that, an extra binary is added. This binary variable takes either the value S or P indicating the schedule generation scheme (serial or parallel) to use in order to generate a

schedule based on the lists AL and ML. To construct this schedule, we use either the serial or the parallel scheduling scheme presented in Kadri and Boctor (2016). The constructed schedule satisfies precedence constraints, renewable resources availability and transfer time constraints. But it may or may not respect the non-renewable resource availability constraints.

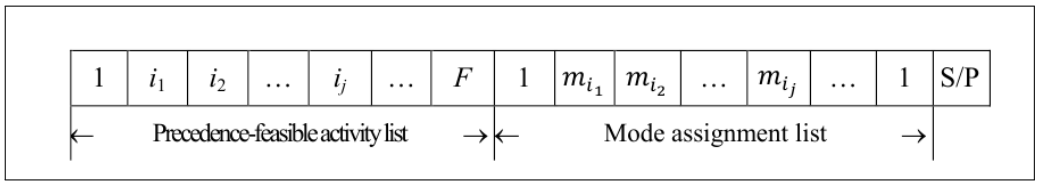


Figure 4.4: Solution coding (representation).

4.4.2 Generation of the initial solutions population

The proposed genetic algorithm starts by generating an initial solutions population. To generate a solution we need to generate its precedence-feasible activity list, its mode list and the binary indicating whether to use the serial or the parallel scheduling procedure. This binary is randomly generated.

Mode lists generation

We start by generating the mode lists for all the individual solutions of the initial population. To generate the mode list of the first individual we use the Minimum Normalized non-renewable Resources (MNR) procedure proposed by Lova et al. (2009). This procedure assigns to each activity i the mode m_i which minimizes $\sum_{w \in W}(n_{imw}/N_{imw})$. However, if using these modes is not feasible; i.e., requires more non-renewable resources than what is available, we repeat the following steps until the mode list becomes feasible.

- For each activity i and each mode $m \in M_i$, calculate $s_{im} = \sum_{w \in W}(n_{im_i w} - n_{imw})$
- Let j and $m_j^{'}$, be the activity and mode that produces the maximum value of s_{im}. Replace the current mode of the activity j in the mode list by $m_j^{'}$.

The mode list of each of the other individuals in the initial population is generated from the mode list of the first individual by randomly selecting half of the activities and randomly modifying their modes. If the resulting mode list is not feasible, apply the two-step procedure given above to bring it to feasibility.

Precedence-feasible activity lists generation

We use a modified version of the regret-based biased sampling heuristic (Kolisch and Drexl

(1997)) to generate the precedence-feasible activity lists of the initial population. This heuristic is an iterative construction heuristic which repeats the following steps until all activities are placed in the list.

- Determine S, the list of eligible activities, i.e., activities whose predecessors are already in the list.
- For each activity $i \in S$, calculate its selection probability $\Psi_j = r_j / \sum_{i \in S} r_i$, where r_i, called the regret value of i, measures the priority given to this activity. In this implementation of the heuristic, priority is a function of one of the following activity characteristics: SLK (total slack), LFT (latest finish time) and LST (latest start time) of the activity.
- Use the simple roulette technique to select one of the eligible activities based on their selection probability Ψ_j. Add the selected activity to the list and go back to step 1.

4.4.3 The mating pool

All the individuals (solutions) of a given generation are used to generate the individuals of the next one and each individual participate only once in this process. First we arrange the individuals of the current generation in the ascending order of their fitness and we divide the arranged list in two halves. Then, among the individuals who haven't yet participated in the generation process, we randomly select and crossover one individual from the top half of the list with another one from the bottom half. We use the crossover operator presented in the next section and generate two new offspring solutions from the selected pair of solutions.

4.4.4 The proposed two-position crossover operator

To generate an offspring solution from two parent solutions we modify the two-position crossover operator proposed in Kadri and Boctor (2016) to solve the single mode case of the RCPSPTT with transfer times. First, the activity lists of the parent solution codes are crossed as follows. A contiguous block of genes is selected from the first parent (called the donor) by randomly selecting two numbers n_1 and n_2 ($1 < n_1 < n_2 < |J|$) which indicate the bounding positions in the donor activity list of the required contiguous block. Then, we determine the position of the first and last elements of this block in the activity list of the second parent (called the receiver). Let these positions be denoted by n_0 and n_3 respectively. The offspring activity list is built by concatenating:

- activities in positions 1 to $n_0 - 1$ from the activity list of the receiver;

- activities in between positions n_0 and n_3 in the receiver activity list which are predecessors to any activity in the block;
- activities of the block;
- activities in positions between n_0 and n_3 in the receiver sequence who are successors to any activity in the block; and,
- activities in positions between $n_3 + 1$ and $|J|$ in the receiver sequence.

However, some of the activities in the positions between n_0 and n_3 in the receiver activity list may neither be a predecessor or a successor to any activity in the block. These activities, called free activities, do not yet figure on the offspring activity list. For every one of these free activities, we determine all possible positions in the offspring activity list where it can be placed without violating precedence relations. Then we randomly put it in one of these positions.

The mode assignment list of the offspring solution is constructed as follows. The modes to assign to the activities of the contiguous block are their modes in the donor parent while the modes of the remaining activities are copied from the receiver parent. Finally, the additional binary (indicating if a serial or parallel scheme is to be used) is copied from the code of the donor solution. Notice that from every pair of parent solutions we generate two offspring solutions. To generate the first offspring, we use the first parent solution as donor and the second as receiver while to generate the second offspring we consider the second parent as donor and the first one as receiver.
To illustrate the application of the proposed two-point crossover operator described above, let us use the numerical example presented in Figure 4.1. Two feasible solution codes are presented in top of the Figure 4.5 and are considered as the donor and receiver parents respectively. Assuming that we randomly draw n_1=3 and n_2=5 then the contiguous block of activities to transfer from the donor parent to the offspring is composed respectively from activities 4, 3 and 6 (activities in positions 3, 4 and 5 in the donor activity list). Consequently, the positions n_0 and n_3 in the receiver parent are respectively position 2 (where we have activity 3) and 7 (where we have activity 4). Now we can start building the activity list and the mode assignment list for the offspring as follows. We copy:

- from the receiver all activities between positions 1 and $n_0 - 1$ (only activity 1 and)
- from the receiver all activities in positions n_0 to n_3 that are predecessors to any activity in the block (only activity 2).
- from the donor all activities of the block (activities 4, 3 and 6)

– from the receiver all activities in positions n_0 to n_3 that are successors to any activity in the block (none).

– from the receiver all activities from positions $n_3 + 1$ to the last position (activities 7, 9 and F).

Two activities (activities 5 and 8) remain free. First we determine the possible positions of activity 5 and randomly place it in one of these positions. Then we do the same for activity 8. Afterwards, we copy the mode of the activities of the block from the donor and the mode of the remaining activities from the receiver. Finally we copy the last binary from the donor. To generate the second offspring solution, we consider the second parent as donor and the first one as receiver and we repeat the same steps.

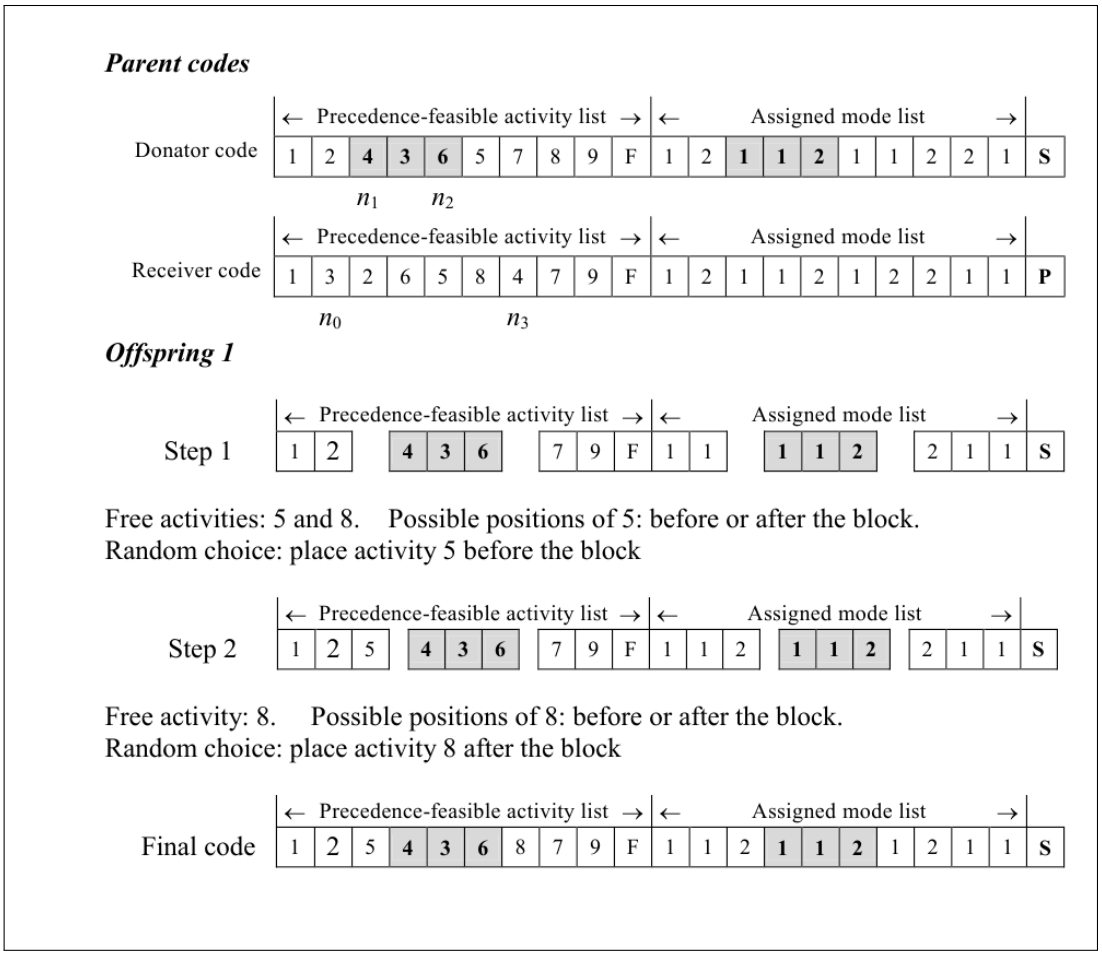


Figure 4.5: Application of the proposed two-position crossover operator.

4.4.5 Fitness function

The fitness of a feasible solution is measure by the project duration as given by this solution. For infeasible solutions the fitness should be larger than the corresponding duration and may contain a penalty reflecting the extent of infeasibility. Notice that all the generated activity lists are precedence-feasible and that the used scheduling heuristics (serial and parallel) always produce solutions that respect the availability of renewable resources. Only the availability of non-renewable resources can be exceeded due to the crossover of the mode lists of the parent solutions. Thus we need to penalize such infeasibilities. Before presenting how we measure the fitness of solutions (I) let us introduce some notation that will be used for this presentation. Let:

T : be an upper bound on the project duration equals the sum of the maximal duration of all project activities,

E : be the sum, over all non-renewable resource types, of the number of required units exceeding the resource availability; thus : $E = \sum_{w \in W} max(0, \sum_{i \in J} n_{im_i w} - N_w)$, where m_i the mode assigned to activity i in the considered solution,

$E^{'}$: be the normalized sum, over all non-renewable resource types, of the number of required units exceeding the resource availability; thus : $E^{'} = \sum_{w \in W} max(0, (\sum_{i \in J} n_{im_i w} - N_w)/N_w)$, where m_i the mode assigned to activity i in the considered solution,

C : be the critical path length of the project when using the minimal duration of activities,

D : be the duration of the project corresponding to the considered solution (I),

D_{max} : be the maximum, over the population of solutions, of the corresponding project durations.

Hartmann (2001) suggest measuring the fitness of infeasible solutions by :

$$f(I) = T + E \tag{4.13}$$

While Alcaraz et al. (2003) suggest measuring it by :

$$f(I) = \{ \begin{array}{ll} D, & \text{if} \quad E = 0 \\ D + D_{max} + E - C, & \text{otherwise} \end{array} \tag{4.14}$$

Another suggestions is made by Lova et al. (2009) :

$$f(I) = \{ \begin{array}{ll} 1 - (D_{max} - D)/D, & \text{if} \quad E = 0 \\ 1 + (D - C)/D + E^{'}, & \text{otherwise} \end{array} \tag{4.15}$$

We tested all these suggestions within our genetic algorithm.

4.4.6 Mutation operator

The following mutation process is applied to every individual (solution) of each generation. First we draw three random numbers, x_1, x_2 and x_3, from a uniform distribution between 0 and 1. If x_1 is less than the preselected mutation probability, denoted μ, we randomly select a position i in the activity list of the considered solution. Now if the activity in position i is not a predecessor of the activity in position $i+1$, we permute these two activities. Similarly, if $x_2 < \mu$, we randomly select an activity j and we randomly change its assigned mode unless j has only one possible mode. Finally, if $x_3 < \mu$, we flip the last binary of the solution code. In our implementation of this genetic algorithm, we used a mutation probability $\mu = 0.05$.

4.4.7 Schedule Improvement by local search

We apply a local search procedure only to feasible schedules with respect to non-renewable resources in order to improve their make-span. The local improvement heuristic is an iterative one where we search the unit-neighborhood of the current mode list by modifying the execution mode of only one activity at a time, provided that this modification does not lead to requiring an amount of any non-renewable resource that exceeds its availability. The procedure stops when we reach the local optimum of this neighborhood.

4.5 Computational results

4.5.1 Test instances

To create our test instances we added transfer times to the standard J10, J20 and J30 sets of the MRCPSP generated by PROGEN (Kolisch et al., 1995) with two independent problem parameters: resource factor (RF), and resource strength (RS). RF reflects the average portion of resources requested per activity-mode combination. RS express the relationship between the resource demand of the activities and the resource availability. The sets J10, J20 and J30 are composed of 536, 554 and 552 projects with 10, 20 and 30 non-dummy activities. Each of the non-dummy activities may be performed in one out of three modes that can require units of two renewable and two non-renewable resources. The duration of a mode varies between 1 and 10 periods of time. Transfer times are randomly sampled from a uniform distribution between 0 and 10 periods. Table 4.1 gives the number of instances in each of the instance subsets. All calculations were run on a computer with i7 core processor, 2 GHz and 6 GB of internal memory.

RF	0.5	0.5	0.5	0.5	1	1	1	1	
RS	[0.2 , 0.25]	0.5	[0.7 , 0.75]	1	[0.2 , 0.25]	0.5	[0.7 , 0.75]	1	All
$J10$	51	70	68	70	68	69	70	70	536
$J20$	69	70	72	60	71	71	71	70	554
$J30$	70	70	70	60	70	70	71	71	552
All	190	210	210	190	209	210	212	211	1642

Table 4.1: Number of Instances in different instances subsets

4.5.2 Optimal solutions

We tried to solve all the test instances to optimality by using the commercial MIP code GUROBI version 5.0.1 to solve model (4.1) to (4.12) presented in section 4.3. It became clear that some of the test instances require very large computational time and consequently we decided to limit computational time to 30000 seconds. Table 4.2 and Table 4.3 indicate the number of instances that was solved to optimality and the execution time obtained, recpectively.
From this table we can see that 40% of all test instances were solved to optimality within the imposed time limit.

In more details, 67% of the J10 instances were solved to optimality while only 30,5% and 22,5% optimal solution were found for the instances of sets J20 and J30 respectively. However, although we have not obtained the optimal solution for the remaining 60% of instances, we got at least one feasible solution for each of them within the imposed run time limit.
Moreover, Table 4.2 shows that the percentage of problems solved to optimality decreases with the decrease of RS (resource strength) and with the increase of RF (resource Factor). No instance with RS= 0.2 and RF= 1.0 was solved to optimality within the run time limit of 30000 seconds while this percentage is 99.5% for instances with RS=1.0 and RF=0.5.

RF	0.5	0.5	0.5	0.5	1	1	1	1	
RS	[0.2 , 0.25]	0.5	[0.7 , 0.75]	1	[0.2 , 0.25]	0.5	[0.7 , 0.75]	1	All
$J10$	47	70	68	70	0	7	31	65	358
$J20$	1	35	68	59	0	0	0	6	169
$J30$	0	8	54	60	0	0	0	2	124
All	48	113	190	189	0	7	31	73	651

Table 4.2: Number of instances where the optimal solution is found

Set	Minimum computation time (sec)	Average computation time (sec)	Maximum computation time (sec)
$J10$	0.53	309.56	1164.34
$J20$	4.49	1021.66	17598.78
$J30$	14.90	1083.41	29028.19

Table 4.3: Computation time required to obtain the optimal solution

4.5.3 Genetic solutions to MRCPSPTT

The results of computational tests instances J10, J20 and J30 with transfer times are given in the Table 4.4, Table 4.5 and Table 4.6 respectvelly. In these tables are presented, the percentage deviation from the optimal solution (%Dev), the number of instance for which the optimal solutions was obtained (Optiml) and the corresponding average exectution time (CPU in sec) for the three stopping criterion (2000, 4000 and 10000 for J10 and 4000, 6000 and 20000 for J20 and J30 instances) using the three fitness function tested in this research. From these tables we can see that the algorithm with version using Lova's fitness function gives better results than those obtained with those Hartmann's and Alcaraz functions. The best obtained average deviation from the optimum is 0.57%, 1.36% and 1.74% for the J10, J20 and J30.

It is shown also that the best version of the proposed genetic algorithm obtained the optimal solution for 321, 125, 83 problems out of 358, 169 and 124 instances solved. Thus the genetic algorithm found the optimum for 89.66% of 10 activity instances, 73.96% of the 20 activity instances and 66.94% of the 30 activity instances.

4.6 Conclusion

This paper study the Multi-Mode Resource Constrained Project Scheduling Problem with sequence dependent Transfer Times; a rich extension of the MRCPSP. It provides a mathematical formulation of the problem as well as a genetic algorithm to solve it. To the best of our knowledge, the literature does not yet provided neither a formulation nor a solution procedure to solve this problem. Results obtained by the proposed genetic algorithm indicate that it is possible to obtain good solutions within a reasonable amount of computational time. Further research is obviously needed to develop more efficient optimal solution procedures as well as better heuristics. The development of an optimal solution procedure capable of solving all the used test instances will allow us to provide a more complete assessment of the performance of the proposed heuristic as well as other heuristics that will be developed in the future.

Fitness function	Number of visited solutions								
	2000			4000			6000		
	Dev(%)	Optiml	CPU(sec)	Dev(%)	Optiml	CPU(sec)	Dev(%)	Optiml	CPU(sec)
Hartmann fitness function	1.40	288	21.01	0.74	312	39.25	0.61	317	57.46
Alcaraz fitness function	1.43	280	20.96	0.86	308	39.24	0.63	315	57.42
Lova fitness function	1.38	284	20.52	0.70	316	39.27	0.57	321	57.41

Table 4.4: Solution for J10 after 2000, 4000 and 6000 solutions

Fitness function	Number of visited solutions								
	6000			10000			20000		
	Dev(%)	Optiml	CPU(sec)	Dev(%)	Optiml	CPU(sec)	Dev(%)	Optiml	CPU(sec)
Hartmann fitness function	4.30	67	63.71	2.23	97	147.33	1.78	110	285.80
Alcaraz fitness function	4.30	57	63.72	2.05	96	147.56	1.51	113	285.70
Lova fitness function	4.48	59	64.58	1.96	102	150.74	1.36	125	293.0

Table 4.5: Solution for J20 after 4000, 10000 and 20000 solutions

Fitness function	Number of visited solutions								
	6000			10000			20000		
	Dev(%)	Optiml	CPU(sec)	Dev(%)	Optiml	CPU(sec)	Dev(%)	Optiml	CPU(sec)
Hartmann fitness function	9.70	35	104.10	2.84	66	226.81	1.78	81	431.50
Alcaraz fitness function	9.86	34	104.13	2.76	62	234	1.78	78	451.30
Lova fitness function	9.70	33	102.58	2.87	69	235.37	1.74	83	450.90

Table 4.6: Solution for J30 after 4000, 10000 and 20000 solutions

4.7 References

Alcaraz, J., M, C., R, R., 2003. Solving the multi-mode resource-constrained project scheduling problem with genetic algorithms. Journal of the Operational Research Society 54 (6), 614–626.

Baar, T., Brucker, P., S, K., 1999. Tabu search algorithm and lower bounds for the resource-constrained project scheduling problem. In: Voss et al. (eds) Meta heuristics : Advances and trends in local search paradigms for optimization. Springer, pp. 1–19.

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

Boctor, F. F., 1993. Heuristics for scheduling projects with resource restrictions and several resource-duration modes. International Journal of Production Research 31 (11), 2547–2558.

Boctor, F. F., 1996a. A new and efficient heuristic for scheduling projects with resource restrictions and multiple execution modes. European Journal of Operational Research 90 (2), 349–361.

Boctor, F. F., 1996b. Resource-constrained project scheduling by simulated annealing. International Journal of Production Research 34 (8), 2335–2351.

Bouleimen, K., Lecocq, H., 2003. A new efficient simulated annealing algorithm for the resource-constrained project scheduling problem and its multiple mode version. European Journal of Operational Research 149 (2), 268–281.

Chiang, C.-W., Huang, Y.-Q., Wang, W.-Y., 2008. Ant colony optimization with parameter adaptation for multi-mode resource-constrained project scheduling. Journal of Intelligent & Fuzzy Systems 19 (4, 5), 345–358.

Drexl, A., Gruenewald, J., 1993. Nonpreemptive multi-mode resource-constrained project scheduling. IIE transactions 25 (5), 74–81.

Elmaghraby, S. E., 1995. Activity nets: A guided tour through some recent developments. European Journal of Operational Research 82 (3), 383–408.

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

Hartmann, S., Drexl, A., 1998. Project scheduling with multiple modes: a comparison of exact algorithms. Networks 32 (4), 283–297.

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

Józefowska, J., Mika, M., Różycki, R., Waligóra, G., Węglarz, J., 2001. Simulated annealing for multi-mode resource-constrained project scheduling. Annals of Operations Research 102 (1-4), 137–155.

Kadri, R. L., Boctor, F. F., 2016. An efficient genetic algorithm to solve the resource-constrained project scheduling problem with sequence dependent transfer times : The single mode case. In: Proceedings of the 15th International Conference on Project Management and Scheduling. pp. 116 – 119.

Kolisch, R., Drexl, A., 1997. Local search for nonpreemptive multi-mode resource-constrained project scheduling. IIE transactions 29 (11), 987–999.

Kolisch, R., Sprecher, A., Drexl, A., 1995. Characterization and generation of a general class of resource-constrained project scheduling problems. Management science 41 (10), 1693–1703.

Krüger, D., 2009. Multi-project scheduling with transfers. Ph.D. thesis, University of Jena, Germany.

Krüger, D., 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.

Lova, A., Tormos, P., Cervantes, M., Barber, F., 2009. An efficient hybrid genetic algorithm for scheduling projects with resource constraints and multiple execution modes. International Journal of Production Economics 117 (2), 302–316.

Maniezzo, V., Mingozzi, A., 1999. The project scheduling problem with irregular starting time costs. Operations Research Letters 25 (4), 175–182.

Merkle, D., Middendorf, M., Schmeck, H., 2002. Ant colony optimization for resource-constrained project scheduling. Evolutionary Computation, IEEE Transactions on 6 (4), 333–346.

Messelis, T., De Causmaecker, P., 2014. An automatic algorithm selection approach for the multi-mode resource-constrained project scheduling problem. European Journal of Operational Research 233 (3), 511–528.

Mika, M., Waligóra, G., Weglarz, J., 2006. Modelling setup times in project scheduling. In: Perspectives in modern project scheduling. Springer, pp. 131–163.

Mika, M., Waligora, G., Węglarz, 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.

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

Mori, M., Tseng, C. C., 1997. A genetic algorithm for multi-mode resource constrained project scheduling problem. European Journal of Operational Research 100 (1), 134–141.

Nonobe, K., Ibaraki, T., 2002. Formulation and tabu search algorithm for the resource constrained project scheduling problem. In: Essays and surveys in metaheuristics. Springer, pp. 557–588.

Özdamar, L., 1999. A genetic algorithm approach to a general category project scheduling problem. Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on 29 (1), 44–59.

Patterson, J. H., Brian Talbot, F., Slowinski, R., Węglarz, J., 1990. Computational experience with a backtracking algorithm for solving a general class of precedence and resource-constrained scheduling problems. European Journal of Operational Research 49 (1), 68–79.

Poppenborg, J., Knust, S., 2014. A flow-based tabu search algorithm for the rcpsp with transfer times. In: Proceedings of the 14th International Conference on Project Management and Scheduling, March 30th– April 2nd 2014. pp. 181 – 184.

Pritsker, A. A. B., Waiters, L. J., Wolfe, P. M., 1969. Multiproject scheduling with limited resources: A zero-one programming approach. Management Science 16 (1), 93–108.

Słowinski, R., 1981. Multiobjective network scheduling with efficient use of renewable and nonrenewable resources. European Journal of Operational Research 7 (3), 265–273.

Słowiński, R., Soniewicki, B., Węglarz, J., 1994. Dss for multiobjective project scheduling. European Journal of Operational Research 79 (2), 220–229.

Sprecher, A., Hartmann, S., Drexl, A., 1997. An exact algorithm for project scheduling with multiple modes. Operations-Research-Spektrum 19 (3), 195–203.

Talbot, F. B., 1982. Resource-constrained project scheduling with time-resource tradeoffs: The nonpreemptive case. Management Science 28 (10), 1197–1210.

Thomas, P. R., Salhi, S., 1998. A tabu search approach for the resource constrained project scheduling problem. Journal of Heuristics 4 (2), 123–139.

Van Peteghem, V., Vanhoucke, M., 2010. A genetic algorithm for the preemptive and non-preemptive multi-mode resource-constrained project scheduling problem. European Journal of Operational Research 201 (2), 409–418.

Węglarz, J., Józefowska, J., Mika, M., Waligóra, G., 2011. Project scheduling with finite or infinite number of activity processing modes–a survey. European Journal of Operational Research 208 (3), 177–205.

Zamani, R., 2013. A competitive magnet-based genetic algorithm for solving the resource-constrained project scheduling problem. European Journal of Operational Research 229 (2), 552 – 559.

Zapata, J. C., Hodge, B. M., Reklaitis, G. V., 2008. The multimode resource constrained multiproject scheduling problem: Alternative formulations. AIChE Journal 54 (8), 2101–2119.

Zhang, H., Tam, C., Li, H., 2006. Multimode project scheduling based on particle swarm optimization. Computer-Aided Civil and Infrastructure Engineering 21 (2), 93–103.

Zhu, G., Bard, J. F., Yu, G., 2006. A branch-and-cut procedure for the multimode resource-constrained project-scheduling problem. INFORMS Journal on Computing 18 (3), 377–390.

Chapitre 5

Problème général d'allocation et de nivellement de ressources pour l'ordonnencement de projet

Dans la littérature certains chercheurs ont étudié le problème d'ordonnancement de projet à ressources limitées où les tâches ont plusieurs modes d'exécution (POPRL/PME). D'autres ont traité le problème en considérant l'objectif de minimiser la variation d'utilisation des ressources durant toute l'exécution du projet. Dans la pratique, nous devons déterminer simultanément la quantité de ressources à allouer au projet au cours de son exécution et réduire la variabilité de l'utilisation des ressources au minimum tout en essayant de terminer le projet à une date de fin acceptable. Les quantités des ressources à allouer au projet devraient permettre l'accomplissement du projet à cette date et devient une limite sur la disponibilité de ces ressources durant toute l'exécution du projet. Dans ce chapitre, nous considérons ce problème plus réaliste d'ordonnancement de projet où, étant donné une limite supérieure sur les ressources qui peuvent être mises à la disposition du projet et une limite supérieure à la date de fin du projet, nous devons déterminer les quantités des ressources à allouer au projet, la date d'échéance à proposer au client tout en minimisant le coût total de réalisation du projet. Le coût d'exécution est composé de deux principaux éléments : le coût direct des ressources à utiliser et le coût des frais généraux qui ne dépend pas de la quantité de ressources allouées, mais qui est proportionnel à la durée du projet. Au meilleur de notre connaissance, ce problème, que nous appelons Problème général d'allocation et de nivellement des ressources d'un projet (PGANRP), n'a jamais fait l'objet d'une publication auparavant.

Article 3 : The Generalized Resource Allocation and Leveling Problem in project scheduling

Abstract

Many research publications dealt with the multi-mode project scheduling problem under the constraint of limited resource availability. Some others treated the problem where it is required to keep the resource utilization level as invariable as possible all over the project execution period. In real-life situations we need to simultaneously determine the amount of resources to allocate to the project during its execution and to reduce the resources utilisation variability to the minimum while trying to finish the project by an acceptable completion date. The amount of resources to allocate to the project should allow finishing the project by this date and becomes a limit on the availability of these resources at any time period. In this paper we consider this more realistic project scheduling problem where, given an upper limit on the resources that can be made available to the project and an upper limit on the project completion date, we have to determine the amount of resources to allocate to its execution, the due date to propose to the client while minimizing the project execution cost. The execution cost is composed of two main elements: the direct cost of resources to use and the overhead cost which does not depend on the amount of allocated resources but is directly proportional to the project duration. To the best of our knowledge this problem, which we call the *Generalized Resource Allocation and Levelling Problem* (GRALP), has never been the subject of any publication before. The paper proposes a mathematical formulation of the problem, a 3-Dimension neighbourhood search heuristic, and a genetic algorithm to solve it. It also presents a numerical experiment to assess the performance of the proposed heuristics.

5.1 Introduction

Project scheduling issues have attracted a lot of attention since the pioneering work of Kelley (1963) and Weist (1963). One of the most studied problems is the resource-constrained project scheduling problem (RCPSP) where the objective is to minimize the project duration while respecting the precedence relations between tasks and the limit on resources availability.

Among the first contributions to this problem is the work by Brand et al. (1964); Wiest (1967); Pritsker et al. (1969); Fisher (1970); Davis (1973) and Patterson and Huber (1974).

Another important problem is called the resource leveling problem (RLP) where we may or may not have limits on resource availabilities, may or may not have a project due date and we need to schedule the execution of the project tasks while minimizing the fluctuation of the amount of resources to use. Among the first contributions in this area are the paper by Burgess and Killebrew (1962) and the one by De Witte (1964).

In practice the project scheduling problem is somewhat different. First we do not have a fixed limit on the amount of resources. Rather we need to determine the amount of resources to allocate to the project and the allocated amount becomes the limit to respect. Also, we usually need to determine and to negotiate with the client a date to finish the project. So we need to determine what is the best date or range of dates to propose to the client. This date becomes then the project due date. Finally, we need to build a schedule with the least possible fluctuations in resource usage.

This paper deals with this more realistic project scheduling problem where given an upper limit on the resources that can be made available to the project and an upper limit on the project completion date; we have to determine the amount of resources to allocate to its execution and the due date to propose to the client while minimizing the project execution cost. The execution cost is composed of two main elements : the direct cost of the resources allocated to the project and the overhead cost, which does not depend on the amount of allocated resources but is directly proportional to the project duration. This problem is called hereafter the *Generalized Resource Allocation and Leveling problem* (GRALP).

In the GRALP we have a project composed of a set of tasks, a set of zero-lag, finish-to-start precedence constraints defined over the set of tasks and a set of renewable resources. There is a limit on the amount of renewable resources that can be made available to the execution of the project. However we need to determine the exact amount that should be allocated to the project. Every activity has a number of possible execution modes where each mode is characterized by a required number of units of each renewable resource and the corresponding duration.

We assume that the direct cost of making resources available to the project is linearly proportional to the allocated amount of resources and to its duration. In addition, there is an overhead cost which is linearly proportional to the duration of the project. This duration is to be determined and should be within a range of acceptable dates. So we also need to determine the date to finish the project within this range.

Thus, the objective is to determine the amount of resources to allocate to the project, to select

for each task an execution mode and to assign to each task its execution dates in order to minimize the overall project execution cost while completing it at what is considered as an acceptable completion date. To the best of our knowledge the GRALP has never been studied in the open literature until now. Thus we do not dispose of any method to solve it.

The remainder of the paper is organized as follows. Section 5.2 presents a review of most recent related literature. Section 5.3 presents a mathematical formulation of the problem and section 5.4 presents a numerical example to emphasize the differences between the GRALP and what is called the resource availability cost problem (RACP). Section 5.5 and 5.6 present the proposed approaches to solve the GRALP. Section 5.7 presents the numerical experience undertaken to assess the performance of these heuristics and the conclusions of this research work are given in section 5.8.

5.2 Review of recent related literature

The purpose of this section is not to review the huge body of literature dealing with the resource-constrained project scheduling or with resource leveling in project scheduling but to review the most related research to the topic of this paper. Many publications review available contributions to the RCPSP. Herroelen et al. (1998) provide a survey of the most important contributions made in the preceding 5 years. Kolisch (2000) present a survey not only of models and solution methods for the deterministic RCPSP but also survey contributions to decision systems building in this area. Herroelen and Leus (2005) review fundamental approaches to project scheduling under uncertainty. They review reactive scheduling, stochastic project scheduling and fuzzy project scheduling approaches. Finally, Hartmann and Briskorn (2010) give an overview of several extensions of the basic RCPSP that generalize some activity execution conditions (for example by integrating setup times or allowing for pre-emption), generalize the precedence and temporal relations (by introducing parallel execution for example) or generalize the resource constraints (by considering partially renewable resources or dedicated resources). Other, but older, surveys are given by Herroelen (1972), Davis (1973), Icmeli et al. (1993), Demeulemeester (1995), Elmaghraby (1995) and Herroelen et al. (1997).

To the best of our knowledge, the GRALP has never been the subject of any publication. However, some researchers studied a problem that can be seen as related to the GRALP to some extent. This problem is called the Resource Availability Cost Problem (RACP) by some researchers, and the Resource Investment Problem (RIP) by some others. In this problem the objective is to determine the amounts of resources to allocate to the project in order to minimize the availability cost of these resources. However, to solve the problem, researchers make a very questionable assumption. They assume that the cost of making a unit of a resource available from the beginning to the end of the project execution is the same whatever

the duration of the project. This assumption considerably simplifies modeling the problem and reduces the complexity of the solution methods. But it is not a real-life assumption. In real life, the availability cost over the project duration is proportional to this duration.

Also we notice that, in dealing with either the RACP or RIP, the overhead cost is not taken into account. Again, in real-life situations, the overhead cost represents a significant part of the project execution cost and should not be neglected. Actually, as observed by Neumann and Zimmermann (2000), the RACP is a special case of the resource leveling problem where the objective is to minimize a weighted function of the maximum number of resource units to use.

Möhring (1984) was the first to introduce the RACP and considered the case where tasks have only one execution mode. He proposed an optimal solution procedure based on extending the partial order defined by the precedence relations in a way that respects the limit on completion time. Demeulemeester (1995) also proposed an optimal solution procedure to solve the single-mode version of the problem and called it the Resource Availability Cost Problem (RACP). Drexl and Kimms (2001) developed two lower bounds for the problem using Lagrangean relaxation and column generation techniques. The problem they considered and renamed as the Resource Investment Problem (RIP) is slightly different from the RACP as they did not consider any limit on the project completion date.

Heuristic approaches are also proposed to solve the RACP. Yamashita et al. (2006) proposed a scatter search heuristic and Shadrokh and Kianfar (2007) proposed a genetic algorithm. All these solution methods are designed to solve the single-mode version of the problem. The only heuristic designed to solve the multi-mode version is the one proposed by Hsu and Kim (2005). They proposed a serial schedule generation method based on two new priority dispatching rules.

Obviously the GRALP studied in this paper is different from the RACP and the RIP. In the GRALP we minimize the total project cost including the resources usage cost and the overhead cost. Both costs are proportional to the project duration. The differences will be discussed in more details in section 5.4 using a numerical example.

It is important to explicitly consider overhead cost as neglecting it may lead to solutions with larger project duration especially in the multi-mode case where longer modes usually require less resource amounts and consequently using such modes reduces the overall resource requirements and the corresponding resource availability cost. That is why we should take into account both overhead costs and resource availability costs.

5.3 Mathematical formulation

Given the set of project tasks to execute, their possible execution modes and precedence relations, the upper bound on the amount of resources to allocate to the project, an upper limit on the project completion time, the availability cost of resources, and the project overhead cost, our problem is to determine for each task its starting date and its execution mode as well as the amount of resources to allocate to the project that minimizes the total execution cost while satisfying the precedence relations, not using more resources than what is allocated to the project, and completing the project by the required deadline.

Recall that each task has one or more execution modes and each possible mode is characterized by its duration and the amount of resources needed by this mode. The total cost is composed of the resource availability cost and the overhead cost over the whole execution duration. Resource availability cost is the product of the amount of resources allocated to the project multiplied by its unit cost and by the project duration.

Before presenting the proposed mathematical formulation let us first present the notation used to build this model.

Indices:

i task index; $i = 1, \ldots, N$

j execution mode index; $j = 1, \ldots, m_i$ (m_i is the number of possible modes for every task i)

k resource type index; $k = 1, \ldots, K$ (K is the number of resource types)

t time period index; $t = 1, \ldots, H$ (H is upper limit on project completion time)

Sets and Parameters

f_t overhead expenses for time period t

d_{ij} duration of task i if execution mode j is used

E_i earliest finish time of task i (as given by the critical path method using the shortest execution mode of every task$i$)

T project earliest finish time (as given by the critical path method)

L_i latest finish time of task i (as given by the critical path method while using the shortest modes and H as a limit on project completion time)

P_i set of immediate predecessors of task i

S_i set of immediate successors of task i

r_{ijk} number of resource units of type k required to execute task i using its execution mode j

c_{kt} availability cost of a unit of resource type k at period t

M_k maximum number of units of resource type k that can be made available to the project

A_t set of tasks that could be scheduled to be in execution at t; i.e.,$A_t = \{i \backslash E_i - d_{i1} < t \leq L_i\}$

Decision variables:

R_k number of units of resource type k to allocate to the project over its execution time

x_{ijt} binary variable that takes the value 1 if and only if task i is executed using mode j and finished at the end of period t

y_t binary variable that takes the value 1 if the project is in execution at period t

R_{kt} variable that takes the value R_k if the project is still in execution at period t and takes the value zero otherwise

It is obvious that y_t should take the value 1 for all periods $t \leq T$. So the model seeks to determine the values y_t of only for $t = T + 1$,...,H. Similarly, the variable x_{ijt} should take the value zero for all periods $t < E_i + d_{ij} - d_{i1}$ or $t > L_i$. So the model seeks to determine the value of x_{ijt} for periods t such that $E_i + d_{ij} - d_{i1} \geq t \leq L_i$

Using the above listed notation, the GRALP can be formulated as follows:

$$Find:$$

$$x_{ijt}; \quad i = 1, \ldots, N, j = 1, \ldots, m_i, t = E_i + d_{ij} - d_{i1}, \ldots, L_i$$

$$R_k; \quad k = 1, ..., K$$

$$R_{kt}; \qquad k = 1, \ldots, K, t = T + 1, \ldots, H$$

$$y_t \in \{0, 1\}; \qquad t = T + 1, \ldots, H$$

$$Minimize: \sum_{t=1}^{T} f_t + \sum_{t=T+1}^{H} f_t y_t + \sum_{k=1}^{K} \sum_{t=1}^{T} c_{kt} R_k + \sum_{k=1}^{K} \sum_{t=T+1}^{H} c_{kt} R_{kt} \tag{5.1}$$

Subject to :

$$\sum_{j=1}^{m_i} \sum_{t=E_i+d_{ij}-d_{i1}}^{L_i} x_{ijt} = 1 \quad ; \qquad i = 1, \ldots, N \tag{5.2}$$

$$\sum_{j=1}^{m_e} \sum_{t=E_e+d_{ej}-d_{e1}}^{L_e} t x_{ejt} - \sum_{j=1}^{m_i} \sum_{t=E_i+d_{ij}-d_{i1}}^{L_i} (t - d_{ij}) x_{ijt} \leq 0 \quad ; \qquad \forall e \in P_i, i = 1, \ldots, N \tag{5.3}$$

$$\sum_{i \in A_t} \sum_{j=1}^{m_i} \sum_{s=max\{t, E_i+d_{ij}-d_{i1}\}}^{min\{t+d_{ij}-1, L_i\}} r_{ijk} x_{ijs} \leqslant R_k \quad ; \qquad k = 1, ..., K, t = 1, \ldots, H \tag{5.4}$$

$$R_k \leq M_k \quad ; \qquad k = 1, \ldots, K \tag{5.5}$$

$$R_{kt} \geq R_k - M_k(1 - y_t) \quad ; \quad k = 1, \ldots, K, t = T+1, ..., H \tag{5.6}$$

$$N y_t \geqslant \sum_{i \in A_t} \sum_{j=1}^{m_i} \sum_{s=max\{t, E_i + d_{ij} - d_{i1}\}}^{min\{t+d_{ij}-1, L_i\}} x_{ijs} \quad ; \quad t = T+1, ..., H \tag{5.7}$$

$$y_t \leqslant y_{t-1} \quad ; \quad t = T+2, \ldots, H \tag{5.8}$$

The objective function 5.1 expresses the total cost of the project. The first and third terms express respectively the overhead and resources availability costs of the T first periods while the second and forth terms give the overhead and resources costs for the periods T+1 up to the and of the project. Constraints 5.2 determine the execution mode and the end period for each task. It also makes sure that each task has one end only one execution mode and also one and only one finish period. Constraints 5.3 assure that any task cannot starts before the end of the execution of its immediate predecessors and constraints 5.4 determine the amount of resources to be allocated to the execution of the project. Constraints 5.5 ensure that the allocated amounts of resources do not exceed what can be made available and the constraints 5.6 make sure that no resources are allocated once the project is completed. Constraints 5.7 ensure that y_t equals one (the project is in execution) if there is any task to execute at period t and constraints 5.8 implies that if the project is not in course of execution at period t-1, it is not in period t.

5.4 Differences between the GRALP and the RACP : A numerical example

The objective of this section is to underline the differences between the GRALP and the RACP as well as to show the effect of not considering the overhead cost explicitly in the scheduling process. To do so, let us first present the mathematical formulation of the RACP under the assumption of multi-mode tasks and let c_k be cost of using a unit of resource k over the project life. As mentioned before, in the RACP it is assumed that this cost is known, constant and does not depend on the duration of the project. Thus, the RACP can be formulated as follows:

$$Find : x_{ijt} \quad ; \forall i = 1, \ldots, N, j = 1, \ldots, m_i, t = E_i + d_{ij} - d_{i1}, \ldots, L_i$$

$$Minimize : \sum_{k=1}^{K} c_k R_k \tag{5.9}$$

Subject to :

$$\sum_{j=1}^{m_i} \sum_{t=E_i+d_{ij}-d_{i1}}^{L_i} x_{ijt} = 1 \quad ; \qquad i = 1, \ldots, N \tag{5.10}$$

$$\sum_{j=1}^{m_e} \sum_{t=E_e+d_{ej}-d_{e1}}^{L_e} t x_{ejt} - \sum_{j=1}^{m_i} \sum_{t=E_i+d_{ij}-d_{i1}}^{L_i} (t - d_{ij}) x_{ijt} \leq 0 \quad ; \quad e \in P_i, i = 1, \ldots, N \tag{5.11}$$

$$\sum_{i \in A_t} \sum_{j=1}^{m_i} \sum_{s=max\{t, E_i+d_{ij}-d_{i1}\}}^{min\{t+d_{ij}-1, L_i\}} r_{ijk}.x_{ijs} \leqslant R_k \quad ; \qquad k \in K, t = 1, \ldots, H \tag{5.12}$$

$$\sum_{t=E_i+d_{ij}-d_{i1}}^{L_i} \sum_{j=1}^{m_i} t x_{ijt} \leq H \quad ; \qquad i = 1, \ldots, N \tag{5.13}$$

Let us now consider the 5-task, 2-resource numerical example shown in Figure 5.1 with an overhead cost of 20 per time unit, resource usage cost of 2 and 5 per time unit respectively for resources 1 and 2, and a limit H on the project duration of 16 time units. The optimal solution of the corresponding GRALP is to allocate 3 and 6 units of resources 1 and 2 to the project which will lead to project duration of 10 time units. The resource usage cost for these 10 time units is 360 while the overhead cost is 200 for an optimal total cost of 560 . This optimal solution is exhibited in Figure 5.2.

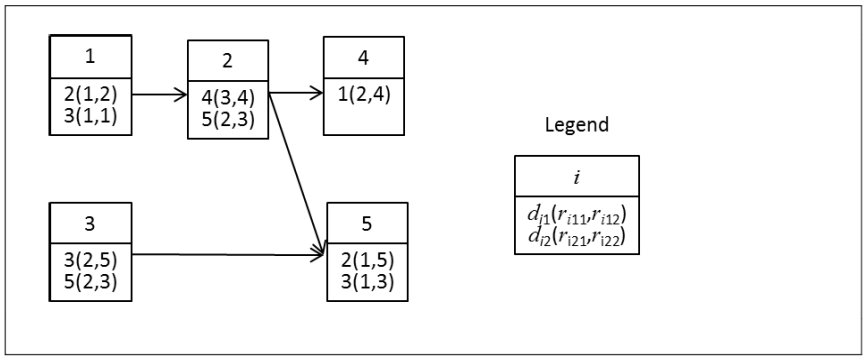


Figure 5.1: the 5-task, 2-resource numerical example

On the other hand, if we handle the problem as a RACP using any multiple of 2 and 5 as availability cost (say 2p and 5p), the optimal duration of the project will be 16 time units and the number of resource units allocated to project will be 2 and 4 respectively. The corresponding availability cost will be 24p. The optimal solution of this RACP is shown in Figure 5.3.

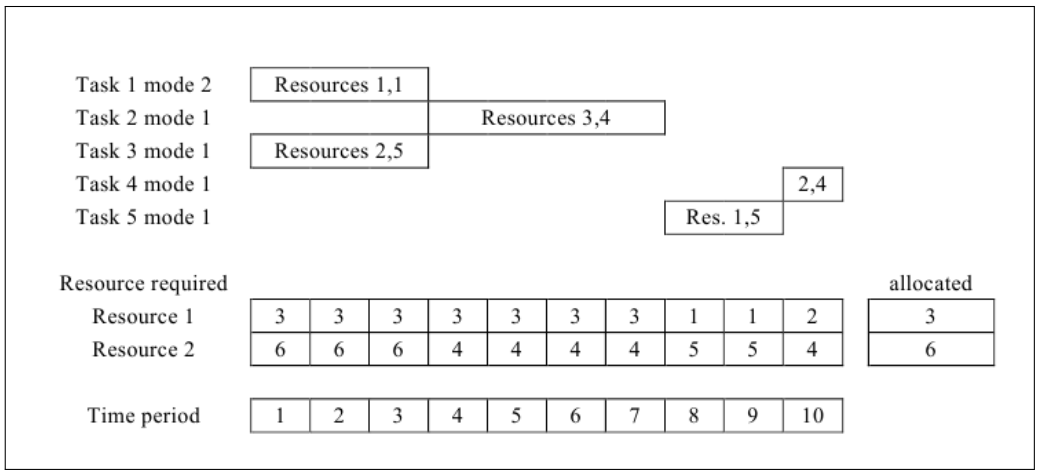


Figure 5.2: Optimal solution of the GRALP

Clearly, as in the GRALP we consider that resource usage cost is proportional to the project duration and as we explicitly consider the overhead cost, it is not optimal to prolong the execution of the project behind a certain date. On the other hand, the optimal solution of the RACP has a duration which is either equal to the given due date or the duration that minimizes the cost of resource units allocated to the project.

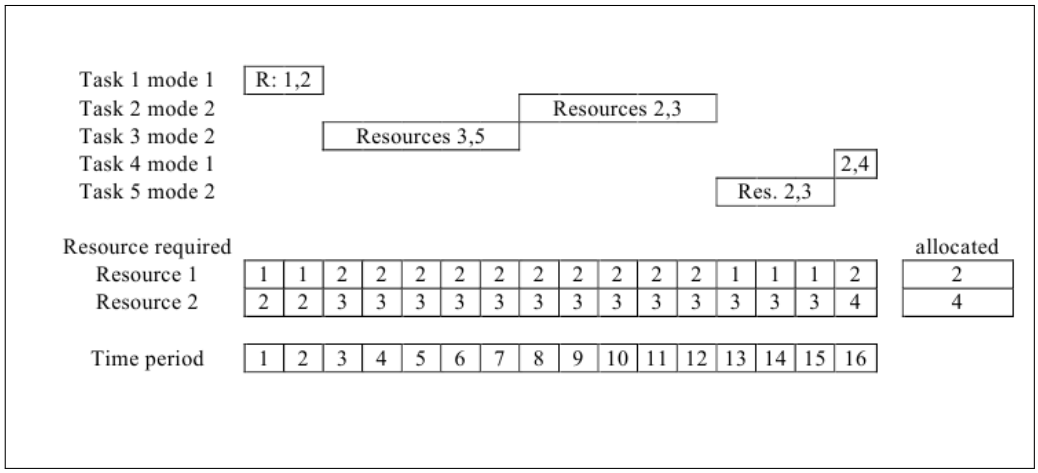


Figure 5.3: Optimal solution of the RACP

Now, to emphasise the importance of explicitly considering the overhead cost, let us set this cost equal to zero in our GRALP. In this case the optimal duration for our numerical example increases to 13 time units and we have to allocate 3 and 4 units of resources 1 and 2 respectively

to the project. The resulting resource usage cost is 338. Obviously, as the overhead cost is neglected, the optimal duration is increased but not as much as if we assume that the availability cost does not depend on the project duration. Actually, if we extend the project duration behind 13 time units, we may require less resource units but for longer period of time resulting a higher resource usage cost.

5.5 A 3-Dimension neighbourhood search heuristic

The Generalized Resource Allocation and Leveling Problem (GRALP) is a complex one that requires making several interrelated decisions. Mainly three decisions should be made. We need : (1) to determine the number of resource units to allocate to the project, (2) to determine the execution mode for each activity or task, and (3) to construct an execution schedule; i.e. to determine the execution dates for the activities. Obviously these decisions are interrelated. For example the optimal execution modes and dates largely depend on the amount of resources allocated to the project and the optimal dates depend on the selected execution modes.

The heuristic approach we propose in this section is a 3-dimension neighbourhood search approach where each neighborhood is related to one of these three decisions. The first dimension or decision is coded by a vector of K elements where the k^{th} element gives the number of units R_k of resource type k to allocate to the project. The second dimension or decision is coded by a vector of N elements where the i^{th} element indicates the mode j_i to be used to execute activity or task i. Finally, the execution schedule is coded by a vector $(\sigma_1, \sigma_2, ..., \sigma_N)$ of N elements indicating the sequence in which activities should be considered for scheduling (a priority vector). This sequence can be translated into a schedule by assigning to each activity (while using the selected execution mode) the earliest possible execution date which satisfies resource limits and precedence relations.

The general framework of the proposed 3-D neighborhood search approach is depicted in Algorithm 1. The details of the approach and its main procedures (called *MMRCPS*, *LNS_ Mode*, *LEVEL* and *LNS_ Sequence*) are explained in the remainder of this section.

5.5.1 Generation of allocated resources vectors

This 3-D search procedure generates a number of vectors of allocated resources R_k; k = 1,...,K. It starts with the vector $R_k = M_k$; k = 1,...,K and then searches its neighborhood. Actually all the vectors with $M_k \geq R_k \geq \lambda_k$; k=1,...,K, except those leading to a project duration that exceeds the duration limit H, will be examined. Here λ_k stands for the smallest amount of resource type k that should be allocate to the project. Clearly, this smallest amount

λ_k is such that:

$$\lambda_k \geq max_i\{min_j r_{ij_k}\} \tag{5.14}$$

As if $\lambda_k < max_i\{min_j r_{ij_k}\}$ if we cannot schedule some of the activities. Also q_k should be such that:

$$\lambda_k \geq (1/H).\sum_{i=1}^{N} min_j\{d_{ij}.r_{ij_k}\} \tag{5.15}$$

As otherwise we cannot construct a schedule with duration that does not exceed the duration target H. Thus λ_k can be calculated by:

$$\lambda_k = max[max_i\{min_j r_{ij_k}\}, (1/H).\sum_{i=1}^{N} min_j\{d_{ij}.r_{ij_k}\}] \tag{5.16}$$

Notice that in the procedure presented by Algorithm 1, at each iteration we reduce R_k, the allocated amount of one of the resources. However, for a resource type k, we stop reducing the value of R_k as soon as the current value produces a schedule with a duration larger than H. In such a case we move to reduce the amount of the next resource type.

5.5.2 *MMRCPS*: the schedule construction heuristic

For each vector of allocated resources R_k; k=1,..., K, the heuristic generates an initial mode vector and a schedule by applying a parallel priority-based scheduling heuristic that we call MMRCPS (multi-mode resource constrained project scheduling) heuristic. The main steps of this heuristic are given in Algorithm 2. The scheduling rule we use is the MINSLK (minimum total slack) rule. However, as the total slack of a task depends on its execution mode, the total slack is calculated for each feasible mode of the task while setting the modes of the other tasks to the shortest one. Several researchers indicated that this rule applied within a parallel scheduling generation heuristic produces the best results (for example see Boctor (1993)).
Following the application of the MMRCPS heuristic, we obtain a mode-vector indicating the used execution mode for each task and a schedule. The schedule can then be coded by the sequence of tasks we selected in the course of this procedure.

Algorithm 1 General framework of the proposed 3-D neighborhood search approach

Set $R_k = M_k; k = 1, ..., K$
Calculate λ_k, k = 1,...,K, ! The smallest amount of resource type k to allocated to the project
Set $k = 0$
WHILE $k < K$
  $k = k + 1$
    WHILE $R_k > \lambda_k$
      R_k =R_k -1
      IF $k > 1$ **THEN**
        FOR $r = 1$ **TO** k:
          R_k =M_k
        END FOR
      END IF
      Calculate earliest and latest dates as well as total slack
      Call *MMRCPS* ! Parallel scheduling scheme using MINSLK rules.
      Call *LNS_ Mode* ! Local search of the mode vector neighborhood.
      Call *LEVEL* ! Solves the resulting Leveling problem.
      IF obtained duration < H **THEN**
        IF obtained cost < Best cost **THEN** store as the best solution found
        Call *LNS_ Sequence* ! Local search on the sequence vector neighbourhood
      ELSE:
        $R_k = \lambda_k$! Duration larger than H is obtained. Move to reduce the amount of the next resource type.
      END IF
    END WHILE
END WHILE

5.5.3 Searching the neighborhood of a mode-vector

Without loss of generality, let us assume that, for every task, its modes are numbered in the ascending order of their duration; i.e. $d_{ij} \leq d_{ij+1}; \forall i, \forall j < m_i$.

The next step of the proposed solution procedure is to perform a local neighborhood search to improve the mode vector. Three versions of this neighborhood search improvement procedure, called *LNS_ Mode*, are tested in this research work. The difference between these versions resides in their neighborhood definition. The steps of these 3 versions are given in Algorithm 3, Algorithm 4 and Algorithm 5. The fastest one is the first one where we examine the neighborhood where only one task to which we assigned its first mode is modified to become the second mode; provided that the task has more than one possible mode. In the second version we examine the neighborhood where only the mode of one task is incremented to the next mode number if it exists. In the third version the modes of two different tasks are modified where the mode of the first task is incremented to the next mode number and the mode of the second task is decremented.

Algorithm 2 Steps of the priority-based parallel scheduling heuristic MMRCPSP

WHILE some non-scheduled tasks remain :

1) Go to the following and nearest time period where there are some available resources;
2) Establish the list of tasks that are schedulable at this time period (i.e., those for which all immediate predecessors are already scheduled and there are sufficient resources to perform them using at at least one of their execution modes);
3) If the list is empty go back to step 1;
4) For each schedulable task determine the list of feasible modes (i.e., modes such that available resources are larger than or equal to their resource requirements);
5) For each combination task-mode determine its total slack;
6) Schedule the combination task-mode having the smallest total slack;
7) Reduce resource availabilities consequently and update the list of schedulable task at the considered time period;
8) Go back to step 3.

END WHILE

Algorithm 3 Steps of the first tested version of *LNS_Mode*

```
Set improvement = TRUE
WHILE improvement = TRUE
  improvement = FALSE
    FOR i = 1 To N :
      IF ( mode of (i = 1 AND number of modes of i > 1) THEN
        mode of i = 2
        Call MMRCPSP
        IF resulting cost < Best cost THEN :
          Save as the best solution found
          Improvement = TRUE
        Else:
          (mode of i) = 1
        END IF:
      END IF
    END FOR
END WHILE
```

Algorithm 4 Steps of the second tested version of *LNS_Mode*

```
Set improvement = TRUE
WHILE improvement = TRUE
  improvement = FALSE
    FOR i = 1 TO N :
      IF mode of i < number of modes of i THEN
        mode of i = 1 + mode of i
        Call MMRCPSP
        IF resulting cost < Best cost THEN :
           Save as the best solution found
           Improvement = TRUE
         ELSE:
           mode of i = (mode of i) - 1
         END IF:
      END IF
    END FOR
END WHILE
```

Algorithm 5 Steps of the third tested version of *LNS_Mode*

```
Set improvement = TRUE
WHILE improvement = TRUE
  improvement = FALSE
    FOR i = 1 TO N :
      FOR l = 1 TO N :
       IF (mode of i > 1 AND mode of l < number of mode of l AND i≠ l)
         (mode of i) = (mode de i) - 1
         (mode of l) = (mode de l) + 1
         Call MMRCPSP
         IF resulting cost < Best cost THEN :
            Save as the best solution found
            Improvement = TRUE
          ELSE:
            (mode of i) = (mode de i) + 1
            (mode of l) = (mode de i) - 1
          END IF:
       END IF
      END FOR
    END FOR
END WHILE
```

5.5.4 Further reduction of total cost by resource leveling

The solution obtained at the end of applying *LNS_ Mode* can sometimes be improved if we can further reduce the cost of the allocated resources. This can be done by solving a corresponding RLP (Resource Leveling Problem). This problem can be stated as follows. Given the set of tasks to perform, the selected execution mode of each task, the precedence relations between these tasks, their execution dates, and the resulting project duration, modify the execution dates in order to minimize the sum of maximum resource requirements weighted by the unit cost of each resource without increasing the project duration.

Within our GRALP solution approach, this RLP is solved by the heuristic sketched in Algorithm 6. It consists in moving tasks having a positive free slack to start at the position that may lead to the maximum possible cost reduction if such position exists.

Algorithm 6 *LEVEL* : The used resource leveling heuristic

```
Calculate the free slack for each task
FOR i = N TO 1
  IF free slack of i > 0 THEN
    Calculate the reduction in resource cost for each possible starting date of task i
      (search a date between current starting date and this date plus its free slack)
    Determine the position which corresponds to the maximum cost reduction
    IF maximum cost reduction > 0 THEN :
      Set the starting date of i to the date leading to this maximum cost reduction
      Update the free slack of tasks 1,2,..., i-1
    END IF
  END IF
END FOR
```

5.5.5 Searching the neighborhood of the sequence vector

The next step in the proposed solution heuristic is to do a local search in the neighborhood of the task sequencing vector used to code the solution schedule. The proposed search procedure, called *LNS_ Sequence*, is presented in Algorithm 7. It consists in randomly modifying the sequence vector a number of times, each time construct a solution using the MMRCPS heuristic, improve the obtained solution by applying *LNS_ Mode* and RLP, and calculate the cost of the resulting solution. The best obtained solution is then retained as the final solution of our GRALP.

Algorithm 7 *LNS_Sequence* : the sequence neighborhood search procedure

FOR $i = 1$ **TO** N
  Determine l_i, the position of the last predecessor of task i in the sequence
  Determine u_i, the position of the first successor of task i in the sequence
  Randomly chose a position p between $l_i + 1$ and $u_i - 1$ and move task i to position p
  Call *MMRCPS*
  Call *LNS_Mode*
  Call *LEVEL*
  IF resulting cost < Best Cost **THEN** :
    Save as the best solution found
  ElSE
    Return task i to its original position
  END IF
END FOR

5.6 Genetic algorithm to the GRALP

This section presents the genetic algorithm used to solve the generalized resource allocation and leveling problem (GRALP). The proposed algorithm starts by generating an initial population of G individuals. Then each individual of the population is evaluated to determine its fitness value. After that, the population is randomly partitioned into $G/2$ pairs of individuals and a crossover operator is applied to each resulting pair of individuals in order to produce two new offspring solutions. Afterwards, a mutation operator is applied to modify some of the offspring solutions. Then, among the G parent individuals and G new solutions, select the best G ones to become the next generation. The proposed genetic algorithm stops once the total number of generated solutions reaches a preselected number L. Algorithm 8 exhibits the main steps of the algorithm.

5.6.1 Solution coding

As shown in section 5.5, each individual is represented by a vector composed of three parts (vectors) and $2N + K$ elements (see Figure 5.4). The first part, composed of N elements, is a precedence-feasible activity list. The second part, also composed of N elements, indicate the mode number to use for each of the N activities. The third part, composed of K elements, the number of units of each of the K resource types to allocate to the project. Notice that, within this genetic algorithm we use a serial scheduling scheme to transform the precedence-feasible activity list into a schedule.

Algorithm 8 Steps of the proposed Genetic Algorithm.

Use the procedure given in section 5.6.2 generate an initial population of G individuals
Set number of generations = 1
WHILE(number of generations $\leq$ L)
Randomly partition the G individuals into $G/2$ pairs
From each pair generate two offspring solutions using the two-position crossover operator presented in section 5.6.4
Apply the mutation procedure presented in section 5.6.6
Determine the fitness of each offspring solution.
Select the best G individuals among those of the new generation and previous one to become the next generation
Set L=L+1
END WHILE
Retain the best obtained solution

5.6.2 Initial population

The proposed genetic algorithm starts by generation an initial population of solutions. To generate a solution we need to generate a precedence-feasible activity list, to assign a mode number to each activity, and to choose the amount of resources to allocate to the project.

Mode list generation
For each activity, execution modes are randomly selected from the list of possible modes of the activity. This is done for every solution of the initial population.

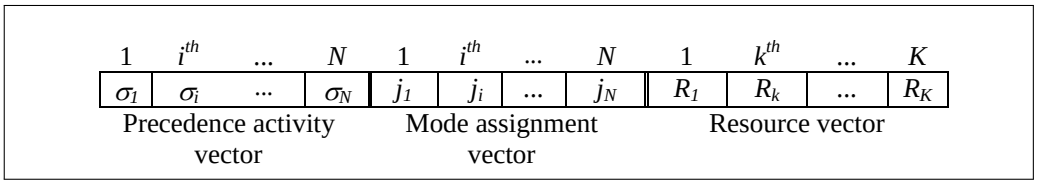


Figure 5.4: Solution coding (representation).

Precedence-feasible activity lists generation
We use a modified version of the regret-based biased sampling heuristic (Kolisch and Drexl (1997)) to generate the precedence-feasible activity lists of the initial population. This heuristic is an iterative construction heuristic which repeats the following steps until all activities are placed in the list.

- Determine S, the list of eligible activities, i.e., activities whose predecessors are already in the list. Arrange the activities of S in descending order of their latest finish time (LFT). Let r_i be the rank of activity i in arranged list;

- For each activity $i \in S$, calculate its selection probability $\Psi_j = r_j / \sum_{i \in S} r_i$
- Use the simple roulette technique to select one of the eligible activities based on their selection probability Ψ_j. Add the selected activity to the list and go back to step 1.

Allocated resources

For each solution of the initial population, allocated resources are randomly sampled from a uniform distribution between q_k, the smallest amount to allocate in order to always obtain a feasibe solution whatever the selected execution modes, and $M_k, k = 1, ..., K$, the maximum number of units of resource type k that can be made available to the project. he following equation gives the value of q_k :

$$q_k = max[max_i\{max_j r_{ik}\}, (1/H).\sum_{i=1}^{N} min_j\{d_{ij}.r_{ijk}\}] \tag{5.17}$$

And M_k the maximum number of units of resource type k that can be made available to the project.

5.6.3 Mating pool

All the individuals (solutions) of a given generation are used to generate the individuals of the next one and each individual participate only once in this process. First, the G individuals of the current generation are randomly divided into $G/2$ pairs of individuals. Then, the crossover operator presented in the next section is used to generate two new offspring solutions from each pair of parent solutions.

5.6.4 Crossover

To generate an offspring solution from two parent solutions we extend the two-position crossover operator proposed in Kadri and Boctor (2016) to solve the single mode case of the RCPSP with transfer times. This is done in three steps. First we crossover the precedence-feasible activity lists of the parent solutions, second, we select the mode to assign to each activity, and, finally we try four combinations of the allocated resources vector of the parents and retain the one producing the best solution.

Precedence-feasible activity lists crossover

First, the activity lists of the parent solution codes are crossed as follows. A contiguous block of genes is selected from the first parent (called the donor) by randomly selecting two numbers n_1 and n_2 $(1 < n_1 < n_2 < |N|)$ which indicate the bounding positions in the donor activity list of the required contiguous block. Then, we determine the position of the first and last elements of this block in the activity list of the second parent (called the receiver). Let these

positions be denoted by n_0 and n_3 respectively. The offspring activity list is then built by concatenating:

- activities in positions 1 to n_0-1 from the activity list of the receiver;
- activities in between positions n_0 and n_3 in the receiver activity list which are predecessors to any activity in the block;
- activities of the block;
- activities in positions between n_0 and n_3 in the receiver sequence who are successors to any activity in the block; and,
- activities in positions between n_3+1 and $|N|$ in the receiver sequence.

However, some of the activities in the positions between n_0 and n_3 in the receiver activity list may neither be a predecessor or a successor to any activity in the block. These activities, called free activities, do not yet figure on the offspring activity list. For every one of these free activities, we determine all possible positions in the offspring activity list where it can be placed without violating precedence relations. Then we randomly put it in one of these positions.

Mode selection
The mode assignment list of the offspring solution is constructed as follows. The modes to assign to the activities of the contiguous block are their modes in the donor parent while the modes of the remaining activities are copied from the receiver parent.

Selecting the best combination of the parents resource vectors
We randomly chose a position p and try the following four allocated-resource vector: the allocated resource vector of the donor, the resource vector of the receiver, the vector resulting from combining the first p elements of the donor vector with the last K-p elements of the receiver vector, and, the one obtained by combining the first p elements of the receiver vector with the last K-p elements of donor vector. Thus we try four allocated-resources vectors and retain the one producing the best solution.

Notice that from every pair of parent solutions we generate two offspring solutions. To generate the first offspring, we use the first parent solution as donor and the second as receiver while to generate the second offspring we consider the second parent as donor and the first one as receiver.

5.6.5 Fitness computation

The fitness of a solution is measured by the project total cost which is the sum of the direct cost of resource utilisation and the overhead cost.

5.6.6 Mutation

To apply the mutation operator we need to preselect a value for the mutation probability parameter, denoted μ ($0 \leq \mu \leq 1$). In our implementation of this genetic algorithm, we used a mutation probability $\mu = 0.05$. Then, the following mutation process is applied to every individual (solution) of the generated offspring solution. First we draw three random numbers, x_1, x_2 and x_3, from a uniform distribution between 0 and 1. If x_1 is less than the preselected mutation probability μ, we randomly select a position i in the activity list of the considered solution. Now, if the activity in position i is not a predecessor of the activity in position $i+1$, we permute these two activities. Similarly, if $x_2 < \mu$, we randomly select an activity j and we randomly change its assigned mode unless j has only one possible mode. Finally, if $x_3 < \mu$, we randomly choose a resource type k and randomly modify the allocated amount of this resource type.

5.7 Performance evaluation

Ninety test instances were generated to evaluate the performance of the proposed heuristics. In the following we present these test instances and the obtained results.

5.7.1 Test instances

Three sets of 30 test instances each are randomly generated and used to evaluate the performance of the proposed heuristic. Each of these instances is composed of 30 tasks and requires two resource types. We limited ourselves to instances of this size as it was very time consuming and often impossible to obtain the optimal solution of larger problems.

The cost of one resource unit is fixed for all instances at 2 and 5 for the two resource types respectively. All parameters values are drawn from a uniform distribution and Table 5.1 indicates the range of these different parameters. The differences between the 3 used instance sets reside in the value given to the overhead cost and the upper limit on the number of resource units that can be made available to the project. For the first instance set, M_k, the upper limit on the amount of resource that can be allocated to the project, is set to equal the number of units required to schedule all tasks at their earliest possible dates while using the shortest modes. This implies that the CPM schedule is feasible with respect to the resource

limits but not necessarily optimal. Also for the first set, the overhead cost per time period is set to be the same for every time period and equals 40 % of the direct cost of the allocated resources.

The overhead cost for second instance set is the same as for the first one but we use different values of M_k. To fix the values of M_k, we solved the instances of the first set to optimality using the MIP code GUROBI 5.0.1 and determined for each instance the optimal value of R_k. Let these optimal values be noted R_k^*. The values of M_k for the second instance set were then set equal to R_k^*-1 if the R_k^* is between 10 and 15, equal to R_k^*-2 if R_k^* is between 16 and 20, and equal to R_k^*-3 if $R_k^* > 20$.

The third instance set is similar to the second one except that the used overhead cost is reduced by one third and truncated to the nearest integer.

Parameter	notation	Distribution range
Number of modes	m_i	1 - 3
Number of predecessors	$\|P_i\|$	0 - 5
Task duration	d_{ij}	1 - 10
Resource requirements	r_{ijk}	0 - 9

Table 5.1: Parameters used to generate test instances

Table 5.2 gives a summary of the computation times required to obtain the optimal solution of the instances of each set. This table shows that solving some instances (of this small size) may require a large computation time. This largely justifies the use of heuristic solution methods. We also notice that reducing both the overhead cost and the limit on the amount of resources that can be allocated to the project, may lead to a significant increase in computation time.

Set	Minimum computation time (sec)	Average computation time (sec)	Maximum computation time (sec)
Set 1	202.15	3270.95	20655.70
Set 2	410.88	3685.35	11443.47
Set 3	690.56	7290.26	54510.35

Table 5.2: Computation time required to obtain the optimal solution

5.7.2 Tested heuristic variants

The results obtained by 5 variants of the proposed 3-D search heuristic , referred to as H1, H2 up to H5, as well as our GA will be presented. The first three variants H1, H2 and H3 are reduced variants of the framework presented in Algorithm 2 where the neighborhood search procedure called LNS_sequence is eliminated and not used. This was done in order to reduce the required computation time. On the other hand, the LNS_sequence procedure is used within H4 and H5. Heuristic variants H1 and H4 use the first version of the neighborhood search procedure LNS_Mode presented in Algorithm 3 while variants H2 and H5 use the version exhibited in Algorithm 4 and variant H3 uses the version given in Algorithm 3.

Tables 5.3, 5.4 and 5.5 give the percentage increase above the optimum for the solutions obtained by the five tested variants of the proposed 3-D neighbourhood search procedure and by the genetic algorithm. From these tables we can see that the genetic algorithm outperforms the 5 variants of the proposed search heuristic for all the three sets tested. The genetic algorithm gives the best results followed by H5, H4, H3, H2 and H1. The best average percentage increase above the optimum is 4.56%, 3.44% and 3.96% for set1, set2 and set3 respectively. The results seem to indicate that the deviation from the optimum solution increases when the amounts of resources that can be made available to the project are increased. We can also see that the average percentage increase above the optimum is inversely proportional to the computation time used by the algorithm. Although the average deviation from the optimum obtained by the genetic algorithm and the five tested variants of the heuristic may seem relatively high, we have to notice that the considered problem is quite difficult to solve and involves 3 sets of decisions.

5.8 Conclusion

In this paper we introduced the Generalized Resource Allocation and Leveling Problem (GRALP), a new project scheduling problem that is closer to real-life project scheduling problems. In this problem we need to determine: (1) the number of units of resources to allocate to the project, (2) the execution mode to use for each task, and (3) the execution dates for these tasks. Our constraints are the precedence constraints, the maximum amount of resources that can be allocated to the project and an upper limit on the project duration. Our objective is to minimize the sum of the resource usage cost and the overhead cost. To the best of our knowledge, the GRALP have never been the subject of any publication and consequently we do not have any previously published method to solve it. The research done in this research work seems to indicate that we are facing a difficult problem as it was not possible to design a heuristic capable to produce an average percentage deviation from the optimum of better than 3.98%.

	H1	H2	H3	H4	H5	GA
Average % deviation from the optimum	8.48	8.19	8.04	6.90	6.63	4.56
Minimum % deviation from the optimum	4.37	4.37	4.37	2.96	2.96	1.00
Maximum % deviation from the optimum	13.04	12.71	12.88	10.62	11.30	8.12
Average computation time (sec.)	34	46	342	1070	1364	107.25

Table 5.3: Summary of the results obtained by the proposed heuristic for instance Set 1

	H1	H2	H3	H4	H5	GA
Average % deviation from the optimum	13.29	13.02	12.47	10.92	10.01	3.44
Minimum % deviation from the optimum	5.76	4.76	5.54	4.02	1.81	0.00
Maximum % deviation from the optimum	23.09	23.09	21.70	19.57	19.57	8.70
Average computation time (sec.)	3	4	37	91	135	120.01

Table 5.4: Summary of the results obtained by the proposed heuristic for instance Set 2

	H1	H2	H3	H4	H5	GA
Average % deviation from the optimum	13.49	13.21	12.65	11.07	10.09	3.96
Minimum % deviation from the optimum	6.18	6.16	5.89	3.82	2.80	0.79
Maximum % deviation from the optimum	22.86	22.86	21.24	18.68	19.27	8.7
Average computation time (sec.)	3	4	36	94	138	120.10

Table 5.5: Summary of the results obtained by the proposed heuristic for instance Set 3

Obviously we need continue our research effort to solve this problem. It is also obvious that we need to find an approach to solve larger problems to optimality. Available MIP codes cannot allow us to solve problems with more than 30 tasks. To solve larger problem we may try to add some valid cuts to our model. We also need to develop a special branch-and-bound heuristic and more efficient heuristics.

5.9 References

Boctor, F. F., 1993. Heuristics for scheduling projects with resource restrictions and several resource-duration modes. International Journal of Production Research 31 (11), 2547–2558.

Brand, J., Meyer, W., Shaffer, L., 1964. The resource scheduling problem in construction. Civil engineering studies report no. 5, Depatement of Civil Engineering, University of Illinois, Urbana, IL.

Burgess, A., Killebrew, J., 1962. variation in activity level on a cyclic arrow diagram. Journal of Industrial Engineering 13, 76–83.

Davis, E. W., 1973. Project scheduling under resource constraints—historical review and categorization of procedures. A I I E Transactions 5 (4), 297–313.

De Witte, L., 1964. Manpower leveling of pert networks. Data processiong for science and Engineering 2, 29–83.

Demeulemeester, E., 1995. Minimizing resource availability costs in time-limited project networks. Management Science 41 (10), pp. 1590–1598.

Drexl, A., Kimms, A., 2001. Optimization guided lower and upper bounds for the resource investment problem. Journal of the Operational Research Society, 340–351.

Elmaghraby, S. E., 1995. Activity nets: A guided tour through some recent developments. European Journal of Operational Research 82 (3), 383–408.

Fisher, M., 1970. Optimal solution of resource constrained network scheduling problem. Technical report no. 56, Operation research center,Massachusetts Institute of Technology.

Hartmann, S., 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.

Herroelen, W., 1972. Resource constrained project scheduling the state of the art. Operational Research Quarterly 23, 168–173.

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

Herroelen, W., Leus, R., 2005. Project scheduling under uncertainty: Survey and research potentials. European Journal of Operational Research 165 (2), 289 – 306.

Herroelen, W. S., Dommelen, P. V., Demeulemeester, E. L., 1997. Project network models with discounted cash flows a guided tour through recent developments. European Journal of Operational Research 100 (1), 97 – 121.

Hsu, C., Kim, D. S., 2005. A new heuristic for the multi-mode resource investment problem. Journal of the Operational Research Society 56 (4), pp. 406–413.

Icmeli, O., Erenguc, S. S., Zappe, C. J., 1993. Project scheduling problems: A survey. International Journal of Operations & Production Management 13 (11), 80–91.

Kadri, R. L., Boctor, F. F., 2016. An efficient genetic algorithm to solve the resource-constrained project scheduling problem with sequence dependent transfer times : The single mode case. In: Proceedings of the 15th International Conference on Project Management and Scheduling. pp. 116 – 119.

Kelley, J., 1963. The critical-path method : Resources planning and scheduling. In: Muth, J. F., Thomson, G. L. E. (Eds.), Industrial Planning Scheduling. Prentice-Hall, New Jersey, pp. 347 – 365.

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

Kolisch, R., Drexl, A., 1997. Local search for nonpreemptive multi-mode resource-constrained project scheduling. IIE transactions 29 (11), 987–999.

Möhring, R. H., 1984. Minimizing costs of resource requirements in project networks subject to a fixed completion time. Operations Research 32 (1), 89–120.

Neumann, K., Zimmermann, J., 2000. Procedures for resource leveling and net present value problems in project scheduling with general temporal and resource constraints. European Journal of Operational Research 127 (2), 425 – 443.

Patterson, J. H., Huber, W. D., 1974. A horizon-varying, zero-one approach to project scheduling. Management Science 20 (6), 990–998.

Pritsker, A. A. B., Waiters, L. J., Wolfe, P. M., 1969. Multiproject scheduling with limited resources: A zero-one programming approach. Management Science 16 (1), 93–108.

Shadrokh, S., Kianfar, F., 2007. A genetic algorithm for resource investment project scheduling problem, tardiness permitted with penalty. European Journal of Operational Research 181 (1), 86 – 101.

Weist, J., 1963. The scheduling of large projects with limited resources. PhD dissertation, Carnegie Institute of Technology.

Wiest, J. D., 1967. A heuristic model for scheduling large projects with limited resources. Management Science 13 (6), pp. B359–B377.

Yamashita, D. S., Armentano, V. A., Laguna, M., 2006. Scatter search for project scheduling with resource availability cost. European Journal of Operational Research 169 (2), 623 – 637.

Conclusion

Le problème d'ordonnancement de projet à ressources limitées (POPRL) est devenu ces dernières années un problème de base dont plusieurs variantes font régulièrement l'objet de nouvelles publications. Il consiste en l'ordonnancement d'un ensemble de tâches, à l'aide d'un ou de plusieurs types de ressources renouvelables ou non renouvelables en quantités limitées.

La résolution d'un POPRL a pour but la détermination des dates d'exécution des tâches en tenant compte des contraintes de préséance et de disponibilité des ressources et ayant comme objectif la minimisation de la durée totale du projet. Le POPRL est un problème d'optimisation combinatoire de complexité NP-dur (Blazewicz et al 1983). Une revue de la littérature relative au POPRL est présentée au chapitre 2. Plus de 125 articles scientifiques sont analysés. Les contributions relatives à ce problème portent sur les méthodes exactes de résolution, la détermination de bornes inférieures sur la durée du projet et les méthodes heuristiques (approchées) de résolution. Afin d'avoir une meilleure représentation des situations partiques plusieurs généralisations du problème classique ont été publiées. Cette analyse nous a permis de constater que malgré les efforts déployés pour définir des POPRL plus généraux, les contraintes de transfert des ressources continuent à être ignorées, nous constatons aussi que l'optimisation du problème en considérant les coûts a été très peu traité dans la littérature. Cela force les gestionnaires dans la plus part des cas à se baser uniquement sur leur expérience pour réaliser ou ajuster manuellement les ordonnancements produits par des heuristiques conçues pour résoudre des versions simplifiées du problème. Afin de combler partiellement ces lacunes nous avons consacré les chapitres 3, 4 et 5 de cette thèse a étudier trois généralisations du problème d'ordonnancement de projet à ressources limitées.

Nous avons traité, dans le chapitre 3, le problème d'ordonnancement de projet à ressources limitées avec des temps de transfert des ressources (POPRLTT). Un temps de transfert est le temps nécessaire pour transférer une ou plusieurs ressources entre les différents lieux d'exécution des tâches. Ainsi, le temps de transfert d'une ressource dépend des lieux des activités à exécuter, ainsi que des caractéristiques des ressources à transférer. Il est à noter que dans ce chapitre, nous assumons que la préemption n'est pas permise, et les contraintes de préséances sont dit de type début-fin sans décalage. On assume aussi que les durées des

tâches et les temps de transfert des ressources sont connues.

L'objectif dans un POPRLTT est la détermination des dates d'exécution des tâches en tenant compte des contraintes de préséance et de disponibilité des ressources et les temps de transfert des ressources, en ayant comme objectif la minimisation de la durée totale du projet. Nous proposons un nouvel algorithme génétique basé sur un nouveau opérateur de croisement de deux positions. L'étude expérimentale menée sur un grand nombre de problèmes-tests prouve la supériorité de l'algorithme génétique proposé par rapport aux méthodes déja existantes dans la littérature.

Au chapitre 4 nous avons proposé la généralisation du problème d'ordonnancement de projet à ressources limitées avec des temps de transfert des ressources en contexte multi modes, (POPRL/PME). Dans ce problème, nous supposons que la préemption est non autorisée, que des ressources renouvelables et non renouvelables sont utilisées, que chaque activité a plusieurs modes d'exécution, et que les relations de préséance sont de type début-fin sans décalage.

Pour ce problème l'objectif est de choisir un temps de début ou de fin et un mode d'exécution pour chaque tâche du projet, de facon à ce que la durée du projet soit minimisée tout en respectant les contraintes de préséance, de disponibilité de ressources et les temps de transfert. Au meilleur de notre connaissance, cette version du problème n'a jamais été abordé auparavant. Nous proposons une formulation mathématique de celle-ci. Ensuite, nous présentons un algorithme génétique que nous avons conçu pour résoudre les instances de grande tailles. Pour tester les méthodes proposées nous développons de nouveaux ensembles de problèmes-tests pour le POPRL/PMETT, qui pourront être utilisés dans l'avenir pour stimuler la recherche dans ce domaine.

Dans le chapitre 5, nous définissons une nouvelle généralisation du problème d'ordonnancent de projet à ressources limitées en considérant l'objectif de minimiser le coût total de réalisation du projet, ce problème est appelé le problème général d'allocation et de nivellement de ressources de projet (PGANRP). Le coût d'exécution est composé de deux principaux éléments : le coût direct des ressources à utiliser et le coût des frais généraux qui ne dépend pas de la quantité des ressources allouées, mais qui est proportionnel à la durée du projet. Ce problème est très commun en pratique, mais aucune publication n'a été consacrée à ce problème. Dans ce cas, nous devons simultanément déterminer les quantités des ressources à allouer au projet au cours de son exécution et minimiser la variabilité de l'utilisation des ressources tout en essayant de terminer le projet à une date de fin acceptable. Les quantités des ressources à allouer au projet devraient permettre la fin du projet à cette date et devient une limite sur la disponibilité de ces ressources durant toute l'exécution du projet. Nous proposons, une formulation mathématique du problème et deux approches de recherche dans le voisinage pour résoudre le problème et particulièrement les instances de grandes tailles.

En résumé, les principales contributions de cette thèse sont :

- Proposer de nouvelles formulations mathématiques aux nouvelles généralisations du problème d'ordonnancement des tâches de projets, le POPRL/PMETT et PGANRP
- Développer des heuristiques efficaces pour résoudre trois généralisations du problème d'ordonnancement de projet à ressources limitées. Notamment, plusieurs algorithmes génétiques ont été developpés. On notes que ces algorithmes peuvent être utilisées pour traiter des instances de grande taille.
- Analyser et démontrer la bonne performance des heuristiques développées.

Au-delà des contributions de cette thèse, de nombreuses pistes de recherche restent encore à être explorées.

La nature combinatoire des problèmes d'ordonnancement de projet à ressources limitées limite la résolution exacte des problèmes par la programmation linéaire en nombres entiers (PLNE) à des instances très petites. Nous avons constaté que cette difficulté s'est fait sentir en faisant appel aux solveurs commerciaux actuels pour la résolution d'instances à 30 tâches pour le PGANRP, POPRLTT et même pour des instances à dix tâches pour le POPRL/PMETT. Ces limites nous encouragent à pousser la réflexion sur la possibilité d'hybrider les algorithmes génétiques dans ce travail avec des méthodes de recherche arborescente. Nous pensons que le fait de considérer les particularités de chaque problème traité pour définir des bornes inférieures et des règles de dominance nous aiderait grandement à améliorer la performance des méthodes de séparation et d'évaluation utilisées par les solveurs actuels.

Dans le cadre de cette thèse et dans la majorité des publications dans le domaine de l'ordonnancement de projet à ressources limitées on considère un contexte déterministe statique pour lequel un ordonnancement que l'on appelle aussi ordonnancement de référence est calculé avant l'exécution du projet. En pratique, cependant, un projet peut subir l'effet de certains événement imprévus. Au cours de l'exécution effective d'un projet, l'ordonnancement de référence peut en effet souffrir d'événements perturbateurs provoquant la non-faisabilité de l'ordonnancement de base prévu. Nous croyons que la généralisation du POPRLTT et du POPRL/PMETT en intégrant cette réalité s'avère une intéressante avenue de recherche qui trouvera aussi son intérêt dans la pratique courante.

Annexe A

Article intitulé «*An efficient genetic algorithm to solve the resource-constrained project scheduling problem with sequence dependent transfer times : The single mode case*» publié dans les actes de la conférence internationale PMS 2016 (*Project Management and Scheduling*,ISBN :978-84-608-7267-2)

An efficient genetic algorithm to solve the single-mode resource-constrained project scheduling problem with transfer times

Roubila Lilia Kadri and Fayez Fouad Boctor

Interuniversity Research Center on Enterprise Networks, Logistics and Transportation (CIRRELT)
Department of Operations and Decision Systems, Université Laval, 2325 rue de la Terrasse, Québec, Canada, G1V0A6
lilia.kadri.1@ulaval.ca and Fayez.Boctor@fsa.ulaval.ca

Keywords: project scheduling, transfer time, genetic algorithm.

1 Abstract

In this paper we address the single-mode Resource-Constrained Project Scheduling Problem with sequence dependent Transfer Times (RCPSPTT). We assume that preemption is not allowed, and precedence relations are zero-lag, finish-to-start relations. Also, we assume that activity durations and resource transfer times are known and deterministic. The objective is to choose a start time for each activity of the project such that the project duration is minimized while satisfying precedence relations, resource availabilities and resource-transfer time constraints. This paper proposes a new genetic algorithm using a modified magnet-based crossover operator (MMBX). Experiment conducted on a large number of instances shows that the proposed algorithm produces better results than the previously published solution methods.

2 Problem description

In the RCPSPTT we have a set J of activities, a set A of zero-lag, finish-to-start precedence relations between these activities and a set of renewable resources R. The limits on resources are denoted a_r; $\forall r \in R$. Each activity has a unique and known duration d_i and uses a constant amount of resources u_{ir}, $\forall r \in R$, for each period of its execution. We assume that transferring resources from an activity to another require time and that this transfer time is sequence dependent. We also define $I_i = [EST_i, LST_i]$, the time window between the earliest start time and the latest start time of activity i. These time windows are calculated using the critical path method while using feasible project duration, denoted T. The transfer of a unit of renewable resource r from activity i to activity j requires a transfer time denoted Δ_{ijr}. All transfer times are assumed to fulfil the triangular inequality; i.e., $\forall i, j, l \in J, \Delta_{ijr} \leq \Delta_{ilr} + \Delta_{ljr}$. Without loss of generality we assume that project activities include a dummy start activity, activity s, and a dummy finish activity, activity e. These two activities have zero duration and require the entire available amount a_r of every resource k; (i.e., $u_{sr} = u_{er} = a_r$, $\forall r \in R$). This is because the dummy start activity should provide resources to other activities while the dummy finish activity should collect them back. Transfer times from the dummy start activity and to the dummy finish activity are nil (i.e., $\Delta_{sir} = \Delta_{ier} = 0; \forall i \in J, r \in R$). The objective in the RCPSPTT is to construct a schedule that minimizes the project duration while satisfying precedence, resource and transfer time constraints. A mathematical formulation of the RCPSPTT is given in Krüger and Scholl (2009).

diate predecessor of the activity in position i+1 and $x_1 < \mu$; the mutation probability. In our implementation of the GA we maintain $\mu = 0.05$.
After computing the fitness of each new individual, we add offspring individuals to the current population. Then among the G parent individuals and G new individuals, the best G individuals are selected and placed in the next generation. This process is repeated until the total number of solutions generated by the algorithm reaches a pre-selected number L.

4 Computational results

To assess the performance of the proposed genetic algorithm we use two instance-sets composed of a total of 869 instances. These instances are the 480 instances of the set $J30$ and 389 instances among the 480 instances of the set $J60$ of the resource-constrained project scheduling instances generated by the instance generator PROGEN (Kolisch and Sprecher (1996)) and provided by the project scheduling problem library PSPLIB, to which we added transfer times. The problem parameters are characterized as follows : The network complexity NC is measured by the average number of immediate predecessors in the network ; The resource factor RF is the portion of resources consumed; and the resources strength RS which is a scaling parameter expressing the resource availability as a convex combination of the sum of the minimum amount of the resources that allows to schedule the project and the amount required to schedule activities at their earliest start time. For all these instances the network complexity NC is between 1.5 and 2.1 while the resource factor RF is between 0.25 and 1 and resource strength RS is between 0.25 and 1. These instances are those for which we were able to obtain their optimal solution. The added transfer times were determined in a way that makes optimal solutions of these instances without transfer times remain optimal for the corresponding instances with transfer times. This way of generating transfer times is similar to what has been done by Krüger (2009), Krüger and Scholl (2009) and Poppenborg and Knust (2014) to generate their test-instance. We used the commercial MIP code GUROBI 5.0.1 to solve the test instances. This MIP code was able to solve all the $J30$ instances but not all the $J60$ instances. So we only used 389 instances for which the optimal solution was found. Krüger and Scholl (2009) indicated that they were able to get the optimal solution for 400 among the 480 instances of the $J60$ set. However, their solutions are not provided anywhere. The proposed genetic algorithm is implemented in Matlab release 2014 and all calculations were run on a computer with i7 core processor, 2 GHz and 6 GB of internal memory. Then all the 869 instances were solved by the proposed GA with a limit of 5000 generated solutions.Tables 1 gives the percentage deviation from the optimal solution for the $J30$ and $J60$ instances. From this table we can see that the proposed GA obtained an average deviation from the optimum of 0.12% and 0.36% for the $J30$ and $J60$ instances respectively.

3 The proposed genetic algorithm

In this section we develop a new genetic algorithm approach to RCPSPTT. Our GA starts with an initial population of G individuals generated through a modified version of the regret-based biased random sampling mechanism proposed by Kolisch (1996). Within this heuristic, three priority rules are used : the LFT (latest finish time), LST (latest start time) and the MINSLK (minimum total slack) rules. Each individual is coded by an ordered activity list, a binary indicating the type of heuristic (serial or parallel) to be used to translate the list into a feasible schedule, and another binary indicating the direction (forward or backward) to be followed while building this schedule. The genetic algorithm then determines the fitness value of each individual, given by the duration of the corresponding schedule. The serial and the parallel scheduling scheme are adapted to take into consideration transfer times. Then the genetic algorithm randomly selects a number of pairs of individuals (solutions), and uses a Modified version of the Magnet-Based Crossover operator (called hereafter MMBX). The MBX operator suggested by Zamani (2013) works as follows. First a contiguous block of genes is selected from the first parent, called the donator, by randomly drawing two numbers n_1 and n_2 ($1 < n_1 < n_2 < |J|$) which become the bounding positions in the donator sequence specifying the required contiguous block. Then we determine both the position of the first and the last element belonging to this block in the sequence of the second parent, called receiver. Let these position be denoted n_0 and n_3 respectively. The offspring sequence is built by concatenating: activities in positions 1 to n_0-1 in the receiver sequence; activities in positions between n_0 and n_3 in the receiver sequence which are predecessors to any activity in the block; activities in the block; activities in positions between n_0 and n_3 in the receiver sequence who are successors to any activity in the block; and activities in positions between n_3+1 and $|J|$ in the receiver sequence. However, some of the activities in the positions between n_0 and n_3 in the receiver sequence may neither be a predecessor nor a successor to any activity in the block. These activities are called free activities, and are randomly placed either before or after the activities of the block in the offspring sequence. Let q be the number of free activities, Zamani suggest that whenever a free activity is encountered, it is placed before the block with a chance of p to overturn this decision. And once this decision is overturned, the remaining free activities are placed after the block. Zamini fixes $p = 0.5$ for $q = 1$ and $p = 2/(q + 2)$ for $q > 1$.

In our MMBX operator, free activities are randomly placed either before or after the activities of the block as long as this does not violate any precedence relations. This is implemented as follows. For every free activity we determine all its possible positions that do not lead to violating precedence relations and we randomly chose one of these positions. The two additional binaries (indicating whether a serial or parallel scheme is to be used and the direction of applying it; Forward or backward) are copied from the code of the donator solution.

This modification of the MBX operator makes it slightly more disruptive than the original one and thus introduces more diversification in the process. Our experiment shows that the MMBX operator produces slightly better results than the original MBX.

Notice that from every pair of parent solutions we generate two offspring solutions. To generate the first offspring, we use the first parent solution as donator and the second as receiver while to generate the second offspring we consider the second parent as donator and the first one as receiver.

Once the new individuals are generated, a mutation is applied to every individual as follows. We randomly select a position, say i, in the activity sequence (i.e., $1 < i < |J| - 2$) and a random number x_1 is drawn from a uniform distribution between 0 and 1. Then we permute the activities in positions i and i+1 if the activity in position i is not an imme-

Table 1: Average percentage deviation from the optimum as obtained by the proposed genetic algorithm

Set	NC	RF	RS	Number of instances	Average percentage deviation from optimum	Number of times the optimal is obtained
*J*30	[1.5 , 2.1]	[0.25 , 1]	0.25	120	0.39	105
	[1.5 , 2.1]	[0.25 , 1]	0.50	120	0.11	114
	[1.5 , 2.1]	[0.25 , 1]	0.75	120	0.00	120
	[1.5 , 2.1]	[0.25 , 1]	1.0	120	0.00	120
All				480	0.12	459
*J*60	[1.5 , 2.1]	[0.25 , 1]	0.25	35	1.13	22
	[1.5 , 2.1]	[0.25 , 1]	0.50	114	0.887	68
	[1.5 , 2.1]	[0.25 , 1]	0.75	120	0.00	120
	[1.5 , 2.1]	[0.25 , 1]	1.0	120	0.00	120
All				389	0.36	330

Table 2 compare the GA developed in this paper with the genetic algorithm of Krüger (2009) and the flow based tabu search algorithm of Poppenborg and Knust (2014). It gives the average percentage deviations from the optimal project duration. The results show that the new genetic algorithm developed here leads to the best results, outperforming the two previously published heuristics.

Table 2: Average percentage deviation from the optimum : a comparison with the published solution methods

Procedure	*J*30	*J*60
GA Proposed in this paper	**0.12**	**0.36**
Krüger' s GA (2009)	0.16	0.38
Popenborg and Knust' s TS (2014)	1.27	0.98

5 Conclusion

In this paper, the Resource Constrained Project Scheduling Problem with sequence dependent Transfer Times RCPSPTT is considered. We assume that a sequence dependent transfer time is needed to transfer resources between activities. A genetic algorithm using a new crossover operator is proposed and experimental results show that this GA is able to efficiently solve this problem. This GA has been compared to methods published in the literature using a set of instances constructed by PROGEN project generator to which we added transfer times. The obtained results show that the proposed genetic algorithm outperforms the genetic algorithm developed by Krüger (2009) as well as the Tabou search algorithm of Poppenborg and Knust (2014).

References

Kolisch R.,1996, "Serial and parallel resource-constrained project scheduling methods revisited: Theory and computation", *European Journal of Operational Research* , Vol. 90, pp. 320-333.

Kolisch R. and A. Sprecher, 1996, "PSPLIB - A project scheduling problem library ", *European Journal of Operational Research* , Vol. 96, pp. 205-216.

Krüger, D., 2009, "Multi-Project scheduling with Transfers", *PhD dissertation, University of Jena.*

Annexe B: Solutions optimales des instances-tests du POPRLTT

PSPLIB caracteristics				Optimal solution	
Number	Tasks	Parameter	Instance	Makespan	CPU (sec)
1	30	1	1	43	0,23
2	30	1	2	47	0,05
3	30	1	3	47	0,10
4	30	1	4	62	0,41
5	30	1	5	39	0,74
6	30	1	6	48	0,31
7	30	1	7	60	0,02
8	30	1	8	53	0,02
9	30	1	9	49	0,68
10	30	1	10	45	0,25
11	30	2	1	38	0,18
12	30	2	2	51	0,07
13	30	2	3	43	0,01
14	30	2	4	43	0,02
15	30	2	5	51	0,02
16	30	2	6	47	0,01
17	30	2	7	47	0,02
18	30	2	8	54	0,27
19	30	2	9	54	0,11
20	30	2	10	43	0,11
21	30	3	1	72	0,01
22	30	3	2	40	0,01
23	30	3	3	57	0,03
24	30	3	4	98	0,02
25	30	3	5	53	0,16
26	30	3	6	54	0,02
27	30	3	7	48	0,01
28	30	3	8	54	0,01
29	30	3	9	65	0,05
30	30	3	10	59	0,08
31	30	4	1	49	0,01
32	30	4	2	60	0,01
33	30	4	3	47	0,01
34	30	4	4	57	0,01
35	30	4	5	59	0,02
36	30	4	6	45	0,01

Following the J30 on the next page...

PSPLIB caracteristics				Optimal solution	
Number	Tasks	Parametre	Instance	Makespan	CPU (sec)
continue...					
37	30	4	7	56	0,01
38	30	4	8	55	0,01
39	30	4	9	38	0,01
40	30	4	10	48	0,01
41	30	5	1	53	17,91
42	30	5	2	82	95,15
43	30	5	3	76	245,80
44	30	5	4	63	240,01
45	30	5	5	76	26,97
46	30	5	6	64	15,77
47	30	5	7	76	97,54
48	30	5	8	67	68,16
49	30	5	9	49	2,49
50	30	5	10	70	24,95
51	30	6	1	59	4,28
52	30	6	2	51	1,00
53	30	6	3	48	0,83
54	30	6	4	42	0,98
55	30	6	5	67	1,14
56	30	6	6	37	0,15
57	30	6	7	46	0,15
58	30	6	8	39	0,15
59	30	6	9	51	0,27
60	30	6	10	61	1,91
61	30	7	1	55	0,05
62	30	7	2	42	0,01
63	30	7	3	42	0,04
64	30	7	4	44	0,02
65	30	7	5	44	0,41
66	30	7	6	35	0,02
67	30	7	7	50	0,01
68	30	7	8	44	0,03
69	30	7	9	60	0,13
70	30	7	10	49	0,21
71	30	8	1	44	0,01
72	30	8	2	51	0,01
73	30	8	3	53	0,02
74	30	8	4	48	0,02
75	30	8	5	58	0,04
76	30	8	6	47	0,02
77	30	8	7	41	0,01
78	30	8	8	51	0,02
79	30	8	9	39	0,01
80	30	8	10	67	0,01
81	30	9	1	83	54,33
82	30	9	2	92	27,24
83	30	9	3	68	135,74
84	30	9	4	71	543,57
85	30	9	5	70	45,27
86	30	9	6	59	213,00
87	30	9	7	63	969,15

Following the J30 on the next page...

PSPLIB caracteristics				Optimal solution	
Number	Tasks	Parametre	Instance	Makespan	CPU (sec)
continue...					
88	30	9	8	91	2464,42
89	30	9	9	63	1556,75
90	30	9	10	88	788,89
91	30	10	1	42	0,05
92	30	10	2	56	3,38
93	30	10	3	62	4,98
94	30	10	4	58	4,28
95	30	10	5	41	0,79
96	30	10	6	44	1,58
97	30	10	7	49	0,36
98	30	10	8	54	2,83
99	30	10	9	49	0,09
100	30	10	10	41	2,35
101	30	11	1	54	1,21
102	30	11	2	56	0,06
103	30	11	3	81	0,08
104	30	11	4	63	0,17
105	30	11	5	49	0,81
106	30	11	6	44	0,05
107	30	11	7	36	0,40
108	30	11	8	62	0,12
109	30	11	9	67	0,10
110	30	11	10	38	0,03
111	30	12	1	47	0,02
112	30	12	2	46	0,02
113	30	12	3	37	0,02
114	30	12	4	63	0,03
115	30	12	5	47	0,04
116	30	12	6	53	0,03
117	30	12	7	55	0,05
118	30	12	8	35	0,01
119	30	12	9	52	0,05
120	30	12	10	57	0,05
121	30	13	1	58	7208,90
122	30	13	2	62	1409,73
123	30	13	3	76	178,05
124	30	13	4	72	48,57
125	30	13	5	67	3329,75
126	30	13	6	64	1114,68
127	30	13	7	77	151,06
128	30	13	8	106	23,80
129	30	13	9	71	239,17
130	30	13	10	64	987,83
131	30	14	1	50	4,05
132	30	14	2	53	158,14
133	30	14	3	58	5,27
134	30	14	4	50	8,61
135	30	14	5	52	1,29
136	30	14	6	35	3,54
137	30	14	7	50	52,20
138	30	14	8	54	0,07

Following the J30 on the next page...

PSPLIB caracteristics				Optimal solution	
Number	Tasks	Parametre	Instance	Makespan	CPU (sec)
continue...					
139	30	14	9	46	4,74
140	30	14	10	61	0,66
141	30	15	1	46	0,08
142	30	15	2	47	0,06
143	30	15	3	48	0,08
144	30	15	4	48	0,06
145	30	15	5	58	6,21
146	30	15	6	67	0,15
147	30	15	7	47	0,08
148	30	15	8	50	0,19
149	30	15	9	54	0,13
150	30	15	10	65	0,06
151	30	16	1	51	0,04
152	30	16	2	48	0,03
153	30	16	3	36	0,01
154	30	16	4	47	0,03
155	30	16	5	51	0,04
156	30	16	6	51	0,04
157	30	16	7	34	0,01
158	30	16	8	44	0,02
159	30	16	9	44	0,06
160	30	16	10	51	0,06
161	30	17	1	64	4,10
162	30	17	2	68	0,06
163	30	17	3	60	0,01
164	30	17	4	49	0,35
165	30	17	5	47	0,52
166	30	17	6	63	0,01
167	30	17	7	57	0,21
168	30	17	8	61	0,18
169	30	17	9	48	0,02
170	30	17	10	66	0,01
171	30	18	1	53	0,02
172	30	18	2	55	0,01
173	30	18	3	56	0,04
174	30	18	4	70	0,06
175	30	18	5	52	0,04
176	30	18	6	62	0,25
177	30	18	7	48	0,05
178	30	18	8	52	0,01
179	30	18	9	47	0,15
180	30	18	10	49	0,04
181	30	19	1	40	0,03
182	30	19	2	58	0,01
183	30	19	3	83	0,02
184	30	19	4	39	0,01
185	30	19	5	48	0,06
186	30	19	6	49	0,02
187	30	19	7	57	0,04
188	30	19	8	55	0,03
189	30	19	9	38	0,00

Following the J30 on the next page...

PSPLIB caracteristics				Optimal solution	
Number	Tasks	Parametre	Instance	Makespan	CPU (sec)
continue...					
190	30	19	10	47	0,02
191	30	20	1	57	0,01
192	30	20	2	70	0,01
193	30	20	3	49	0,01
194	30	20	4	43	0,00
195	30	20	5	61	0,01
196	30	20	6	51	0,01
197	30	20	7	42	0,01
198	30	20	8	51	0,01
199	30	20	9	41	0,01
200	30	20	10	37	0,01
201	30	21	1	84	28,08
202	30	21	2	59	28,57
203	30	21	3	76	1,61
204	30	21	4	70	34,36
205	30	21	5	55	14,51
206	30	21	6	76	43,47
207	30	21	7	65	24,91
208	30	21	8	62	9,04
209	30	21	9	69	32,36
210	30	21	10	69	33,51
211	30	22	1	42	0,48
212	30	22	2	45	0,11
213	30	22	3	63	0,04
214	30	22	4	42	0,36
215	30	22	5	52	0,35
216	30	22	6	52	0,80
217	30	22	7	60	1,98
218	30	22	8	55	6,38
219	30	22	9	76	0,07
220	30	22	10	55	0,58
221	30	23	1	63	0,04
222	30	23	2	53	0,02
223	30	23	3	46	0,02
224	30	23	4	65	0,07
225	30	23	5	52	0,42
226	30	23	6	48	0,15
227	30	23	7	60	0,20
228	30	23	8	48	0,03
229	30	23	9	63	0,11
230	30	23	10	61	0,02
231	30	24	1	53	0,01
232	30	24	2	58	0,01
233	30	24	3	69	0,03
234	30	24	4	53	0,02
235	30	24	5	51	0,02
236	30	24	6	56	0,02
237	30	24	7	44	0,01
238	30	24	8	38	0,01
239	30	24	9	43	0,00
240	30	24	10	53	0,02

Following the J30 on the next page...

PSPLIB caracteristics				Optimal solution	
Number	Tasks	Parametre	Instance	Makespan	CPU (sec)
continue...					
241	30	25	1	93	1538,48
242	30	25	2	75	1279,61
243	30	25	3	76	2498,47
244	30	25	4	81	3090,32
245	30	25	5	72	2688,62
246	30	25	6	58	851,44
247	30	25	7	95	24,34
248	30	25	8	69	1692,71
249	30	25	9	84	344,31
250	30	25	10	58	50,84
251	30	26	1	59	0,64
252	30	26	2	40	0,04
253	30	26	3	58	0,06
254	30	26	4	62	0,10
255	30	26	5	74	0,14
256	30	26	6	53	4,39
257	30	26	7	56	0,10
258	30	26	8	66	0,18
259	30	26	9	43	3,11
260	30	26	10	49	1,08
261	30	27	1	43	0,03
262	30	27	2	58	0,06
263	30	27	3	60	0,03
264	30	27	4	64	0,03
265	30	27	5	49	0,05
266	30	27	6	59	0,04
267	30	27	7	49	0,60
268	30	27	8	66	0,08
269	30	27	9	55	0,06
270	30	27	10	62	0,08
271	30	28	1	69	0,04
272	30	28	2	57	0,03
273	30	28	3	40	0,01
274	30	28	4	49	0,02
275	30	28	5	73	0,04
276	30	28	6	55	0,02
277	30	28	7	48	0,02
278	30	28	8	53	0,02
279	30	28	9	62	0,03
280	30	28	10	59	0,05
281	30	29	1	85	732,49
282	30	29	2	90	6681,36
283	30	29	3	78	40,10
284	30	29	4	103	26,29
285	30	29	5	98	3996,86
286	30	29	6	92	48,33
287	30	29	7	73	7814,49
288	30	29	8	80	353,74
289	30	29	9	97	31,09
290	30	29	10	76	350,56
291	30	30	1	47	10,14

Following the J30 on the next page...

PSPLIB caracteristics				Optimal solution	
Number	Tasks	Parametre	Instance	Makespan	CPU (sec)
continue...					
292	30	30	2	68	21,99
293	30	30	3	55	2,57
294	30	30	4	53	2,55
295	30	30	5	54	2,48
296	30	30	6	62	79,32
297	30	30	7	68	18,83
298	30	30	8	46	2,55
299	30	30	9	46	3,61
300	30	30	10	53	11,36
301	30	31	1	43	0,04
302	30	31	2	63	0,14
303	30	31	3	58	0,09
304	30	31	4	50	0,06
305	30	31	5	52	0,95
306	30	31	6	53	0,07
307	30	31	7	61	0,05
308	30	31	8	58	0,06
309	30	31	9	50	3,13
310	30	31	10	55	10,87
311	30	32	1	61	0,04
312	30	32	2	60	0,04
313	30	32	3	57	0,03
314	30	32	4	68	0,04
315	30	32	5	54	0,04
316	30	32	6	44	0,01
317	30	32	7	35	0,01
318	30	32	8	54	0,03
319	30	32	9	65	0,05
320	30	32	10	51	0,03
321	30	33	1	65	0,03
322	30	33	2	60	0,19
323	30	33	3	55	0,34
324	30	33	4	77	0,01
325	30	33	5	53	0,76
326	30	33	6	59	0,03
327	30	33	7	58	0,02
328	30	33	8	61	0,72
329	30	33	9	65	0,94
330	30	33	10	53	0,13
331	30	34	1	68	0,11
332	30	34	2	44	0,01
333	30	34	3	69	0,02
334	30	34	4	67	0,01
335	30	34	5	63	0,06
336	30	34	6	52	0,04
337	30	34	7	58	0,04
338	30	34	8	58	0,03
339	30	34	9	60	0,02
340	30	34	10	47	0,01
341	30	35	1	57	0,01
342	30	35	2	53	0,01

Following the J30 on the next page...

PSPLIB caracteristics				Optimal solution	
Number	Tasks	Parametre	Instance	Makespan	CPU (sec)
continue...					
343	30	35	3	60	0,01
344	30	35	4	50	0,05
345	30	35	5	60	0,01
346	30	35	6	58	0,03
347	30	35	7	61	0,03
348	30	35	8	63	0,01
349	30	35	9	59	0,02
350	30	35	10	59	0,05
351	30	36	1	66	0,01
352	30	36	2	44	0,01
353	30	36	3	61	0,01
354	30	36	4	59	0,01
355	30	36	5	64	0,01
356	30	36	6	46	0,01
357	30	36	7	56	0,00
358	30	36	8	63	0,01
359	30	36	9	59	0,01
360	30	36	10	59	0,01
361	30	37	1	79	205,53
362	30	37	2	69	18,37
363	30	37	3	81	86,20
364	30	37	4	83	32,00
365	30	37	5	80	162,97
366	30	37	6	73	170,76
367	30	37	7	92	5795,31
368	30	37	8	72	25,46
369	30	37	9	57	7,88
370	30	37	10	81	11,99
371	30	38	1	48	0,41
372	30	38	2	54	0,23
373	30	38	3	59	0,06
374	30	38	4	59	0,23
375	30	38	5	71	1,59
376	30	38	6	63	0,07
377	30	38	7	65	0,20
378	30	38	8	61	0,37
379	30	38	9	63	0,54
380	30	38	10	60	0,16
381	30	39	1	55	0,03
382	30	39	2	54	0,04
383	30	39	3	54	0,10
384	30	39	4	53	0,03
385	30	39	5	55	0,01
386	30	39	6	69	0,12
387	30	39	7	56	0,06
388	30	39	8	67	0,03
389	30	39	9	64	0,10
390	30	39	10	60	0,04
391	30	40	1	51	0,01
392	30	40	2	56	0,01
393	30	40	3	57	0,01

Following the J30 on the next page...

PSPLIB caracteristics				Optimal solution	
Number	Tasks	Parametre	Instance	Makespan	CPU (sec)
continue...					
394	30	40	4	57	0,01
395	30	40	5	65	0,01
396	30	40	6	60	0,01
397	30	40	7	46	0,01
398	30	40	8	57	0,01
399	30	40	9	64	0,02
400	30	40	10	51	0,01
401	30	41	1	86	589,57
402	30	41	2	89	0,43
403	30	41	3	85	402,25
404	30	41	4	78	579,26
405	30	41	5	99	1046,51
406	30	41	6	103	1110,84
407	30	41	7	92	1399,38
408	30	41	8	88	10758,61
409	30	41	9	92	499,54
410	30	41	10	99	7,42
411	30	42	1	58	0,04
412	30	42	2	50	2,54
413	30	42	3	60	18,21
414	30	42	4	49	5,74
415	30	42	5	52	0,05
416	30	42	6	66	0,93
417	30	42	7	66	0,09
418	30	42	8	82	1,36
419	30	42	9	60	6,16
420	30	42	10	75	0,10
421	30	43	1	55	0,99
422	30	43	2	43	0,01
423	30	43	3	57	0,98
424	30	43	4	67	0,06
425	30	43	5	64	0,61
426	30	43	6	58	0,66
427	30	43	7	52	0,04
428	30	43	8	62	0,05
429	30	43	9	57	0,13
430	30	43	10	60	0,11
431	30	44	1	50	0,02
432	30	44	2	54	0,01
433	30	44	3	51	0,02
434	30	44	4	57	0,01
435	30	44	5	55	0,02
436	30	44	6	56	0,03
437	30	44	7	42	0,01
438	30	44	8	49	0,02
439	30	44	9	64	0,02
440	30	44	10	63	0,02
441	30	45	1	82	1096,83
442	30	45	2	125	0,27
443	30	45	3	92	1571,61
444	30	45	4	84	895,74

Following the J30 on the next page...

PSPLIB caracteristics				Optimal solution	
Number	Tasks	Parametre	Instance	Makespan	CPU (sec)
continue...					
445	30	45	5	86	456,02
446	30	45	6	129	1,62
447	30	45	7	101	970,25
448	30	45	8	94	0,37
449	30	45	9	82	8,32
450	30	45	10	90	0,38
451	30	46	1	59	3,49
452	30	46	2	67	8,81
453	30	46	3	65	18,97
454	30	46	4	64	8,04
455	30	46	5	57	1,18
456	30	46	6	59	10,71
457	30	46	7	59	276,78
458	30	46	8	58	9,86
459	30	46	9	49	4,85
460	30	46	10	55	30,47
461	30	47	1	58	0,03
462	30	47	2	59	0,08
463	30	47	3	55	0,07
464	30	47	4	49	0,62
465	30	47	5	47	0,95
466	30	47	6	53	2,54
467	30	47	7	66	0,93
468	30	47	8	48	0,04
469	30	47	9	65	0,08
470	30	47	10	60	7,47
471	30	48	1	63	0,02
472	30	48	2	54	0,02
473	30	48	3	50	0,02
474	30	48	4	57	0,03
475	30	48	5	58	0,05
476	30	48	6	58	0,02
477	30	48	7	55	0,04
478	30	48	8	44	0,01
479	30	48	9	59	0,03
480	30	48	10	54	0,02
J30 is finished.					

PSPLIB caracteristics				Optimal solution	
Number	Tasks	Parametre	Instance	Makespan	CPU (sec)
1	60	1	1	77	0,18
2	60	1	2	68	3,31
3	60	1	3	68	0,61
4	60	1	4	91	3,71
5	60	1	5	73	3,28
6	60	1	6	66	0,67
7	60	1	7	72	45,44
8	60	1	8	75	3,92
9	60	1	9	85	4,02
10	60	1	10	80	0,27
Following the J60 on the next page...					

PSPLIB caracteristics				Optimal solution	
Number	Tasks	Parameter	Instance	Makespan	CPU (sec)
continue...					
11	60	2	1	65	0,05
12	60	2	2	82	0,10
13	60	2	3	78	0,11
14	60	2	4	78	0,14
15	60	2	5	54	0,63
16	60	2	6	64	0,57
17	60	2	7	53	0,07
18	60	2	8	66	0,11
19	60	2	9	65	0,07
20	60	2	10	69	1,00
21	60	3	1	60	0,02
22	60	3	2	69	0,10
23	60	3	3	105	0,15
24	60	3	4	81	0,07
25	60	3	5	83	0,08
26	60	3	6	57	0,07
27	60	3	7	59	0,03
28	60	3	8	55	0,51
29	60	3	9	67	0,12
30	60	3	10	69	0,34
31	60	4	1	84	0,05
32	60	4	2	60	0,03
33	60	4	3	58	0,03
34	60	4	4	65	0,03
35	60	4	5	75	0,03
36	60	4	6	71	0,03
37	60	4	7	67	0,05
38	60	4	8	65	0,02
39	60	4	9	75	0,04
40	60	4	10	77	0,03
41	60	5	6	84	904,82
42	60	5	9	83	23,48
43	60	6	1	60	1,75
44	60	6	2	67	1,45
45	60	6	3	72	0,16
46	60	6	4	67	0,23
47	60	6	5	78	0,16
48	60	6	6	55	10,86
49	60	6	7	61	1,63
50	60	6	8	72	0,17
51	60	6	9	64	0,25
52	60	6	10	74	0,22
53	60	7	1	77	0,21
54	60	7	2	85	0,17
55	60	7	3	62	0,11
56	60	7	4	63	0,07
57	60	7	5	71	0,14
58	60	7	6	65	0,10
59	60	7	7	89	0,25
60	60	7	8	66	0,08
61	60	7	9	44	0,05

Following the J60 on the next page...

PSPLIB caracteristics				Optimal solution	
Number	Tasks	Parameter	Instance	Makespan	CPU (sec)
continue...					
62	60	7	10	82	0,20
63	60	8	1	64	0,03
64	60	8	2	61	0,02
65	60	8	3	79	0,03
66	60	8	4	64	0,02
67	60	8	5	83	0,06
68	60	8	6	56	0,02
69	60	8	7	62	0,02
70	60	8	8	66	0,03
71	60	8	9	58	0,03
72	60	8	10	97	0,05
73	60	10	1	85	0,13
74	60	10	2	62	0,23
75	60	10	3	72	0,61
76	60	10	4	80	0,11
77	60	10	5	79	0,19
78	60	10	6	67	0,22
79	60	10	7	69	6,44
80	60	10	8	65	0,72
81	60	10	9	73	4,51
82	60	10	10	73	0,11
83	60	11	1	71	0,08
84	60	11	2	61	0,06
85	60	11	3	76	0,09
86	60	11	4	69	0,06
87	60	11	5	65	0,09
88	60	11	6	70	0,06
89	60	11	7	70	0,08
90	60	11	8	69	0,09
91	60	11	9	62	0,08
92	60	11	10	58	0,06
93	60	12	1	59	0,03
94	60	12	2	58	0,03
95	60	12	3	75	0,06
96	60	12	4	69	0,06
97	60	12	5	63	0,03
98	60	12	6	54	0,03
99	60	12	7	71	0,06
100	60	12	8	60	0,03
101	60	12	9	59	0,03
102	60	12	10	79	0,09
103	60	14	1	61	57780,90
104	60	14	2	65	0,71
105	60	14	3	61	20678,56
106	60	14	4	65	57,98
107	60	14	5	59	15,78
108	60	14	6	65	1,28
109	60	14	7	69	0,98
110	60	14	8	88	0,26
111	60	14	9	61	1,04
112	60	14	10	72	31879,34

Following the J60 on the next page...

PSPLIB caracteristics				Optimal solution	
Number	Tasks	Parameter	Instance	Makespan	CPU (sec)
continue...					
113	60	15	1	84	0,26
114	60	15	2	89	0,57
115	60	15	3	72	0,44
116	60	15	4	75	0,35
117	60	15	5	70	0,21
118	60	15	6	76	0,23
119	60	15	7	64	0,22
120	60	15	8	79	0,32
121	60	15	9	72	0,24
122	60	15	10	61	0,27
123	60	16	1	64	0,14
124	60	16	2	64	0,09
125	60	16	3	53	0,14
126	60	16	4	60	0,12
127	60	16	5	66	0,22
128	60	16	6	66	0,06
129	60	16	7	82	0,24
130	60	16	8	68	0,14
131	60	16	9	54	0,13
132	60	16	10	68	0,17
133	60	17	1	86	8,42
134	60	17	2	69	1,43
135	60	17	3	89	4,00
136	60	17	4	71	0,43
137	60	17	5	59	5,78
138	60	17	6	69	1,50
139	60	17	7	83	1,30
140	60	17	8	85	879,62
141	60	17	9	76	8,54
142	60	17	10	72	5,72
143	60	18	1	81	0,42
144	60	18	2	69	0,10
145	60	18	3	77	0,10
146	60	18	4	71	0,05
147	60	18	5	80	0,09
148	60	18	6	61	0,05
149	60	18	7	93	0,49
150	60	18	8	78	0,14
151	60	18	9	69	0,04
152	60	18	10	97	0,07
153	60	19	1	62	0,03
154	60	19	2	83	0,04
155	60	19	3	83	0,08
156	60	19	4	67	0,09
157	60	19	5	73	0,04
158	60	19	6	69	0,04
159	60	19	7	60	0,05
160	60	19	8	87	0,27
161	60	19	9	69	0,23
162	60	19	10	78	0,02
163	60	20	1	60	0,02

Following the J60 on the next page...

PSPLIB caracteristics				Optimal solution	
Number	Tasks	Parameter	Instance	Makespan	CPU (sec)
continue...					
164	60	20	2	78	0,04
165	60	20	3	69	0,01
166	60	20	4	86	0,03
167	60	20	5	71	0,04
168	60	20	6	97	0,06
169	60	20	7	74	0,03
170	60	20	8	65	0,03
171	60	20	9	74	0,03
172	60	20	10	70	0,04
173	60	21	6	84	0,01
174	60	22	1	64	1,00
175	60	22	2	83	0,52
176	60	22	3	70	0,87
177	60	22	4	73	133,38
178	60	22	5	76	0,24
179	60	22	6	79	0,28
180	60	22	7	69	0,41
181	60	22	8	59	1,22
182	60	22	9	65	0,10
183	60	22	10	70	0,22
184	60	23	1	75	0,13
185	60	23	2	69	0,10
186	60	23	3	78	0,14
187	60	23	4	83	0,15
188	60	23	5	72	0,10
189	60	23	6	81	0,22
190	60	23	7	60	0,06
191	60	23	8	72	0,12
192	60	23	9	64	0,08
193	60	23	10	68	0,10
194	60	24	1	65	0,03
195	60	24	2	55	0,02
196	60	24	3	67	0,07
197	60	24	4	78	0,10
198	60	24	5	76	0,08
199	60	24	6	75	0,08
200	60	24	7	68	0,06
201	60	24	8	81	0,06
202	60	24	9	80	0,06
203	60	24	10	66	0,06
204	60	26	1	80	0,25
205	60	26	2	66	34,86
206	60	26	3	76	155,70
207	60	26	4	67	267,67
208	60	26	5	61	0,92
209	60	26	6	74	21,12
210	60	26	7	72	0,23
211	60	26	8	89	0,27
212	60	26	9	65	268,29
213	60	26	10	85	0,22
214	60	27	1	96	0,25

Following the J60 on the next page...

PSPLIB caracteristics				Optimal solution	
Number	Tasks	Parameter	Instance	Makespan	CPU (sec)
continue...					
215	60	27	2	74	0,12
216	60	27	3	76	0,22
217	60	27	4	60	0,07
218	60	27	5	78	0,24
219	60	27	6	64	0,16
220	60	27	7	83	0,22
221	60	27	8	88	0,25
222	60	27	9	76	0,13
223	60	27	10	57	0,13
224	60	28	1	92	0,16
225	60	28	2	64	0,10
226	60	28	3	72	0,12
227	60	28	4	84	0,26
228	60	28	5	71	0,10
229	60	28	6	89	0,14
230	60	28	7	75	0,10
231	60	28	8	62	0,12
232	60	28	9	74	0,09
233	60	28	10	74	0,06
234	60	30	1	70	4,42
235	60	30	3	82	105,76
236	60	30	4	76	0,37
237	60	30	6	68	0,14
238	60	30	8	63	5,72
239	60	30	9	98	0,81
240	60	31	1	65	0,24
241	60	31	2	74	0,11
242	60	31	3	66	0,20
243	60	31	4	68	0,23
244	60	31	5	72	0,31
245	60	31	6	72	0,24
246	60	31	7	76	0,26
247	60	31	8	75	0,17
248	60	31	9	86	0,30
249	60	31	10	56	0,12
250	60	32	1	69	0,25
251	60	32	2	114	0,24
252	60	32	3	85	0,23
253	60	32	4	56	0,10
254	60	32	5	77	0,18
255	60	32	6	93	0,17
256	60	32	7	76	0,27
257	60	32	8	76	0,24
258	60	32	9	74	0,23
259	60	32	10	77	0,17
260	60	33	1	105	3,70
261	60	33	2	100	0,16
262	60	33	3	79	0,58
263	60	33	4	81	1,32
264	60	33	5	108	18,58
265	60	33	6	75	25,67
Following the J60 on the next page...					

PSPLIB caracteristics				Optimal solution	
Number	Tasks	Parameter	Instance	Makespan	CPU (sec)
continue...					
266	60	33	7	78	11,02
267	60	33	8	79	7,77
268	60	33	9	108	6,84
269	60	33	10	84	1,56
270	60	34	1	72	1,03
271	60	34	2	68	0,54
272	60	34	3	61	0,19
273	60	34	4	83	0,12
274	60	34	5	80	0,11
275	60	34	6	81	0,32
276	60	34	7	85	0,68
277	60	34	8	63	0,28
278	60	34	9	77	0,08
279	60	34	10	92	0,08
280	60	35	1	78	0,03
281	60	35	2	77	0,03
282	60	35	3	89	0,19
283	60	35	4	72	0,04
284	60	35	5	76	0,52
285	60	35	6	79	0,06
286	60	35	7	73	0,03
287	60	35	8	78	0,07
288	60	35	9	76	0,23
289	60	35	10	71	0,03
290	60	36	1	61	0,02
291	60	36	2	75	0,02
292	60	36	3	81	0,03
293	60	36	4	85	0,03
294	60	36	5	57	0,02
295	60	36	6	76	0,02
296	60	36	7	71	0,02
297	60	36	8	69	0,03
298	60	36	9	86	0,03
299	60	36	10	77	0,04
300	60	37	9	96	0,01
301	60	37	10	96	485,14
302	60	38	1	73	0,14
303	60	38	2	76	2331,30
304	60	38	3	77	1,05
305	60	38	4	58	2,59
306	60	38	5	103	0,23
307	60	38	6	86	0,46
308	60	38	7	74	0,65
309	60	38	8	71	13,08
310	60	38	9	66	0,40
311	60	38	10	66	8,37
312	60	39	1	80	0,09
313	60	39	2	84	0,06
314	60	39	3	83	0,14
315	60	39	4	92	0,13
316	60	39	5	73	0,09

Following the J60 on the next page...

PSPLIB caracteristics				Optimal solution	
Number	Tasks	Parameter	Instance	Makespan	CPU (sec)
continue...					
317	60	39	6	84	0,90
318	60	39	7	68	0,06
319	60	39	8	77	0,16
320	60	39	9	72	0,09
321	60	39	10	74	0,09
322	60	40	1	86	0,06
323	60	40	2	81	0,06
324	60	40	3	70	0,03
325	60	40	4	87	0,08
326	60	40	5	83	0,05
327	60	40	6	69	0,03
328	60	40	7	68	0,03
329	60	40	8	80	0,05
330	60	40	9	90	0,07
331	60	40	10	73	0,08
332	60	42	1	83	0,37
333	60	42	2	68	0,30
334	60	42	4	103	0,03
335	60	42	5	73	1,14
336	60	42	6	82	6,84
337	60	42	7	59	639,06
338	60	42	8	82	142,74
339	60	42	9	71	9,69
340	60	42	10	87	1,74
341	60	43	1	108	0,25
342	60	43	2	85	0,27
343	60	43	3	74	0,14
344	60	43	4	75	1,41
345	60	43	5	64	0,09
346	60	43	6	84	0,21
347	60	43	7	89	0,19
348	60	43	8	69	0,08
349	60	43	9	70	0,17
350	60	43	10	78	0,22
351	60	44	1	84	0,11
352	60	44	2	68	0,10
353	60	44	3	87	0,17
354	60	44	4	77	0,13
355	60	44	5	74	0,06
356	60	44	6	81	0,11
357	60	44	7	76	0,08
358	60	44	8	83	0,26
359	60	44	9	65	0,06
360	60	44	10	65	0,09
361	60	46	1	79	65,85
362	60	46	2	78	1,05
363	60	46	3	79	35549,98
364	60	46	4	74	16098,22
365	60	46	5	91	15789,98
366	60	46	6	90	3782,48
367	60	46	7	78	18976,89
Following the J60 on the next page...					

PSPLIB caracteristics				Optimal solution	
Number	Tasks	Parameter	Instance	Makespan	CPU (sec)
continue...					
368	60	46	8	75	476,71
369	60	46	9	69	45685,69
370	60	47	1	75	0,23
371	60	47	2	66	0,37
372	60	47	3	69	0,79
373	60	47	4	76	0,21
374	60	47	5	87	0,24
375	60	47	6	76	0,24
376	60	47	7	68	0,14
377	60	47	8	71	0,58
378	60	47	9	76	0,15
379	60	47	10	66	0,83
380	60	48	1	71	0,19
381	60	48	2	87	0,20
382	60	48	3	84	0,34
383	60	48	4	62	0,11
384	60	48	5	101	0,20
385	60	48	6	66	0,14
386	60	48	7	77	0,17
387	60	48	8	88	0,42
388	60	48	9	82	0,20
389	60	48	10	70	0,15
J60 is finished.					

Annexe C

Article intitulé «*Multi-Mode, Resource-Constrained Project Scheduling with sequence dependent Transfer Times*». est publié dans les actes de la conférence internationale PMS 2014 (Project Management and Scheduling, ISBN : 978-3-00-045630-5)

Multi-Mode Resource-Constrained Project Scheduling with Sequence Dependent Transfer Times

Roubila Lilia Kadri and Fayez F. Boctor
Centre Interuniversitaire de Recherche sur le Réseau d'entreprises, la Logistique et le Transport
Faculté des sciences de l'administration, Université Laval, Québec, Canada.

Keywords: Project scheduling, Multi-mode activities, Resource transfer times.

1. Introduction

This paper deals with the *Multi-Mode Resource-Constrained Project Scheduling Problem with sequence dependent Transfer Times* (MMRCPSPTT). A sequence dependent transfer time is the time needed to transfer the required resource from the site of the preceding activity to the site of the succeeding one. Thus transfer time of a resource depends on the locations of the activities to execute as well as on the characteristics of the resource to transfer.

In the studied problem we assume that preemption is not allowed, the used resources are either renewable or non-renewable, each activity has a discrete number of execution modes and precedence relations are zero-lag finish-to-start relations. Also we assume that activity durations and transfer times are known and deterministic. The objective is to determine start time and execution mode for each activity of the project such that the project duration is minimized. Recently Krüger and Scholl (2009 and 2010) considered the single-mode resource-constrained project scheduling problem with transfer times (SMRCPSPTT). They proposed priority rule based solution procedures and presented numerical results to assess their performance.

To the best of our knowledge, the MMRCPSPTT has never been addressed before. Section 2 describes the considered MMRCPSPTT and section 3 presents a mathematical formulation of the problem. Section 4 presents a quick description of the heuristic we designed to solve large scale instances and section 5 describes the experiment conducted to assess the performance of the proposed heuristic.

2. Problem description

In the MMRCPSPSTT we have a set J of activities, a set P of finish-to-start precedence constraint, a set K of renewable resource and a set W of non-renewable resources. The limits on renewable resources are denoted Q_k; $k \in K$ and the availability over the entire project of the non-renewable resources is given by N_w; $w \in W$. Each activity i has a set of immediate predecessors $P_i \subseteq J$, a set of immediate and indirect predecessors, denoted π_i, a set of immediate successors $Z_i \subseteq J$ and a set of immediate and indirect successors, denoted σ_i.

In addition, every activity i has a set of possible execution modes M_i. A mode $m \in M_i$ is characterized by q_{imk}, the required number of units of the renewable resource k, n_{imw}, the required number of units of the non-renewable resource w, and the corresponding duration d_{im}. We also define I_i as the time window between the earliest start time and the latest start time of activity i. These times are calculated using the critical path method based on a feasible project duration T.

We assume that activities are to be performed in a number of different locations (sites). The transfer of a unit of renewable resource k from the execution location of activity i to the execution location of activity j requires a transfer time denoted Δ_{ijk}. The sum of transfer times of non-renewable resources to the project activities is constant and consequently is not considered in the problem.

Without loss of generality we assume that project activities include a dummy start activity, activity 1, and a dummy finish activity, activity F. These two activities have only one execution mode of duration zero, require no non-renewable resources and requires the entire available amount Q_k of the renewable resource k; $\forall k \in K$ (i.e., $q_{1 1k}=q_{F1k}=Q_k$). This is because the dummy start activity should provide renewable resources to the other activities while the dummy finish activity should collect them back. Transfer times from the dummy start activity and to the dummy finish activity are nil (i.e., $\Delta_{1ik}=\Delta_{iFk}=0; \forall i \in J, \forall k \in K$).

The objective in MMRCPSPTT is to assign a mode to each activity and to construct a schedule that minimizes the project duration while satisfying precedence, resource and transfer constraints.

3. Mathematical formulation

Decision variables

$f_{imjm'k}$ number of units of resource k directly transferred from activity i executed using mode m, to activity j executed using mode m'

$y_{imjm'k}$ a binary that takes the value 1 if and only if a unit or more of renewable resource k are transferred from activity i executed using mode m to activity j executed using mode m'

x_{imt} a binary that takes the value 1 if and only if activity i is executed using mode m and starts at the beginning of period t

Mathematical model

$$Minimize: \sum_{t \in I_F} t.x_{F1t} \quad (1)$$

Subject to:

$$\sum_{t \in I_i} \sum_{m \in M_i} x_{imt} = 1, \qquad \forall i \in J \quad (2)$$

$$\sum_{t \in I_j} \sum_{m' \in M_j} t x_{jm't} - \sum_{t \in I_i} \sum_{m \in M_i} (t + d_{im}) x_{imt} \geq 0 \quad \forall (i,j) \in P \quad (3)$$

$$\sum_{m' \in M_j} \sum_{t \in I_j} t x_{jm't} - \sum_{m \in M_i} \sum_{t \in I_i} (t + d_{im}) x_{imt} - T \sum_{m \in M_i} \sum_{m' \in M_j} y_{imjm'k} \geq -T + \Delta_{ijk}, \ \forall k \in K, i \in J - \{F\}, \ j \in J - \pi_i \quad (4)$$

$$2y_{imjm'k} \leq \sum_{t \in I_i} x_{imt} + \sum_{t \in I_j} x_{jm't}, \quad k \in K, i \in J - \{F\}, \ j \in J - \pi_i, m \in M_i, \ m' \in M_j \quad (5)$$

$$f_{imjm'k} \leq \min(q_{imk}, q_{jm'k}) \, y_{imjm'k}, \quad k \in K, i \in J - \{F\}, j \in J - \pi_i, m \in M_i, m' \in M_j \quad (6)$$

$$\sum_{m \in M_i} \sum_{j \in J - \pi_i} \sum_{m' \in M_j} f_{imjm'k} = \sum_{m \in M_i} \sum_{t \in I_i} q_{imk} \, x_{imt}, k \in K, i \in J - \{F\} \quad (7)$$

$$\sum_{m' \in M_j} \sum_{i \in J - \sigma_i} \sum_{m \in M_i} f_{imjm'k} = \sum_{m' \in M_j} \sum_{t \in I_j} q_{jm'k} \, x_{jm't}, k \in K, i \in J - \{1\} \quad (8)$$

$$\sum_{i \in J} \sum_{m \in M_i} \sum_{t \in I_i} n_{imw} x_{imt} \leq N_w, \forall w \in W \quad (9)$$

$$f_{imjm'k} \geq 0, \forall m \in M_i, \forall m' \in M_j, \forall k \in K, \forall i \in J - \{F\}, \forall j \in J - \pi_i \quad (10)$$

$$y_{imjm'k} \in \{0,1\}, \forall m \in M_i, \forall m' \in M_j, \forall k \in K, \forall i \in J - \{F\}, \forall j \in J - \pi_i \quad (11)$$

$$x_{imt} \in \{0,1\}, \forall i \in J, \forall m \in M_i, \forall t \in I_i \quad (12)$$

The objective function (1) minimizes the project duration. Constraints (2) ensure that each activity is performed in only one mode and is started within its time window. Constraints (3) are the precedence constraints. Constraints (4) ensure that the transfer time of the renewable resource k is taken into consideration when transferred from activity i performed using mode m to activity j performed using mode m'. Constraints (5) make sure that $y_{imjm'k}$ take the value 0 if the modes of i and j are not respectively m and m'. Constraints (6) imply that if $y_{imjm'k} = 0$ then $f_{imjm'k} = 0$; otherwise $f_{imjm'k}$ takes a value equal the less between the requirement of j and what is available at i. Constraints (6) imply that if $y_{imjm'k} = 0$ then $f_{imjm'k} = 0$. Constraints (7) and (8) represent the flow conservation of renewable resources. Constraints (9) ensure that the used quantities of non-renewable resources do not exceed the imposed limits.

4. The proposed heuristic

To solve the MMRCPSPTT we propose a three-phase heuristic. In the **first** phase, called the mode selection phase, we select an execution mode for each activity. We only require that the selected modes be feasible with respect to the non-renewable resource limits. This is done by solving an integer linear model. Two such models were tested. The first one ($M1$) is:

Find: $x_{im} \in \{0,1\}$; $\forall i \in J$, $\forall m \in M_i$ which

$$Minimize: \quad \sum_w (N_w - \sum_{i \in J} \sum_{m \in M_i} n_{imw} x_{im}) \tag{14}$$

$$Subject\ to: \quad \sum_{m \in M_i} x_{im} = 1 \tag{15}$$

$$\sum_{i \in J} \sum_{m \in M_i} n_{imw} x_{im} \leq N_w; \forall w \in W. \tag{16}$$

Where: x_{im} takes the value 1 if the mode m is assigned to the activity i.

The second model ($M2$) has the same constraints but the object function is replaced by:

$$Minimize: \quad \sum_{i \in J} \sum_{m \in M_i} d_{im} x_{im} \tag{17}$$

In the **second** phase we use the execution modes obtained in phase I and solve the resulting Single-Mode Resource Constrained Project Scheduling Problem with transfer times (SMRCPSPTT). This is done by using a combination of selection rules. The activity selection rule is the MinSlk (activity with minimum slack) rule and the resource transfer rule is the MinTT (minimum transfer time) rule. These two rules are used within a parallel scheduling scheme. Kruger et al (2009) tested several combinations of priority rules and indicated that this is the most efficient way to solve the SMRCPSPTT.

The **third** phase is an improvement phase. Two improvement procedures were tested. The first procedure ($P1$) is a simple hill climbing approach where we search the unit neighborhood where we modify the execution mode of only one activity, provided that this modification does not lead to requiring an amount of any non-renewable resource that exceeds its availability. Once a local optimal is reached, we use again the hill climbing procedure to search a pairwise neighborhood where we modify the execution mode of two activities simultaneously. The procedure stops when we reach a local optimum of this second neighborhood. The second improvement procedure ($P2$) is a multi-neighborhood improvement approached (see Boctor, 1993). The same two neighborhoods presented above are used within this approach. The multi-neighborhood approach is an iterative approach where each iteration consists of searching the first neighborhood until no more improvement can be achieved. Next we search the second neighborhood until a better solution is found. Then we go back to search the first neighborhood of the obtained solution. The procedure stops if no improvement can be achieved during a complete iteration.

5. Computational results

Test instances and optimal solutions

We conducted a computational study to assess the performance of the proposed heuristic. To do this study we extended three ProGen instance sets: the sets J10, J20 and J30 of the MMRCPSP (Kolisch et al., 1996). Transfer times are randomly sampled from a uniform distribution between 0 and 10 periods. In all, we used 536 instances with 10 non dummy activities, 554 instances with 20 non dummy activities and 552 instances with 30 non dummy activities. All coding was done in Matlab 2012 environment and all experiments were run on a computer with i7 core processor, 2 GHz and 6 GB of internal memory. GUROBI 5.5 was used to solve the proposed mathematical formulation with a run time limit of 3600 CPU-seconds per instance. Table 1 shows that the optimal solution was found for only 40% of the test-instances and that problems with RF=1(Resource Factor) are more difficult to solve than those with RF=0.5. Notice that

although we have not obtained the optimal solution for the 60 % other instances, we got at least a feasible one for each of them within the imposed run time limit.

Table 1: Number of times the optimal solution was found and average percentage deviation

Set	RF	RS	Total number of instances	Number of instances where the optimum is found	Percentage of instances where the optimum is found	Average percentage deviation from the optimum (calculated only over instances where the optimum is found)				Average percentage deviation from the best found solution (calculated over all instances)			
						*M*1*P*1	*M*1*P*2	*M*2*P*1	*M*2*P*2	*M*1*P*1	*M*1*P*2	*M*2*P*1	*M*2*P*2
All J10, J20 & J30	0,5	0.25	190	48	25,26%	7,99	6,46	5,40	6,33	5,56	3,84	4,76	3,69
		0.5	210	113	53,81%	9,67	7,77	9,52	8,64	8,24	6,33	6,85	6,20
		0.75	210	190	90,48%	8,21	6,84	6,88	6,39	8,31	6,98	6,82	6,29
		1	190	189	99,47%	5,78	4,74	4,80	4,48	5,82	4,78	4,85	4,53
	1	0.25	209	0	0,00%	-	-	-	-	2,45	1,78	2,15	1,44
		0.5	210	7	3,33%	7,81	6,39	6,86	6,86	3,31	2,38	2,14	1,57
		0.75	212	31	14,62%	11,80	11,58	9,94	9,71	4,81	4,08	3,31	2,86
		1	211	73	34,60%	9,96	9,79	8,60	8,17	5,30	5,19	4,48	4,18
	All RF-RS		1642	651	39,65%	8,10	6,92	6,96	6,58	5,47	4,43	4,41	3,84

Results obtained by four versions of the proposed heuristic

As shown in Table 1, four versions of the heuristic were tested. These versions are the result of combining *M*1 or *M*2 in the first phase with *P*1 or *P*2 in the third one. The table gives the percentage deviation from the optimum (when the optimum is obtained) and the percentage deviation from the best found solution (for all instances) for each version. Table 2 gives the number of times the optimal and the best solutions are found. Many other results are in Kadri et Boctor (2014).

Table 2: Number of times the optimal solution and the best solution are found by each version of the heuristic

Set	RF	RS	Total number of instances	Number of instances where the optimum is found	Number of times the optimum is found				Number of times the best solution is found			
					*M*1*P*1	*M*1*P*2	*M*2*P*1	*M*2*P*2	*M*1*P*1	*M*1*P*2	*M*2*P*1	*M*2*P*2
J10, J20 & J30	0,5	0.25-0.5	400	161	39	49	46	49	189	263	217	260
		0.75-1.0	400	379	154	172	154	166	255	299	281	305
	1	0.25-0.5	419	7	1	1	1	1	214	263	256	297
		0.75-1.0	423	104	18	17	16	16	252	270	287	306
All			1642	651	212	239	217	232	910	1095	1041	1168

Acknowledgement

This research was partially supported by the Canadian Natural Sciences and Engineering Research Council Grant OPG0036509. This support is gratefully acknowledged.

Reference

Boctor F.F. (1993). Discrete optimization and multi-neighborhood local improvement heuristics. Document de travail 93-35, FSA, Université Laval.

Kolisch R. and A. Sprecher (1996). PSPLIB – A project scheduling problem library. *European Journal of Operational Research*, 96: 205–216.

Krüger, D. and A.Scholl (2009). A heuristic solution framework for the resource constrained (multi-)project scheduling problem with sequence-dependent transfer times. *European Journal of Operational Research*, 197: 492-508.

Krüger, D. and A.Scholl (2010). Managing and modelling general resource transfers in (multi-)project scheduling. *OR SPECTRUM*, 32: 369-394.

Kadri R L and Boctor F F (2014) Multi-mode, resource-constrained project scheduling with sequence-dependent transfer times, document de travail, FSA Université Laval.

Annexe D : Solutions optimales des instances-tests du POPRL/PMETT

PSPLIB caracteristics				Optimal solution	
Number	Tasks	Parametre	Instance	Makespan	CPU (sec)
1	10	2	2	26	60.12
2	10	2	4	22	50.33
3	10	2	6	26	136.12
4	10	2	7	41	1539.70
5	10	2	9	23	41.53
6	10	2	10	52	186.24
7	10	3	2	25	178.53
8	10	3	3	34	2568.26
9	10	3	4	29	22.32
10	10	3	5	32	316.44
11	10	3	6	45	4356.33
12	10	3	7	24	129.30
13	10	3	8	32	579.99
14	10	3	9	36	136.36
15	10	3	10	25	35.54
16	10	4	1	38	191.44
17	10	4	2	28	8.92
18	10	4	3	37	148.53
19	10	4	4	30	213.99
20	10	4	5	44	123.70
21	10	4	6	31	68.68
22	10	4	8	26	115.70
23	10	4	9	23	8.13
24	10	4	10	20	7.28
25	10	5	1	63	6.21
26	10	5	3	37	48.83
27	10	5	4	31	192.09
28	10	5	6	47	303.74
29	10	5	8	46	2202.54
30	10	5	10	36	415.55
31	10	6	1	19	10.92
32	10	6	3	41	197.82
33	10	6	7	35	646.82
34	10	6	8	34	539.33
35	10	7	1	26	136.32
36	10	7	2	37	25.09

Following the J10 on the next page...

PSPLIB caracteristics				Optimal solution	
Number	Tasks	Parametre	Instance	Makespan	CPU (sec)
suite...					
37	10	7	3	35	360.53
38	10	7	4	24	27.84
39	10	7	5	33	108.71
40	10	7	8	46	122.84
41	10	7	9	38	60.59
42	10	8	3	20	64.12
43	10	8	5	35	51.11
44	10	8	6	39	423.14
45	10	8	7	31	53.98
46	10	8	8	22	10.71
47	10	8	10	40	1098.96
48	10	10	1	25	42.06
49	10	10	2	27	37.81
50	10	10	3	24	11.12
51	10	10	4	17	8.29
52	10	10	5	32	69.45
53	10	10	6	22	119.40
54	10	10	7	18	16.14
55	10	10	8	19	116.16
56	10	10	9	14	15.13
57	10	10	10	24	358.10
58	10	11	1	21	0.75
59	10	11	2	16	24.51
60	10	11	3	15	11.22
61	10	11	4	23	1.72
62	10	11	5	18	11.39
63	10	11	6	20	17.09
64	10	11	7	16	11.69
65	10	11	8	16	13.38
66	10	11	9	19	12.72
67	10	11	10	25	7.71
68	10	12	1	20	9.25
69	10	12	2	20	84.35
70	10	12	3	19	17.44
71	10	12	4	17	1.62
72	10	12	5	17	14.50
73	10	12	6	25	166.52
74	10	12	7	20	33.24
75	10	12	8	19	1.41
76	10	12	9	24	3.14
77	10	12	10	14	58.04
78	10	13	1	25	6.89
79	10	13	2	25	42.97
80	10	13	3	27	10.32
81	10	13	4	37	76.31
82	10	13	5	26	13.16
83	10	13	6	28	17.02
84	10	13	7	26	20.11
85	10	13	8	22	49.85
86	10	13	9	29	26.48
87	10	13	10	34	16.80

Following the J10 on the next page...

PSPLIB caracteristics				Optimal solution	
Number	Tasks	Parametre	Instance	Makespan	CPU (sec)
suite...					
88	10	14	1	19	15.89
89	10	14	2	20	15.83
90	10	14	3	28	79.16
91	10	14	4	22	28.59
92	10	14	5	20	13.13
93	10	14	6	20	24.68
94	10	14	7	25	43.42
95	10	14	8	26	32.78
96	10	14	9	25	14.04
97	10	14	10	18	46.81
98	10	15	1	16	24.92
99	10	15	2	32	113.43
100	10	15	3	17	8.99
101	10	15	4	18	22.42
102	10	15	5	12	15.64
103	10	15	6	25	73.91
104	10	15	7	14	25.99
105	10	15	8	18	18.36
106	10	15	9	16	33.83
107	10	15	10	24	63.74
108	10	16	1	18	7.79
109	10	16	2	23	40.20
110	10	16	3	13	6.94
111	10	16	4	26	8.55
112	10	16	5	27	5.43
113	10	16	6	26	21.70
114	10	16	7	19	40.70
115	10	16	8	10	7.75
116	10	16	9	20	10.86
117	10	16	10	28	9.47
118	10	18	1	17	15.72
119	10	18	2	19	11.06
120	10	18	3	18	10.89
121	10	18	4	23	14.63
122	10	18	5	21	35.05
123	10	18	6	21	21.91
124	10	18	7	14	6.20
125	10	18	8	18	8.14
126	10	18	9	18	16.74
127	10	18	10	18	10.30
128	10	19	1	16	3.95
129	10	19	2	16	5.17
130	10	19	3	19	18.66
131	10	19	4	18	11.54
132	10	19	5	15	7.02
133	10	19	6	21	5.34
134	10	19	7	15	8.86
135	10	19	8	21	9.62
136	10	19	9	17	16.99
137	10	19	10	19	10.87
138	10	20	1	18	10.85

Following the J10 on the next page...

PSPLIB caracteristics				Optimal solution	
Number	Tasks	Parametre	Instance	Makespan	CPU (sec)
suite...					
139	10	20	2	24	18.57
140	10	20	3	23	7.19
141	10	20	4	18	3.39
142	10	20	5	13	5.97
143	10	20	6	21	0.74
144	10	20	8	17	6.91
145	10	20	9	12	5.04
146	10	20	10	14	9.29
147	10	21	1	27	5.99
148	10	21	2	27	44.99
149	10	21	3	39	56.65
150	10	21	4	23	1.87
151	10	21	5	36	14.99
152	10	21	6	29	9.42
153	10	21	7	27	38.82
154	10	21	8	27	33.52
155	10	21	9	26	11.31
156	10	21	10	26	4.09
157	10	22	1	24	27.28
158	10	22	2	18	58.38
159	10	22	3	24	5.35
160	10	22	4	18	12.12
161	10	22	5	16	13.04
162	10	22	6	17	7.75
163	10	22	7	22	139.97
164	10	22	8	21	30.61
165	10	22	9	21	21.29
166	10	22	10	19	2.30
167	10	23	1	22	119.88
168	10	23	2	18	9.86
169	10	23	3	21	18.83
170	10	23	4	26	60.29
171	10	23	5	20	37.41
172	10	23	6	20	16.99
173	10	23	8	20	16.47
174	10	23	9	16	25.11
175	10	23	10	16	11.86
176	10	24	1	10	3.17
177	10	24	2	14	0.91
178	10	24	3	14	4.45
179	10	24	4	15	2.78
180	10	24	5	16	7.37
181	10	24	6	18	3.63
182	10	24	7	17	2.40
183	10	24	8	27	6.35
184	10	24	9	15	5.98
185	10	24	10	19	9.47
186	10	26	1	15	6.15
187	10	26	2	14	15.85
188	10	26	3	18	2.74
189	10	26	4	18	12.32

Following the J10 on the next page...

PSPLIB caracteristics				Optimal solution	
Number	Tasks	Parametre	Instance	Makespan	CPU (sec)
suite...					
190	10	26	5	18	6.36
191	10	26	6	18	1.18
192	10	26	7	16	3.42
193	10	26	8	17	10.56
194	10	26	9	12	4.87
195	10	26	10	19	9.96
196	10	27	1	16	10.54
197	10	27	2	15	8.71
198	10	27	3	15	1.05
199	10	27	4	17	9.09
200	10	27	5	18	2.81
201	10	27	6	14	3.43
202	10	27	7	11	3.72
203	10	27	8	12	1.03
204	10	27	9	15	1.57
205	10	27	10	15	2.90
206	10	28	1	19	0.79
207	10	28	2	24	0.56
208	10	28	3	20	0.59
209	10	28	4	16	0.96
210	10	28	5	18	0.81
211	10	28	6	10	1.00
212	10	28	7	16	1.20
213	10	28	8	16	0.61
214	10	28	9	13	0.68
215	10	28	10	18	2.08
216	10	29	1	20	3.33
217	10	29	2	22	6.97
218	10	29	3	26	5.81
219	10	29	4	23	3.52
220	10	29	5	25	4.03
221	10	29	6	27	4.34
222	10	29	7	29	4.32
223	10	29	8	23	4.65
224	10	29	9	25	9.86
225	10	29	10	29	4.43
226	10	30	1	17	2.81
227	10	30	2	17	13.97
228	10	30	3	17	0.96
229	10	30	4	13	3.01
230	10	30	5	28	3.08
231	10	30	6	22	1.59
232	10	30	7	15	1.52
233	10	30	8	14	2.22
234	10	30	9	16	11.40
235	10	30	10	17	0.98
236	10	31	1	17	4.06
237	10	31	2	15	1.57
238	10	31	3	15	1.28
239	10	31	4	18	1.33
240	10	31	5	15	1.21

Following the J10 on the next page...

PSPLIB caracteristics				Optimal solution	
Number	Tasks	Parametre	Instance	Makespan	CPU (sec)
suite...					
241	10	31	6	20	0.84
242	10	31	7	14	0.98
243	10	31	8	16	1.39
244	10	31	9	15	2.14
245	10	31	10	15	5.15
246	10	32	1	19	4.87
247	10	32	2	13	3.11
248	10	32	3	19	0.67
249	10	32	4	12	0.53
250	10	32	5	11	0.63
251	10	32	6	17	0.82
252	10	32	7	16	1.08
253	10	32	8	12	1.05
254	10	32	9	15	0.61
255	10	32	10	13	0.81
256	10	43	8	30	2280.35
257	10	44	1	30	841.34
258	10	44	9	25	436.95
259	10	45	4	23	2452.86
260	10	45	7	35	355.24
261	10	45	9	30	1294.07
262	10	47	2	20	678.79
263	10	51	1	30	3449.87
264	10	51	2	19	135.42
265	10	51	8	24	604.16
266	10	51	10	18	2041.75
267	10	52	2	28	1668.05
268	10	52	3	18	1263.85
269	10	52	4	25	714.87
270	10	52	6	19	53.46
271	10	52	8	23	814.71
272	10	52	9	27	1119.28
273	10	52	10	21	825.23
274	10	53	1	36	3108.09
275	10	53	2	34	19.74
276	10	53	4	27	489.54
277	10	53	7	23	11164.34
278	10	53	9	32	1555.01
279	10	53	10	33	37.19
280	10	54	1	23	719.92
281	10	54	3	28	1175.88
282	10	54	4	23	8755.83
283	10	54	5	17	112.39
284	10	54	7	28	5070.97
285	10	55	1	23	515.18
286	10	55	3	18	137.14
287	10	55	4	30	1433.37
288	10	55	5	18	1610.75
289	10	55	10	26	1749.58
290	10	56	1	23	3623.67
291	10	56	2	17	53.65
Following the J10 on the next page...					

PSPLIB caracteristics				Optimal solution	
Number	Tasks	Parametre	Instance	Makespan	CPU (sec)
suite...					
292	10	56	4	16	39.67
293	10	56	5	23	811.14
294	10	58	1	21	77.34
295	10	58	2	19	670.35
296	10	58	3	21	154.41
297	10	58	4	20	88.21
298	10	58	5	18	12.48
299	10	58	6	14	42.91
300	10	58	7	17	96.36
301	10	58	8	25	495.45
302	10	58	9	18	942.86
303	10	58	10	21	171.04
304	10	59	1	18	71.53
305	10	59	2	18	117.30
306	10	59	3	19	52.09
307	10	59	4	19	1702.65
308	10	59	5	17	107.56
309	10	59	6	18	116.45
310	10	59	7	15	64.57
311	10	59	8	18	301.73
312	10	59	9	20	103.37
313	10	59	10	24	61.95
314	10	60	1	21	215.49
315	10	60	2	23	38.81
316	10	60	3	13	39.35
317	10	60	4	19	605.50
318	10	60	5	21	64.68
319	10	60	6	17	1302.16
320	10	60	7	19	28.84
321	10	60	8	19	55.98
322	10	60	9	21	290.69
323	10	60	10	15	1.51
324	10	61	1	27	786.19
325	10	61	2	26	59.57
326	10	61	3	32	2.70
327	10	61	4	29	11.82
328	10	61	5	30	2128.08
329	10	61	6	33	32.82
330	10	61	7	23	26.13
331	10	61	8	34	19.54
332	10	61	9	34	585.83
333	10	61	10	42	22.11
334	10	62	1	20	51.77
335	10	62	2	26	1006.14
336	10	62	3	21	2550.88
337	10	62	4	19	428.22
338	10	62	5	23	526.53
339	10	62	7	16	92.67
340	10	62	8	23	1803.94
341	10	62	9	20	106.53
342	10	62	10	22	294.36

Following the J10 on the next page...

PSPLIB caracteristics				Optimal solution	
Number	Tasks	Parametre	Instance	Makespan	CPU (sec)
suite...					
343	10	63	3	19	209.49
344	10	63	4	17	62.66
345	10	63	6	22	2777.92
346	10	63	7	16	33.11
347	10	63	8	16	300.36
348	10	63	9	20	2320.85
349	10	63	10	15	578.78
350	10	64	1	19	67.62
351	10	64	2	18	31.96
352	10	64	3	16	37.39
353	10	64	4	13	10.93
354	10	64	6	19	199.10
355	10	64	7	13	16.76
356	10	64	8	17	36.77
357	10	64	9	22	15.07
358	10	64	10	18	26.59
J10 is finished.					

PSPLIB caracteristics				Optimal solution	
Number	Tasks	Parametre	Instance	Makespan	CPU (sec)
1	20	25	6	31	1725.74
2	20	10	2	25	324.39
3	20	10	5	28	8895.97
4	20	18	1	30	61.17
5	20	18	2	29	413.91
6	20	18	3	25	3497.38
7	20	18	5	28	4644.11
8	20	18	7	30	629.80
9	20	18	8	34	632.13
10	20	26	1	27	1272.83
11	20	26	2	26	1201.99
12	20	26	3	31	577.97
13	20	26	5	28	3067.09
14	20	26	8	32	595.32
15	20	34	1	47	10140.99
16	20	34	8	35	10623.24
17	20	42	1	24	3628.77
18	20	42	3	31	914.33
19	20	42	4	41	4553.74
20	20	42	6	35	12953.34
21	20	42	9	32	1319.71
22	20	50	1	32	9397.10
23	20	50	5	31	1453.46
24	20	50	6	27	1485.41
25	20	50	7	22	949.23
26	20	50	8	25	426.60
27	20	50	10	30	7490.70
28	20	58	1	40	1294.26
29	20	58	2	25	25.35
30	20	58	3	35	3194.02
Following the J20 on the next page...					

PSPLIB caracteristics				Optimal solution	
Number	Tasks	Parametre	Instance	Makespan	CPU (sec)
continue...					
31	20	58	4	25	415.44
32	20	58	5	26	3658.26
33	20	58	6	24	84.78
34	20	58	7	31	149.38
35	20	58	8	28	1027.81
36	20	58	9	21	1047.95
37	20	3	2	37	1647.30
38	20	3	5	34	57.92
39	20	11	1	22	41.45
40	20	11	2	20	145.17
41	20	11	3	26	194.65
42	20	11	4	31	1048.21
43	20	11	5	24	23.37
44	20	11	6	31	30.31
45	20	11	7	27	2594.34
46	20	11	8	20	472.98
47	20	11	9	26	3148.34
48	20	11	10	27	36.47
49	20	19	1	25	21.30
50	20	19	3	28	693.46
51	20	19	4	22	110.02
52	20	19	5	28	204.17
53	20	19	6	35	34.39
54	20	19	7	30	29.07
55	20	19	8	22	26.02
56	20	19	9	22	108.00
57	20	19	10	33	510.88
58	20	27	1	19	31.08
59	20	27	2	23	19.84
60	20	27	3	25	86.27
61	20	27	4	25	18.30
62	20	27	6	24	40.78
63	20	27	7	26	6.39
64	20	27	8	22	17.37
65	20	27	9	22	6.15
66	20	27	10	21	16.57
67	20	35	1	35	164.64
68	20	35	2	30	1430.51
69	20	35	3	31	147.32
70	20	35	4	36	1562.33
71	20	35	5	32	240.88
72	20	35	6	32	127.79
73	20	35	7	33	3898.55
74	20	35	8	29	428.60
75	20	35	9	35	74.45
76	20	35	10	34	546.55
77	20	43	1	34	247.98
78	20	43	2	32	1595.69
79	20	43	3	32	131.47
80	20	43	5	30	69.47
81	20	43	6	30	234.29

Following the J20 on the next page...

PSPLIB caracteristics				Optimal solution	
Number	Tasks	Parametre	Instance	Makespan	CPU (sec)
continue...					
82	20	43	8	28	90.28
83	20	43	9	27	76.74
84	20	43	10	30	57.28
85	20	51	1	20	39.61
86	20	51	2	25	25.54
87	20	51	3	23	326.06
88	20	51	4	25	1796.03
89	20	51	5	31	32.16
90	20	51	6	24	62.17
91	20	51	7	23	572.33
92	20	51	8	27	632.26
93	20	51	9	24	57.20
94	20	51	10	21	12.48
95	20	59	1	26	52.21
96	20	59	2	25	9.07
97	20	59	3	28	539.80
98	20	59	4	22	15.29
99	20	59	5	30	20.54
100	20	59	6	32	6.49
101	20	59	7	21	21.96
102	20	59	8	25	31.87
103	20	59	9	30	77.20
104	20	59	10	27	92.06
105	20	12	1	21	179.66
106	20	12	2	20	23.64
107	20	12	3	34	27.43
108	20	12	4	19	23.66
109	20	12	5	23	26.00
110	20	12	6	20	13.71
111	20	12	7	21	12.83
112	20	12	8	28	21.44
113	20	12	10	21	31.57
114	20	20	1	26	31.79
115	20	20	2	29	20.09
116	20	20	3	23	43.11
117	20	20	4	23	7.93
118	20	20	5	21	20.93
119	20	20	6	21	4.62
120	20	20	7	25	6.23
121	20	20	8	17	4.49
122	20	20	9	34	25.41
123	20	20	10	21	15.06
124	20	28	1	22	7.36
125	20	28	2	24	11.78
126	20	28	3	18	13.67
127	20	28	4	23	16.57
128	20	28	5	26	9.29
129	20	28	6	18	16.69
130	20	28	7	25	12.84
131	20	28	8	22	62.44
132	20	28	9	19	4.50

Following the J20 on the next page...

PSPLIB caracteristics				Optimal solution	
Number	Tasks	Parametre	Instance	Makespan	CPU (sec)
continue...					
133	20	28	10	25	12.60
134	20	44	1	25	243.21
135	20	44	2	29	251.46
136	20	44	3	30	534.78
137	20	44	4	30	46.85
138	20	44	5	26	21.91
139	20	44	6	24	30.72
140	20	44	7	29	44.62
141	20	44	8	26	111.53
142	20	44	9	29	23.88
143	20	44	10	26	21.79
144	20	52	1	20	23.12
145	20	52	2	24	62.53
146	20	52	3	25	30.23
147	20	52	4	21	56.47
148	20	52	5	23	28.35
149	20	52	6	20	22.58
150	20	52	7	27	60.84
151	20	52	8	27	34.79
152	20	52	9	19	24.97
153	20	52	10	25	22.47
154	20	60	1	21	299.24
155	20	60	2	22	5.73
156	20	60	3	23	7.23
157	20	60	4	24	5.98
158	20	60	5	21	13.98
159	20	60	6	23	5.61
160	20	60	7	28	13.32
161	20	60	8	16	15.19
162	20	60	9	20	17.09
163	20	60	10	30	11.18
164	20	16	7	25	13446.81
165	20	24	7	22	137.98
166	20	32	8	32	116.32
167	20	48	3	25	3380.33
168	20	56	10	25	17598.78
169	20	64	5	29	4440.46
J20 is finished..					

PSPLIB caracteristics				Optimal solution	
Number	Tasks	Parametre	Instance	Makespan	CPU (sec)
1	30	10	1	29	29028,19
2	30	10	6	26	753,02
3	30	10	9	40	2917,04
4	30	26	4	36	426,42
5	30	26	8	40	129,46
6	30	26	10	24	1206,29
7	30	58	1	33	9554,75
8	30	58	5	29	8893,747
9	30	11	1	35	23,43
Following the J30 on the next page...					

PSPLIB caracteristics				Optimal solution	
Number	Tasks	Parameter	Instance	Makespan	CPU (sec)
continue...					
10	30	11	3	31	164,46
11	30	11	5	35	405,75
12	30	11	6	33	150,13
13	30	11	7	32	122,69
14	30	11	8	30	93,94
15	30	11	10	35	123,52
16	30	19	2	23	162,2
17	30	19	3	32	318,56
18	30	19	4	30	288,91
19	30	19	5	29	1078,37
20	30	19	6	24	139,18
21	30	19	7	36	116,91
22	30	19	8	34	269,91
23	30	19	9	28	245,54
24	30	19	10	30	186,93
25	30	27	1	38	71,16
26	30	27	2	31	260,97
27	30	27	3	28	86,99
28	30	27	4	28	62,22
29	30	27	5	40	153,93
30	30	27	6	26	437,72
31	30	27	7	30	82,26
32	30	27	8	42	168,47
33	30	27	9	38	4768,16
34	30	27	10	29	70,34
35	30	35	3	39	1013,28
36	30	35	4	48	3332,08
37	30	35	5	42	1292,47
38	30	35	6	41	102,56
39	30	35	7	35	704,72
40	30	35	8	41	649,49
41	30	35	9	36	1776,97
42	30	35	10	51	4771,79
43	30	43	1	30	3786,23
44	30	43	5	44	221,136
45	30	43	10	32	6065,22
46	30	51	1	32	489,568
47	30	51	2	24	1140,518
48	30	51	3	27	255,835
49	30	51	4	51	174,323
50	30	51	5	33	243,796
51	30	51	6	38	607,564
52	30	51	7	26	191,588
53	30	51	8	35	2340,948
54	30	51	10	27	196,137
55	30	59	2	23	144,68
56	30	59	3	35	351,28
57	30	59	4	25	167,442
58	30	59	5	28	94,739
59	30	59	6	27	303,95
60	30	59	7	34	167,087

Following the J30 on the next page...

PSPLIB caracteristics				Optimal solution	
Number	Tasks	Parameter	Instance	Makespan	CPU (sec)
continue...					
61	30	59	8	28	245,918
62	30	59	10	30	288,828
63	30	12	1	29	131,52
64	30	12	2	31	91,95
65	30	12	3	34	93,84
66	30	12	4	29	114,62
67	30	12	5	25	88,26
68	30	12	6	34	135,69
69	30	12	7	27	103,94
70	30	12	8	28	143,21
71	30	12	9	22	27,42
72	30	12	10	28	210,28
73	30	20	1	30	103,41
74	30	20	2	25	75,1
75	30	20	3	27	38,07
76	30	20	4	35	113,2
77	30	20	5	29	78,93
78	30	20	6	24	124,28
79	30	20	7	31	69,1
80	30	20	8	33	88,11
81	30	20	9	26	84,48
82	30	20	10	28	87,16
83	30	28	1	37	47
84	30	28	2	25	73,38
85	30	28	3	31	68,33
86	30	28	4	33	98,8
87	30	28	5	23	14,9
88	30	28	6	29	23,84
89	30	28	7	37	67,16
90	30	28	8	36	44,24
91	30	28	9	28	98,74
92	30	28	10	26	70,98
93	30	44	1	41	78,44
94	30	44	2	28	1877,85
95	30	44	3	34	256,93
96	30	44	4	33	722,652
97	30	44	5	29	605,86
98	30	44	6	25	956,816
99	30	44	7	36	1215,438
100	30	44	8	36	238,235
101	30	44	9	35	366,28
102	30	44	10	33	20303,94
103	30	52	1	32	125,46
104	30	52	2	23	652,563
105	30	52	3	23	321,296
106	30	52	4	30	120,64
107	30	52	5	30	205,426
108	30	52	6	30	197,799
109	30	52	7	30	155,619
110	30	52	8	26	233,727
111	30	52	9	31	186,671

Following the J30 on the next page...

PSPLIB caracteristics				Optimal solution	
Number	Tasks	Parameter	Instance	Makespan	CPU (sec)
continue...					
112	30	52	10	29	172,339
113	30	60	1	22	93,34
114	30	60	2	38	120,56
115	30	60	3	40	926,702
116	30	60	4	23	134,162
117	30	60	5	34	92,482
118	30	60	6	36	21,777
119	30	60	7	25	130,217
120	30	60	8	29	25,524
121	30	60	9	31	5418,226
122	30	60	10	23	87,764
123	30	8	6	46	696,96
124	30	16	2	41	908,94
J30 is finished.					

Annexe E

Article intitulé «*The Generalized Resource Allocation and Leveling Problem in project scheduling*». La version incluse dans cette thèse a été arbitrée et acceptée par le comité d'évaluation et publiée dans les actes de la conférence internationale PMS 2016 (*Project Management and Scheduling*, ISBN :978-3-00-045630-5)

Resource Allocation and Levelling in multi-mode project scheduling

Fayez F. Boctor and Roubila Lilia Kadri
Centre interuniversitaire de recherche sur les réseaux d'entreprises, la logistique et le transport (*CIRRELT*)
Operations and Decision Systems Department, Université Laval, Quebec, Canada
Fayez.Boctor@fsa.ulaval.ca and lilia.kadri.1@ulaval.ca

Keywords: Project Scheduling, Resource Allocation, Resource levelling, Heuristics.

1. Abstract

Many research publications dealt with the multi-mode project scheduling problem under the constraint of limited resource availability. Some others dealt with the problem where it is required to keep the resource utilization level as invariable as possible all over the project execution period. In real-life we need to simultaneously determine the amount of resources to allocate to the project during its execution and to reduce the resources utilisation variability to the minimum while trying to finish the project by an acceptable completion date. The amount of resources to allocate to the project should allow finishing the project by this date and becomes a limit on the availability of these resources at any time period.

In this paper we consider this more realistic project scheduling problem where, given an upper limit on the amount of resources that can be made available to the project and an upper limit on the project completion time, we have to determine the amount of resources to allocate to its execution, the completion date to propose to the client while minimizing the project total execution cost. The execution cost is composed of two main cost elements: the direct cost of resources to use and the overhead cost. The overhead cost does not depend on the amount of allocated resources but is directly proportional to the project duration. To the best of our knowledge this problem, we call the *Generalized Resource Allocation and Levelling Problem* (GRALP), has never been the subject of any publication before. The paper proposes a mathematical formulation of the problem as well as a neighbourhood search heuristic to solve it. It also presents a numerical experiment to assess the performance of the proposed heuristic.

2. Problem description

In the GRALP we have a project composed of a set of tasks $i = 1, ..., N$, a set P of zero-lag, finish-to-start precedence relations defined over the set of tasks, and a set K of renewable resources. There is a limit on the amount of resource k that can be made available to the execution of the project, denoted M_k; $\forall\ k \in K$. However we need to determine R_k ($R_k \leq M_k$) the exact amount of resource k that should be allocated to the project. Each activity i has a number m_i of possible execution modes, where each mode j is characterized by its resource requirements, denoted r_{ijk}, and the corresponding duration, denoted d_{ij}. We also define E_i and L_i, the earliest and latest start time of activity i, and T the finish date of the project. These times are calculated using the critical path method while using the maximal acceptable project duration, denoted H ($H \geq T$). We assume that the direct cost of making a unit of resource k available to the project at period t is known and denoted c_{kt}. In addition, there is an

availability costs for periods T+1 up to the end of the project. Constraints (2) determine the execution mode and the finish period for each task. It also make sure that each task has one end only one execution mode, as well as one and only one finish period t. Constraints (3) assure that any task cannot start before the end of the execution of its immediate predecessors and constraints (4) determine the amount of resources to be allocated to carry out the project. Constraints (5) ensure that the allocated amounts of resources do not exceed what can be made available and constraints (6) make sure that no resources are allocated once the project is completed. Constraints (7) ensure that y_t takes the value one (the project is in execution) if there are any tasks to execute at period t and constraints (8) imply that if the project is not in course of execution at period t-1, it is not in period t.

4. The GRALP versus the RACP

The resource availability cost problem (RACP) is the problem of determining the project duration and the quantity of resources to allocate to a project while minimizing a weighted function of these quantities (Mohring 1984 and Demeulemeester 1995). The weights used, denoted c_k, are independent of the project duration. The constraints considered in this problem are the precedence constraints, resource availability constraints and the duration of the project that should be less than or equal to a given maximal acceptable duration. This problem is also known as the resource investment problem (RIP; see Hsu and Kim; 2005).

This problem is different from the generalized resource allocation and levelling problem (GRALP), introduced in the paper, as the function to minimize in the RACP is not the project total cost. Using the above presented notation, the RACP objective is to minimize $\sum_k c_k R_k$. We notice that this objective function does not include the overhead cost. Even the function does not express the direct cost of the allocated resources over the whole project execution period as c_k is independent of the project duration.

5. The proposed neighbourhood search heuristic

The heuristic approach we propose to solve the GRALP is a 3-dimension neighbourhood search approach where each explored neighbourhood is related to one of three decisions to make. The first dimension or decision, the number of units of resource type k to allocate to the project, is coded by a vector of K elements where the k^{th} element gives the number of units of resource type k to allocate to the project. The second dimension or decision is coded by a vector of N elements where the i^{th} element indicates the mode to be used to execute activity or task i. Finally, the execution schedule is coded by a vector of N elements indicating the sequence in which activities should be considered for scheduling. This sequence can be translated into a schedule by assigning to each activity (while using the preselected execution mode) the earliest possible execution date which satisfies resource limits and precedence relations. The way to transform a sequence into a schedule is straightforward and described in Boctor (1996). The general framework of the proposed neighborhood search approach is depicted in Figure 1. The details of the embedded sub-procedures (in bold face characters in Figure 1) cannot be given in this extended abstract but will be given in the full paper.

overhead cost f_t for the execution of the project at period t. This overhead cost is not dependant on the amount of resources allocated to the project. The objective is to determine the amount of resources to allocate to the project, to select for each task an execution mode, and to assign to each task its execution dates in order to minimize the overall project execution cost while completing it by the maximal acceptable completion date H.

3. Mathematical formulation

Before presenting the proposed mathematical formulation let us first present the decision variables used to model the GRALP:

Decision variables:

R_k number of units of resource type k to allocate to the project over its execution time

x_{ijt} binary variable that takes the value 1 if and only if task i is executed using mode j and finished at the end of period t

y_t binary variable that takes the value 1 if the project is in execution at period t

R_{kt} a variable that takes the value R_k if the project is in execution at period t, and zero otherwise.

It is obvious that y_t should take the value 1 for all periods $t \leq T$. So the model seeks to determine the value of y_t only for $t = T + 1$, ..., H. Similarly, the variable x_{ijt} should take the value zero for all periods $t < E_i + d_{ij} - d_{i1}$ or $t > L_i$. So the model seeks to determine the value of x_{ijt} for periods t between $E_i + d_{ij} - d_{i1}$ and L_i. Also, let A_t be the set of tasks that could be in execution at t; thus $A_t = \{i | E_i - d_{i1} < t \leq L_i\}$.

The model:

Find: $x_{ijt}; i = 1, \dots, N, j = 1, \dots, m_i, t = E_i + d_{ij} - d_{i1}, \dots, L_i$,

$R_k; k = 1, \dots, K$,

$R_{kt}; k = 1, \dots, K, t = T + 1, \dots, H$, and

$y_t \in \{0,1\}; t = T + 1, \dots, H$; which:

Minimize: $\sum_{t=1}^{T} f_t + \sum_{t=T+1}^{H} f_t y_t + \sum_{k=1}^{K} \sum_{t=1}^{T} c_{kt} R_k + \sum_{k=1}^{K} \sum_{t=T+1}^{H} c_{kt} R_{kt}$ (1)

Subject to: $\sum_{j=1}^{m_i} \sum_{t=E_i+d_{ij}-d_{i1}}^{L_i} x_{ijt} = 1; \quad i = 1, \dots, N$ (2)

$$\sum_{j=1}^{m_e} \sum_{t=E_e+d_{ej}-d_{e1}}^{L_e} t x_{ejt} - \sum_{j=1}^{m_i} \sum_{t=E_i+d_{ij}-d_{i1}}^{L_i} (t - d_{ij}) x_{ijt} \leq 0; \quad i = 1, \dots, N, e \in P_i \quad (3)$$

$$\sum_{i \in A_t} \sum_{j=1}^{m_i} \sum_{s=\max\{t, E_i+d_{ij}-d_{i1}\}}^{\min\{t+d_{ij}-1, L_i\}} r_{ijk} x_{ijs} \leq R_k; \quad k = 1, \dots, K, t = 1, \dots, H \quad (4)$$

$$R_k \leq M_k; \quad k = 1, \dots, K \quad (5)$$

$$R_{kt} \geq R_k - M_k(1 - y_t); \quad k = 1, \dots, K; t = T + 1, \dots, H \quad (6)$$

$$N y_t \geq \sum_{i \in A_t} \sum_{j=1}^{m_i} \sum_{s=\max\{t, E_i+d_{ij}-d_{i1}\}}^{\min\{t+d_{ij}-1, L_i\}} x_{ijs}; \quad t = T + 1, \dots, H \quad (7)$$

$$y_t \leq y_{t-1}; \quad t = T + 2, \dots, H \quad (8)$$

The objective function (1) expresses the total cost of the project. The first and third terms express respectively the overhead and the direct resources availability costs of the T first periods; the second and forth terms give the overhead and resources

Optimal solutions

Optimal solutions were obtained by solving the mathematical formulation given in section 3 by the MIP commercial code GUROBI 5.0.1 on a computer equipped with a 2 GHz, i7 core processor. The average computation time is 5957 seconds with a minimum time of 202 seconds, a maximum time of 63 626 seconds, and a standard deviation of 10 934 seconds.

Heuristic solutions

Table 2 gives the percentage deviation from the optimal solution for the 90 instances of the GRALP. From this table we can see that the proposed heuristic obtained an average deviation from the optimum of 8.85 %.

Table 2: Computational results for the test-instances

Average percentage deviation from the optimum	8.85 %
Minimum percentage deviation from the optimum	1.81 %
Maximum percentage deviation from the optimum	19.57 %
Standard deviation	3.82 %
Average computational time (sec)	54.6

7. Conclusion

In this paper we introduce the *Generalized Resource Allocation and Levelling Problem* (GRALP), a new project scheduling problem that is closer to real-life project scheduling problems. In this problem we need to determine: (1) the number of units of resources to allocate to the project, (2) the execution mode to use for each task, and (3) the execution dates for these tasks. Our constraints are the precedence constraints, the maximum amount of resources that can be allocated to the project and an upper limit on the project duration. Our objective is to minimize the sum of the resource availability cost and the overhead cost.

To the best of our knowledge, the GRALP have never been the subject of any publication and consequently we do not have any previously published method to solve it. The research done in this research work seems to indicate that we are facing a difficult problem as it was not possible to design a heuristic capable to produce an average percentage deviation from the optimum of better than 8.85 %. Certainly we need to pursue our research effort to improve our solution tools. It is obvious that we need to find an approach to solve larger problems to optimality and to develop more efficient heuristics.

References

Boctor F. F. 1996. A new and efficient heuristic for scheduling projects with resource restrictions and multiple execution modes. *European Journal of Operational Research*. 90: 349–361.

Demeulemeester E. 1995. Minimizing resource availability cost in time-limited project networks. *Management Science*. 41: 1590-1598.

Hsu C. C. and Kim D. S. 2005. A new heuristic for the multi-mode resource investment problem. *Journal of the operational research society*. 56: 406-413.

Mohring R 1984. Minimizing costs of resource requirements in project networks subject to fixed completion time. *Operations Research*. 32: 89-120.

Annexe F : Solutions optimales des instances-tests du PGANRP

Set 1	CPM	Makespan	R_1	R_2	Optimal cost	CPU (sec)
1	28	30	13	11	3450	13263
2	18	18	20	19	3456	448.21
3	20	21	13	18	3570	377.88
4	26	27	16	16	4266	7810
5	22	23	22	17	4232	7360
6	22	22	20	16	3696	280
7	25	26	18	15	4134	1331
8	25	30	17	10	3600	2730
9	26	33	11	13	4488	6041
10	29	35	13	10	3955	906
11	32	37	11	12	4366	1202
12	25	25	16	20	4525	1750
13	16	19	21	17	3743	1385
14	20	22	16	17	3872	1360
15	38	39	14	13	5109	20655.701
16	16	16	27	24	3872	202.151
17	19	22	17	10	2816	1305
18	22	22	21	17	3982	9151.49
19	26	28	15	16	4396	2355.32
20	29	38	9	9	3648	1522.43
21	19	21	14	16	3444	634.784
22	25	26	15	13	3588	397.102
23	22	26	13	11	3302	1347.267
24	22	26	14	16	4446	457.656
25	24	26	15	13	3588	9253.314
26	15	15	26	22	3435	828.029
27	24	28	17	12	4004	1633.782
28	26	27	15	15	4158	284.743
29	21	24	17	14	3672	1397.904
30	22	23	14	13	3197	457.672

Set 2	CPM	Makespan	R_1	R_2	Optimal cost	CPU (sec)
1	28	35	11	9	3535	3406.25
2	18	23	15	13	3496	883.56
3	20	24	11	15	3624	1455.388
4	26	30	14	14	4320	10133.3
5	22	26	19	14	4238	10273.04
6	22	25	17	14	3800	1123.497
7	25	31	15	12	4278	7404.849
8	25	34	14	9	3706	4249.436
9	26	36	9	12	4572	6374.082
10	29	40	12	8	4040	1040.91
11	32	42	10	11	4662	5529.713
12	25	28	12	18	4564	1971.027
13	16	22	17	14	3828	3018.882
14	20	26	13	14	4030	3682.497
15	38	42	12	12	5124	11443.47
16	16	19	23	19	3971	1065.434
17	19	26	14	8	2912	1957.474
18	22	25	18	14	4000	3734.978
19	26	34	14	11	4420	2551.036
20	29	38	9	9	3648	410.881
21	19	23	12	14	3450	1297.989
22	25	28	14	12	3668	2040.616
23	22	29	12	9	3335	2557.381
24	22	30	12	14	4710	5039.919
25	24	29	13	11	3596	7872.856
26	15	17	21	19	3468	955.612
27	24	33	13	10	4125	1368.243
28	26	29	14	14	4263	4001.426
29	21	28	14	11	3696	2466.706
30	22	25	13	12	3300	1250.105

Set 3	CPM	Makespan	R_1	R_2	Optimal cost	CPU (sec)
1	28	35	11	9	3150	26871.1
2	18	23	15	13	3059	6160.085
3	20	24	11	15	3192	2946.777
4	26	34	11	12	3842	25245.27
5	22	28	16	13	3752	54510.358
6	22	25	17	14	3400	1864.909
7	25	31	15	12	3782	6747.412
8	25	34	14	9	3298	22323
9	26	36	9	12	3996	5998.326
10	29	40	12	8	3560	1411.823
11	32	50	9	8	4100	6053.688
12	25	31	12	15	4092	5539.662
13	16	22	17	14	3322	4695.055
14	20	27	13	13	3510	2841.694
15	38	47	11	10	4559	25850.38
16	16	19	23	19	3534	1108.698
17	19	26	14	8	2522	2791.066
18	22	25	18	14	3550	4858.15
19	26	36	13	10	3852	3867.289
20	29	38	9	9	3230	1178.456
21	19	23	12	14	3013	1169.105
22	25	28	14	12	3276	2723
23	22	29	12	9	2900	2291.915
24	22	30	12	14	4080	15272.35
25	24	32	10	10	3168	14116.35
26	15	17	21	19	3094	958.409
27	24	33	13	10	3597	7030.067
28	26	29	14	14	3799	1892.035
29	21	28	14	11	3248	3992.566
30	22	26	13	11	2912	690.56