UNIVERSITÉ DE MONTRÉAL

GESTION DE PROJET AVEC CONTRAINTES DE RESSOURCES

OLIVIER MILON
DÉPARTEMENT DE MATHÉMATIQUES
ET DE GÉNIE INDUSTRIEL
ÉCOLE POLYTECHNIQUE DE MONTRÉAL

MÉMOIRE PRÉSENTÉ EN VUE DE L'OBTENTION
DU DIPLÔME DE MAÎTRISE ÈS SCIENCES APPLIQUÉES
(MATHÉMATIQUES APPLIQUÉES)
AOÛT 1999

© Olivier Milon, 1999.

National Library of Canada
Bibliothèque nationale du Canada

Acquisitions and Bibliographic Services
Acquisitions et services bibliographiques

395 Wellington Street
Ottawa ON K1A 0N4
Canada

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence

Our file Notre référence

The author has granted a non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of this thesis in microform, paper or electronic formats.

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de cette thèse sous la forme de microfiche/film, de reproduction sur papier ou sur format électronique.

The author retains ownership of the copyright in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

L'auteur conserve la propriété du droit d'auteur qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

0-612-54038-3

Canada

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Ce mémoire intitulé:

GESTION DE PROJET
AVEC
CONTRAINTES DE RESSOURCES

présenté par : MILON Olivier
en vue de l'obtention du diplôme de: Maîtrise ès sciences appliquées
a été dûment accepté par le jury d'examen constitué de:

M. LANGEVIN André, Ph. D., président du jury
M. SOUMIS François, Ph.D., membre et directeur de recherche
M. TACHEFINE Beyime, Ph.D., membre et codirecteur de recherche
Mme. LAPIERRE Sophie, Ph.D., membre

A Suzanne et Yanie

REMERCIEMENTS

Je tiens à remercier premièrement, mon directeur de recherche **M. François SOUMIS** pour l'accueil qu'il m'a réservé au sein du laboratoire, la confiance qu'il m'a accordée et sa disponibilité dont j'ai pu profiter tout au long de ce projet ; deuxièmement, mon codirecteur de recherche **M. Beyime TACHEFINE** pour sa sympathie et son soutien professionnel. Leurs conseils et directives de recherche m'ont permis d'avancer de façon toujours constructive et rapide, tout en conservant une certaine autonomie dans mon travail.

Je remercie **M. André LANGEVIN** de bien vouloir présider mon jury, ainsi que **Mme Sophie LAPIERRE** d'avoir accepté d'être membre de ce jury.

Je voudrais souligner également l'aide précieuse de certains membres et professionnels du GERAD qui m'ont aidé dans mon travail, je pense à **Norbert LINGAYA**, **Daniel VILLENEUVE**, et l'équipe Gencol en général.

Merci enfin à **Mme Sylvie GÉLINAS** pour ses travaux qui m'ont largement inspiré et à **M. Edouard WAGNEUR** pour m'avoir fait connaître le GERAD et permis de venir au Québec.

RÉSUMÉ

Ce mémoire présente une nouvelle méthode exacte pour résoudre les problèmes de gestion de projet avec fenêtres de temps et contraintes de ressources. Nous avons formulé, mis en place et testé cette méthode puis nous y avons apporté des améliorations pour la rendre plus efficace. La méthode de calcul procède par génération de colonnes avec un sous-problème qui génère des colonnes (ou horaires) en respectant les contraintes de préséances, de durées des tâches, et de fenêtres de temps pour les fournir au problème maître qui résout la relaxation linéaire du problème avec les contraintes de ressources.

Le problème maître relaxé est résolu par la méthode primale du simplexe. Le sous-problème a été transformé en un problème de flot maximum sur un graphe. Nous le résolvons par un algorithme de pré-flot très rapide de complexité $O(n^2\sqrt{m})$, où m est le nombre d'arcs et n le nombre de nœuds du graphe. La solution finale en nombres entiers (0 ou 1) est obtenue grâce à un branchement ou recherche arborescente sur les fenêtres de temps de chaque activité.

Dans ce mémoire, nous présentons l'état de l'art, le modèle mathématique, l'algorithme de flot maximum et la mise en pratique de la méthode de génération de colonne. Les tests numériques de notre méthode ont été obtenus après avoir premièrement reformulé le problème de flot pour qu'il puisse être traîté par l'algorithme de pré-flot. Nous avons, deuxièmement, développé un code informatique en langage C de l'algorithme de base de notre méthode. Nous avons fondé notre expérimentation sur des problèmes disponibles dans la littérature et reconnus comme des références en termes de jeux d'essais pour le problème de gestion de projet avec contraintes de ressources avec pour objectif la minimisation de la durée du projet.

Le programme linéaire fournit une solution en nombres réels de ces problèmes qui est une borne inférieure du problème en nombres entiers. Cette borne étant

assez éloignée de la solution optimale dans près de la moitié des cas, nous avons proposé, mis en place et testé plusieurs perfectionnements de la méthode de base afin d'améliorer la qualité de la borne inférieure. Afin de réduire la taille du graphe du sous-problème, nous avons ajouté un programme de calcul de borne supérieure trouvé dans la littérature ([28]).

En observant les itérations du problème maître, nous avons mis au point un système d'accélération des itérations. Le nombre de variables du problème est divisé par trois en moyenne grâce aux bornes inférieure et supérieure, et le nombre d'itérations est divisé par quatre avec le système d'accélération. Nous avons pu résoudre des problèmes de 30 activités et 4 ressources en 8.1 secondes avec 16 nœuds de branchements et 137 itérations, et avons proposé différentes techniques de branchement ainsi qu'une méthode de coupe.

ABSTRACT

This report is a presentation of a new exact method to solve resource constrained project scheduling problems. We have formulated, implemented, and tested this method, and we have added enhancements to make it better. The solving method uses a column generation iterative process between a sub-problem that generates columns (or schedule) under precedence, processing time, and time windows constraints, and a master problem that solves the linear relaxation of the problem with resources constraints. The relaxed master problem is solved with the primal simplex method and the sub-problem is solved as a maximum flow problem. The latter is solved with a very fast pre-flow algorithm whose complexity is $O(n^2\sqrt{m})$, where m is the number of arcs and n the number of nodes of the network. The final solution is an integer solution (0 or 1) obtained with to a branch and bound procedure applied on the activities time windows.

We first started our work following four steps which are the state of the art, the mathematical model, the maximum flow algorithm and the column generation method. We then formulated the sub-problem in order to treat it with the pre-flow algorithm. Finally, we implemented the algorithm with the C language to test the method as described by the mathematical formulation. We tested this basic method with several problems that we found in the literature. These problems are a reference for the resource constrained project scheduling problems with the objective of minimizing the total duration of the project.

The linear program gives a real solution to the mathematical problem which is a lower bound of the integer solution. Since this bound is often very far from the optimal solution, we have proposed, implemented and tested several enhancements of the method in order to find a tighter lower bound. We have also added a program that we found in the literature to compute an upper bound to reduce the size of the graph of the sub-problem [28].

The analysis of the iterative mechanism conducted us to develop a treatment to accelerate the iterative process between the master problem and the sub-problem.

The average problem size is reduced by three thanks to the new bounds, and the number of iterations is divided by four when the accelerating system is used. We solved problems of 30 activities and 4 resources in 8.1 seconds using 16 nodes on the tree search, and 137 iterations. We present and test different techniques to make the branching decision and also a method of branch cutting.

Table des matières

DÉDICACE iv

REMERCIEMENTS v

RÉSUMÉ vi

ABSTRACT viii

TABLES DES MATIÈRES x

LISTES DES TABLEAUX xiv

LISTES DES FIGURES xvi

CHAPITRE 1 : Introduction **1**

CHAPITRE 2 : Présentation du problème **4**

2.1 La modélisation du problème 4

2.1.1 Les données initiales du problème 4

2.1.2 Le diagramme potentiel-tâche 5

2.1.3 Définition d'un ordonnancement 6

2.1.4 Le diagramme de GANTT & les profils des ressources 7

CHAPITRE 3 : Revue de Littérature **10**

3.1 Les méthodes exactes 11

3.1.1 La programmation linéaire en nombres entiers 11

3.1.2 La recherche arborescente 13

3.1.3 La décomposition du problème 17

3.2 Les méthodes approchées 19

3.2.1 Les règles de décisions 20

3.3 Bilan de la revue de littérature 21

CHAPITRE 4 : Méthode et outils **22**

4.1 Présentation du modèle mathématique 25

4.1.1 Définition du programme linéaire en nombres entiers 26

4.1.2 Décomposition de Dantzig-Wolfe 28

4.1.3 Reformulation du sous-problème 31

4.2 Construction et résolution du sous-problème 35

4.2.1 Utilisation du sous-problème 35

4.2.2 Construction du réseau 36

4.2.3 Modélisation de la date de fin du projet 37

4.2.4 Les algorithmes de flot maximum 38

4.3 Construction et résolution du problème maître 40

4.3.1 Méthode de résolution 40

4.3.2 Améliorations de la résolution 42

4.4 Description de la méthode de branchement 43

4.4.1 La décision de branchement 43

4.4.2 La progression dans l'arbre 44

4.5 Les bornes inférieures 45

4.5.1 Démonstration de la validité de la méthode 46

4.5.2 Implémentation 48

4.5.3 Tests numériques 48

4.5.4 Extension de la méthode 50

4.5.5 Proposition : calcul des fenêtres avec les bornes LB1 et LB2 . 51

4.5.6 Essais numériques de la borne LB1 57

4.6 Les bornes supérieures 58

4.6.1 Heuristique pour calculer la borne supérieure 58

4.6.2 Une méthode de coupe :
calcul dynamique de la borne supérieure 61

4.7 Diminution du nombre d'itérations 63

4.7.1 Exploitation du problème maître 63

4.7.2 Génération de colonne :
combinaisons de points intérieurs 73

4.7.3 Stabilisation des coûts réduits 82

CHAPITRE 5 : Résultats numériques 89

5.1 La génération des problèmes 89

5.1.1 Les problèmes types de Patterson 89

5.1.2 Les problèmes de "KSD" 90

5.2 Tests effectués avec les problèmes de KSD 92

5.2.1 Performances et bornes 92

5.2.2 Résultats des tests sur la relaxation linéaire 94

5.2.3 Résultats des tests sur le branchement 103

5.2.4 Résultats des tests sur d'autres fonctions objectifs 107

CHAPITRE 6 : Conclusion 110

BIBLIOGRAPHIE 114

Liste des tableaux

2.1 *Données initiales d'un projet* 5

4.1 *Résultats des tests avec et sans perturbations* 70

4.2 *Résultats de la méthode de base* 74

4.3 *Résultats de la méthode avec l'analyse de la fermeture* 74

4.4 *Résultats de la méthode avec analyse de fermeture restreinte* 76

4.5 *Résultats de la méthode avec analyse de fermeture très restreinte* . . . 77

4.6 *Itérations et temps de calcul avec et sans heuristique* 79

4.7 *Résultats de la méthode avec heuristique* 80

4.8 *Proportion des colonnes alternatives dans la solution linéaire* 81

5.1 *Types de problèmes et temps de caculs* 92

5.2 *Description des versions de la méthode* 95

5.3 *Description des versions de la méthode (bis)* 96

5.4 *Description des versions de la méthode (ter)* 97

5.5 *Résultats des versions V0 à V17* 99

5.6 *Influence de la taille du projet* 103

5.7 *Influence de la taille du projet(bis)* 103

5.8 *Influence de la taille du projet et des ressources* 104

5.9 *Résultats des versions de branchement* 105

5.10 *Résultats de la méthode avec minimisation des retards* 107

Table des figures

2.1 *Diagramme potentiel-tâche* 6

2.2 Figure 7

2.3 *Problème de gestion de projet avec contraintes de ressources* 8

4.1 *Algorithme de résolution* 24

4.2 *Amélioration de la borne inférieure* 49

4.3 *Exemple de calcul de borne inférieure* 56

4.4 *Performance de la borne inférieure LB1* 57

4.5 *Evolution des coûts réduits avec la méthode de base* 83

4.6 *Evolution des coûts réduits avec la méthode de pondération* 85

4.7 *Influence de ρ sur le nombre d'itérations* 87

5.1 *Comportement des bornes* 93

Chapitre 1

Introduction

Nous nous intéressons au problème de gestion de projet avec contraintes de ressources et fenêtres de temps, encore appelé dans la littérature "problème généralisé de gestion de projet avec contraintes de ressources" (Generalized Resource Constrained Project Scheduling Problem). Certains articles parlent également de problème de gestion de projet à multi-ressources continues.

Un "projet" est un ensemble d'activités ou de tâches qui doivent toutes être exécutées une fois, suivant un certain ordre pour rendre le "projet" réalisable. Cet ordre n'est en général pas unique mais il est partiellement défini par les contraintes sur les activités. Ces activités doivent en effet respecter :

- un ordre chronologique d'éxécution ou préséance,
- optionnellement, une date de début au plus tôt et une date de fin au plus tard ou fenêtre de temps,
- une durée déterminée.

La définition des fenêtres de temps peut être omise. Elles sont prises dans ce cas par défaut en concordance avec les relations de préséances pour réduire la taille du problème. La "gestion de projet" consiste à définir des dates de début pour l'exécution des activités du projet de manière à respecter les contraintes sur les activités.

Les "contraintes de ressources" sont des contraintes globales portant sur le total de consommation des ressources à chaque instant par les activités en cours d'exécution. Les ressources possèdent les propriétés suivantes :

- il y a plusieurs types de ressources,
- chaque activité consomme une partie ou tous les types de ressources,
- la consommation par unité de temps de chaque type de ressource par chaque activité du projet est connue et constante durant l'éxécution de l'activité,
- la disponibilité des ressources est constante dans le temps,
- les ressources sont renouvelables dans le temps.

Ces données initiales sont généralement des entiers naturels positifs ; certaines méthodes employent des ressources binaires (entiers égal à 0 ou 1).

Le problème est de trouver un horaire réalisable qui minimise une fonction objectif. La fonction objectif peut être la date de fin du projet ou bien toute fonction des dates de début des activités du projet. Les essais numériques portent sur la minimisation de la durée du projet.

Le problème peut être résumé de la manière suivante : *Etant donné une liste d'activités de durées connues et connaissant leurs consommations des différents types de ressources disponibles en quantités limitées, quel est l'horaire qui minimise une fonction des dates de début des activités, tout en respectant la préséance, les fenêtres de temps des activités et les limites de consommations de ressources ?*

Ce problème est NP-difficile (Lenstra [30]).

Ce mémoire se divise en 6 parties. Après cette introduction, nous présentons le problème à l'aide des modèles déjà existants pour le problème sans contraintes de ressources et mettons en évidence l'influence de ces contraintes supplémentaires sur les solutions recherchées. Nous récapitulons dans la troisième partie les travaux déjà effectués sur le problème sous forme de revue de littérature depuis 1959 jusqu'à nos

jours. Cette revue fait un tour d'horizon des méthodes développées pour résoudre le problème de façon exacte ou approchée. Cette observation qui se veut exhaustive nous a permis de reprendre des idées développées dans le chapitre quatre. La quatrième partie présente la méthode de résolution que nous avons implémentée et les améliorations que nous avons élaborées. Chaque amélioration est décrite selon une approche théorique et testée ensuite numériquement. La cinquième partie présente les jeux d'essais sur lesquels nous avons testé le programme ainsi que les résultats de ces tests effectués sur la méthode de départ puis sur les méthodes d'améliorations présentées dans la partie quatre. La sixième et dernière partie est une conclusion sur les travaux de cette maîtrise et propose des extensions non encore explorées de la méthode.

Chapitre 2

Présentation du problème

2.1 La modélisation du problème

Nous considérons ici un cas particulier du problème de gestion de projet, où la fonction objectif est la minimisation de la date de fin du projet. Il existe d'autres fonctions objectifs qui pourront être traitées par notre méthode et dont nous verrons les définitions dans la revue de littérature (chapitre 3). Le problème de gestion de projet sans contraintes de ressources peut être représenté sous la forme d'un graphe orienté et valué appelé graphe potentiel-tâche ou d'un graphe potentiel-étape appelé diagramme PERT (Program Evaluation and Research Technique) [9]. Le diagramme potentiel-tâche représente les activités avec les nœuds du graphe, tandis que le nœud du diagramme potentiel-étape représente une date de début ou de fin de chaque activité. Nous allons représenter le problème avec contraintes de ressources à l'aide du diagramme potentiel-tâche et définir les notions et le vocabulaire associés à cette représentation du problème. Nous décrirons ensuite le modèle que nous avons développé.

2.1.1 Les données initiales du problème

Le diagramme potentiel-tâche est un graphe orienté $G(N, A)$ où N est l'ensemble des nœuds et A l'ensemble des arcs. N regroupe les n activités du projet plus deux nœuds fictifs appelés respectivement source (S) et puits (P). L'ensemble A regroupe les m relations de préséances entre les activités du projet et les nœuds fictifs.

Soit i ($i \in \{1, \ldots, n\}$) une activité du projet, on distingue les données associées

suivantes :

- la durée de l'activité p_i,
- la consommation de ressources par unité de temps r_i^k où k est le numéro de la ressource et i le numéro de l'activité.
- l'ordre de déroulement du projet ou préséance A,

Ces informations sont complétées par le tableau des disponibilités de ressources ainsi que T, l'horizon du projet.

Considérons un exemple de projet à 3 activités et plusieurs ressources décrit par le tableau 2.1. Ces données peuvent être complétées par des fenêtres de temps sur

Tableau 2.1: *Données initiales d'un projet*

Activité	Successeur(s)	Durée	Besoin en ressource
S	1, 2	0	0
1	3	p_1	$r_1^1, r_1^2, \ldots$
2	3	p_2	$r_2^1, r_2^2, \ldots$
3	P	p_3	$r_3^1, r_3^2, \ldots$
P	$\emptyset$	0	0,0,...

chaque activité ; elles peuvent être données ou calculées en fonction des contraintes de préséances. Ces fenêtres de temps sont représentées pour chaque activité i par l'intervalle de temps $[r_i,\ d_i]$ ce qui signifie que l'activité i doit commencer au plus tôt à l'instant r_i et doit terminer au plus tard à l'instant d_i. Les informations sur ces fenêtres peuvent également être définies *a priori* pour modéliser des dates de disponibilités au plus tôt de certains intrants et des dates de livraisons au plus tard.

2.1.2 Le diagramme potentiel-tâche

Ce projet peut être représenté sous forme de graphe conjonctif ou potentiel-tâche (**Roy** [43]). Dans la figure 2.1, chaque activité est représentée par un nœud du

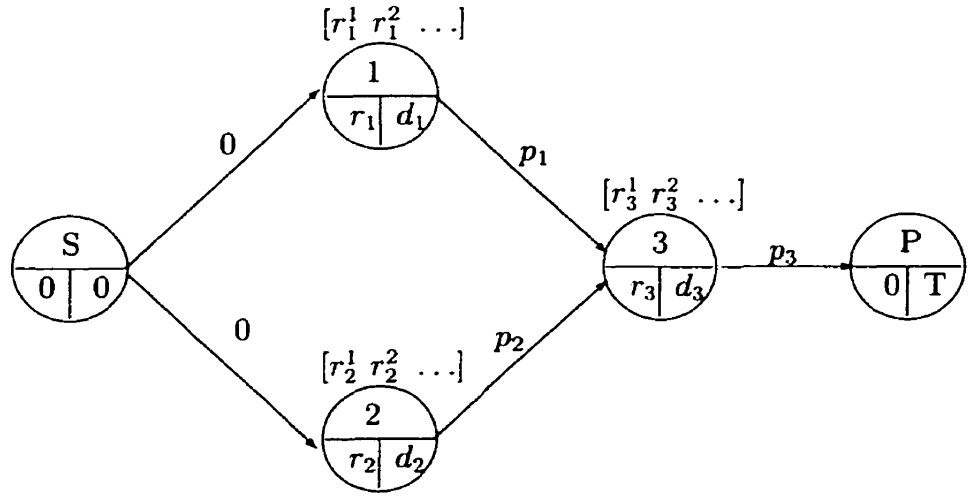


Figure 2.1: *Diagramme potentiel-tâche*

diagramme contenant les informations suivantes : numéro de l'activité et sa fenêtre de temps. Au dessus du nœud figure la consommation en ressource de l'activité par unité de temps. Les nœuds sont reliés par des arcs qui représentent les contraintes de précédence entre les activités. Chaque arc possède une information de durée de l'activité associée au nœud d'origine de l'arc. Un tel diagramme permet de construire un horaire pour le projet encore appelé ordonnancement. Cet horaire définit une date de début pour chaque activité.

2.1.3 Définition d'un ordonnancement

Un ordonnancement est un ensemble de dates de début t_i pour chaque activité i du projet. Un ordonnancement de durée minimale est un ordonnancement minimisant t_{n+1}, la date de début de la tâche fictive du diagramme potentiel-tâche. La recherche des ordonnancements au plus tôt sans contraintes de ressources se fait simplement avec l'algorithme de Bellman, présenté au chapître 4, de complexité $O(m)$ où m est le nombre d'arcs du diagramme potentiel-tâche. La première étape ou itération de notre méthode revient à effectuer ce calcul simple pour ensuite vérifier les contraintes de ressources en calculant le profil de consommation de chaque ressource dans le temps. Nous pouvons illustrer ce calcul à l'aide des

deux diagrammes présentés ci-dessous : le diagramme de GANTT et le profil des ressources.

2.1.4 Le diagramme de GANTT & les profils des ressources

Le diagramme de GANTT est tracé lorsque l'ordonnancement du projet est défini. Il permet de représenter graphiquement l'occupation du temps par les activités du projet. Et de ce diagramme découle le profil de consommation des ressources allouées aux activités. C'est à partir du profil de consommation des ressources que l'on peut identifier les "conflits de ressources". C'est à dire les dates pour lesquelles l'ordonnancement fait entrer en concurrence des activités par rapport à une ou plusieurs ressources. Pour remédier à un tel problème, il est nécessaire de décaler l'ordonnancement des activités en conflit de manière à éviter qu'elles soient ordonnancées en même temps. L'exemple suivant illustre ce qu'est un conflit de ressource.

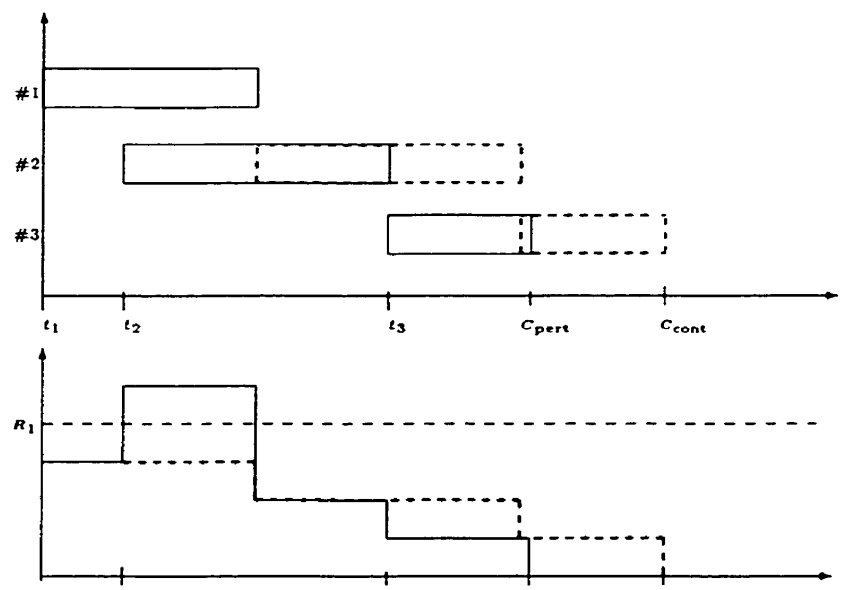


Figure 2.2: *Diagramme de GANTT*

Le premier graphe de la figure 2.2 représente le diagramme de GANTT du

projet de la figure 2.1. L'activité #1 commence à la date t_1, puis l'activité #2 à l'instant $t_2 \geq t_1$. Enfin l'activité #3 ne peut commencer qu'à l'instant t_3 qui correspond au plus tôt à la date de fin des deux activités #1 et #2. Au dessous de ce graphe, le deuxième graphe représente le profil de consommation de la ressource numéro 1 par les activités du projet dans le temps. Il est possible de construire de la même manière les profils de consommations pour chaque ressource du problème. La courbe en traits pleins est associée à l'ordonnancement du diagramme de GANTT en traits pleins. Respectivement, la courbe en traits pointillés est le profil de consommation de l'ordonnancement du diagramme de GANTT en traits pointillés. Le premier ordonnancement engendre une violation de consommation de la ressource 1, car la consommation de cette ressource à l'instant t_2 dépasse la capacité R_1 de la ressource 1. En revanche la courbe de consommation du deuxième ordonnancement (en pointillés) reste toujours en dessous de la droite R_1, donc cet ordonnancement ne génère pas de violations. La date C_{pert} est la date de fin du projet résolu sans contraintes de ressources. La date C_{cont} correspond à la date de fin du même projet en tenant compte de la ressource numéro 1.

Le problème de gestion de projet avec contraintes de ressources peut être schématisé par la figure 2.3.

Minimum de la durée du projet
Contraintes d'exécution des tâches
Contraintes de préséance sur les tâches
Contraintes de fenètres de temps sur les tâches
Contraintes de ressources

Figure 2.3: *Problème de gestion de projet avec contraintes de ressources*

Nous traîtons ce problème avec la décomposition de Dantzig-Wolfe [12]. Une partie des contraintes est transférée dans un sous-problème et le problème restant est appelé le problème maître. Les deux problèmes sont résolus suivant une procédure itérative de génération de colonnes à l'image du logiciel GENCOL [18]. Avant de construire la méthode que nous détaillons au chapitre 4, nous avons recherché dans

la littérature s'il existait une approche semblable à la notre, sinon de quelle manière le problème était traîté pour déterminer l'ordonnancement optimal.

Chapitre 3

Revue de littérature

Afin de connaître l'état de l'art sur le problème de gestion de projet avec contraintes de ressources, voici la synthèse d'une revue de litterature qui s'est voulue exhaustive. Le tour d'horizon des techniques employées depuis les années 1959 nous a permis de récolter des idées et de reprendre des algorithmes que nous présenterons au chapitre 4. Cette recherche nous a également permis de vérifier que rien n'avait déjà été réalisé avec les méthodes de génération de colonnes, pour résoudre le problème.

Pour reprendre les propos de R. Kolish [29], on distingue deux grandes catégories de méthodes pour la résolution du problème de gestion de projet avec contraintes de ressources (Resource Constrained Project Scheduling Problem) qui sont les méthodes exactes et les méthodes approchées. Les premières fournissent la solution optimale au problème tandis que les méthodes approchées donne un solution sans garantie sur l'optimalité.

Les méthodes trouvées dans la littérature sont les suivantes :

- les méthodes exactes :
  - programmation dynamique (Carruthers et al. [10])
  - programmation linéaire en nombres entiers (Bowman [6], Pritsker et al. [41], Patterson et al. [36])
  - recherche arborescente (Balas [2], Stinson et al. [47], Demeulemeester et al. [15])
  - décomposition (Roundy et al. [42])
- les méthodes approchées :

- règles de décisions (R. Kolisch [28])
- recherche arborescente partielle (Alvarez-Valdès et al. [1])
- programmation en nombres entiers avec heuristiques (Oğuz et al. [34])
- concept des arcs disjoints (Bell et al. [4])
- recherche locale (Leon et al. [31])

Parmi les méthodes approchées, celle des règles de décisions est reconnue comme étant la meilleure; quant aux méthodes exactes, c'est celle par recherche arborescente qui a jusqu'à maintenant fait l'objet d'applications réalistes.

3.1 Les méthodes exactes

Cette partie concerne les méthodes qui fournissent la solution optimale au problème de gestion de projet avec contraintes de ressources. Nous présentons ici les méthodes développées depuis 1959 sur le sujet, en nous concentrant sur les trois dernières qui sont la programmation linéaire en nombres entiers, la recherche arborescente et la décomposition. Ces trois familles présentent l'intéret d'apporter soit les meilleurs résultats actuels, soit des similitudes avec l'approche que nous proposons. Cette revue nous a permis de connaître les différentes fonctions objectifs liées à ce problème générique et les techniques employées pour réduire la taille des problèmes.

3.1.1 La programmation linéaire en nombres entiers

La programmation linéaire a fait l'objet de nombreux travaux depuis les années 1960 concernant les problèmes liés à la gestion de projet. Les formulations ont connus des progrès en terme de complexité algorithmique et taille de modèles (nombre

de variables et de contraintes) mais leurs exploitations était à l'époque difficile à valoriser à cause de la performance réduite des ordinateurs. Ce handicap peut expliquer pourquoi ces techniques n'ont pas été reprises plus tard.

E. W. Davis 1973 [11] fait un état de l'art des méthodes de programmation linéaires en nombre entiers plutôt pèssimiste et dont nous récapitulons ici seulement les traits principaux liés à notre travail.

Wiest (1963) [52] a publié une formulation en Programmation Linéaire en Nombre Entier (PLNE) du problème avec contraintes de ressources, et visant à minimiser la durée totale du projet. Il s'inspirait d'une formulation de problème de job-shop publiée en 1959 par **Bowman** [6].

En 1968, **Pritsker et Watters** [40] ont formulé le problème généralisé de gestion de projet avec fenêtres de temps sous forme de programme linéaire mais n'ont pas proposé de résolution.

En 1969, **Pritsker, Watters et Wolfe** [41] ont présenté un PLNE pour la gestion de plusieurs projets avec plusieurs ressources. Trois fonctions objectifs différentes peuvent être adoptées :

- durée totale minimum (minimisation de la durée du projet maître : date de fin - date de début),
- temps de fin minimum (minimisation de la date de fin du projet),
- coût des retards minimum (minimisation des retards de fin d'activités des projets).

Les extensions possibles au problème sont la préemption des tâches, les permutations des ressources entre deux projets, et des besoins en ressources variables par période de temps. L'application numérique a été illustrée par un problème de 8 tâches, 3 projets et 3 ressources. Les résultats ont été comparés avec ceux de méthodes heuristiques diverses. Cette formulation requière 33 variables et 37 contraintes, alors que la formulation de Bowman nécessite 72 variables et 125 contraintes. De plus, le temps de calcul est de 2.3 minutes sur IBM 7044. Ce modèle a été repris plus tard notamment pour être la structure d'une méthode de décomposition (**Deckro et al.** [14], en 1991).

La programmation linéaire en nombres entiers était, selon E. W. Davis [11], un outil lourd à utiliser pour résoudre les problèmes de gestion de projet car il requiert beaucoup de mémoire et de temps de calculs, ce qui limitait, à l'époque de ces articles (1966 à 1985), les tailles des problèmes à résoudre.

3.1.2 La recherche arborescente

La partie suivante n'est pas directement liée à notre méthode mais elle permet de présenter les meilleurs résultats actuels de résolution du problème de gestion de projet avec contraintes de ressources. C'est à travers la description de la méthode la plus performante que nous avons trouvé des idées d'améliorations de notre méthode comme le calcul des bornes inférieures par exemple.

Au regard des limites de la programmation linéaire en nombres entiers, se sont développées en parallèle des méthodes d'énumérations telles que la recherche arborescente. D'après **J.H. Patterson** [37] en 1984, la programmation linéaire est un échec comme le montrent **Brand et al.** [7] en 1964 et **Davis** [11] en 1973 puisque la taille des problèmes résolus reste petite comparée aux méthodes plus récentes de recherche arborescente.

Selon lui, les seules méthodes exactes valables entrent dans les trois classes suivantes :

1. énumération bornée (*bounded enumeration*)
2. recherche arborescente (*branch and bound*, [21], [47])
3. énumération avec retour en arrière (*implicit enumeration*)

D'après les tests qu'il a réalisés sur ces trois méthodes il retient la dernière ("implicit enumeration") comme étant la plus rapide. La méthode d'énumération bornée ne convient pas lorsque le nombre de sous-ensembles d'activités réalisables pour un horaire est trop élevé.

Les prévisions de Patterson étaient justes car la technique qui s'est développée par la suite pour résoudre le problème de gestion de projet est celle de la recherche arborescente avec retour en arrière, au dépens des méthodes de PLNE.

L'article de **E. Demeulemeester et al.** [15] qui présente la méthode de "Branch and Bound" en est le meilleur exemple.

Brièvement, cette procédure de recherche démarre avec une grande valeur (infini) pour la borne supérieure de la valeur de l'objectif. L'espace de recherche est par la suite restreint à toutes les solutions dont la valeur de l'objectif est strictement inférieur à la valeur courante de la borne supérieure. Chaque fois qu'une solution trouvée est réalisable, (dont la valeur de l'objectif est inférieure à la borne sup.) la borne supérieure est remise à jour pour prendre dès lors la valeur de l'objectif de la solution trouvée. La recherche est alors poursuivie avec un ensemble de solutions réalisables plus petit (toutes les solutions dont la valeur de l'objectif est inférieure à la borne supérieure d'au moins une unité de temps). La recherche s'arrête lorsque toutes les solutions ont été testées, soit par énumération, si elles ont appartenu à l'ensemble réalisable, soit par élimination, si elles faisaient dépasser la valeur de l'objectif de la borne supérieure. C'est en effet la borne supérieure qui dimensionne

l'espace de recherche. Elle est garante de la performance de la méthode.

Les nœuds de l'arbre de recherche arborescente sont des sous-ensembles d'activités satisfaisant les contraintes de précédence et de capacités de ressources.

Chaque activité est commencée dès que possible suivant l'ordre de précédence et les ressources disponibles. Les activités sont alors ordonnancées mais de façon temporaire car les sous-ensembles formés peuvent être retardés plus tard dans l'algorithme.

Les ordonnancements temporaires sont construits à partir de l'instant 0 par ajout de sous-ensembles d'activités, jusqu'à l'inclusion de l'ensemble des activités du projet. Pour définir les configurations d'ordonnancement partiel, chaque sous-ensemble d'activités ordonnancées est comparé à une borne inférieure L_q (Stinson [47]). Cette borne considère les contraintes de précédences et de ressources, et elle est calculée pour toutes les configurations.

Idée de l'algorithme :

1. trouver l'ordonnancement partiel associé à un sous-ensemble d'activités ordonnancées
2. déterminer le chemin critique de précédence du nouveau réseau.
3. construire l'ensemble des activités hors chemin critique et non encore ordonnancées.
4. s'il existe une activité i, de durée d_i, de cet ensemble (activités non ordonnancées) qui est compatible avec les contraintes de précédence et de ressources pendant $e_i \leq d_i$ périodes de temps, alors la fin du projet peut être retardée de seulement $d_i - e_i$ périodes.

D'après Stinson [47] la borne inférieure s'écrit :

$$L_q = \max_{i \in NC}(z + d_i - e_i)$$

où z est la longueur du chemin critique du nouveau réseau engendré par l'ordonnancement partiel courant et NC est l'ensemble des activités hors chemin critique.

Cette méthode résout les problèmes de Patterson (cf. chapitre 5) en 0.125 secondes en moyenne (écart type : 0.314 sec.) sur un PC avec un programme écrit en Turbo C. Ces résultats sont presque 12 fois meilleurs que ceux de la procédure de Stinson [47]. Cette méthode permet, de plus, d'avoir une solution réalisable proche de l'optimum très rapidement même pour des problèmes de grandes tailles (nb. d'activités > 50).

Les articles de **Kolish et al. [29]** et **Mingozzi et al. [33]** reconnaissent que c'est la meilleure méthode exacte existant actuellement dans la littérature.

Mingozzi et al. [33] ont mis au point une nouvelle borne inférieure fondée sur une formulation du problème différente de celle de Demeulemeester. Elle est calculée par une heuristique de résolution d'un problème de sac à dos ("set packing").

La nouvelle procédure de E.L. Demeulemeester [17] utilise cette nouvelle borne et peut maintenant résoudre les problèmes KSD [28] en moins de 45 minutes (30 activités et 4 ressources), pour une moyenne de 12.3 secondes sur un PS/2 P75 avec un processeur de 486 à 25Mhz et une mémoire vive de 32 Mb. D'après l'auteur, l'optimalité n'était pas trouvée lorsqu'il y avait un problème de mémoire saturée, ce qui n'arrive plus avec la nouvelle procédure.

La nouvelle borne de Mingozzi [33] est en effet plus efficace que celle de Stinson [47]. Son calcul est décrit ci-après. Pour chaque activité i, on définit une "sœur"

comme une activité qui peut être réalisée en parallèle sans violation de contraintes de précédence ou de ressources. Toutes les activités non ordonnancées sont enregistrées dans une liste L dans l'ordre croissant de leur nombre de sœurs. La borne prend initialement la valeur de la première date de fin des activités en cours de réalisation. A cette valeur sont ajoutées les durées des activités contenues par L.

La méthode de E.L. Demeulemeester [17], bien que reconnue comme la plus rapide pour résoudre les problèmes de 30 activités et 4 ressources, présente les limites suivantes :

- Il faut 3 heures de calcul pour résoudre le problème KSD291 (30 activités et 4 ressources).
- L'objectif de sa méthode ne peut pas être modifié sans entraîner des modifications de l'algorithme de recherche.
- La méthode ne peut pas être étendue au cas multi-mode.
- 429 sur 480 problèmes sont résolus en 14.757 secondes en moyenne de façon exacte, les autres temps de calculs sont de l'échelle de l'heure.
- Elle ne peut pas résoudre les problèmes de KSD de plus de 30 activités.

3.1.3 La décomposition du problème

Nous proposons une méthode de décomposition dite de Dantzig-Wolfe [12]. Elle a déjà été utilisée comme le montre cette partie, mais pour des problèmes différents. **Roundy et al.** [42] en 1991 ont repris le modèle de Pritsker et al. (1968) [40] pour résoudre un problème de planification des opérations d'un problème de production en temps réel.

Leur modèle mathématique est un modèle d'atelier ("job shop") où chaque pièce à fabriquer possède une suite d'opérations sur des machines définies à l'avance. L'objectif est de minimiser le retard des fabrications par rapport à des dates données.

Le problème est reformulé pour devenir une fonction de maximisation. La contrainte qui porte sur la durée des pièces sur chaque machine est remontée dans l'objectif par relaxation lagrangienne. Ceci aboutit à N sous-problèmes (un par pièce) indépendants. Or l'auteur observe que les contraintes de ces sous-problèmes sont les contraintes duales d'un problème de flot dans un réseau.

Le sous-problème est relaxé en remplaçant la contrainte d'intégrité de la variable binaire x par des contraintes linéaires du type : $0 \leq x \leq 1$.

Le problème ainsi formulé est le dual d'un problème de flot maximum dans un graphe. La structure du réseau dans lequel doit circuler le flot maximum a permis à l'auteur de créer un algorithme de calcul de flot maximum de complexité linéaire. La résolution des sous-problèmes avec un vecteur des cœfficients de Lagrange λ donné permet de trouver une borne inférieure au problème (P_λ) qui est une borne supérieure du profit optimal du problème (P). La difficulté de la résolution réside dans le choix des cœfficients de Lagrange. Il est fait grâce à la méthode standard des sous-gradients. La résolution du flot maximum donne une coupe de coût minimum dans le graphe qui permet de retrouver les dates de débuts des opérations de chaque pièce.

La plus grande taille des problèmes abordés dans l'article est entre 6 et 12 machines et jusqu'à 54 pièces. La qualité des solutions de cette méthode augmente par rapport aux méthodes heuristiques avec la taille des problèmes testés.

Les auteurs concluent leur article en disant que leur formulation est prometteuse mais qu'il reste à améliorer les algorithmes et les structures de données avant que l'implémentation de leur méthode soit exploitable.

Deckro et al. [14] ont mis au point une méthode de décomposition qui permet de résoudre des problèmes de grandes tailles. Ils se servent d'une méthode de décomposition pour trouver des solutions heuristiques. (Ils se contentent de bornes supérieures). Ils reprennent le modèle de Pritsker et al. [41] pour le problème de plusieurs projets et décomposent ce problème en sous-problèmes. Chaque sous-problème représente un projet à ordonnancer pour respecter les précédences, les contraintes des ressources étant remontées dans l'objectif par relaxation lagrangienne. Le problème maître lie les sous-problèmes en choisissant les horaires de chaque projet qui rendent une solution finale sans violation de contraintes de ressources.

Le problème résolu dans l'article possède 116 activités et 3 types de ressources dont la consommation est étalée sur 28 périodes de temps. Ces résultats montrent la difficulté combinatoire de ce problème. Aussi, le problème sur lequel nous voulons travailler est plus simple que le problème à plusieurs projets, puisqu'il ne possède qu'un seul sous-problème, ce qui nous permet d'envisager de meilleurs performances en terme de taille de problème avec une méthode de décomposition.

3.2 Les méthodes approchées

Cette partie concerne des méthodes fondamentalement différentes de la notre mais nous allons reprendre certaines techniques développées dans cette littérature pour réduire la taille du problème (cf. chapitre 4). En effet les méthode heuristiques offrent l'avantage de calculer rapidement des bornes supérieures de bonne qualité pour le problème de durée de projet minimale. Nous retiendrons parmi ces méthodes celles de règles de décisions car elles sont faciles à implémenter.

3.2.1 Les règles de décisions

Les régles de décisions suivent un raisonnement intuitif qui consiste à fixer une date de début aux activités de façon séquentielle en fonction d'un critère de choix prédéfini.

Les méthodes heuristiques avec règles de décisions se décomposent suivant deux caractéristiques :

- la méthode de construction de l'ordonnancement :
  - méthode sérielle (tri des activités avant l'opération d'ordonnancement)
  - méthode parallèle (tri des activités chaque fois qu'une activité est ordonnancée)
- la règle de priorité :
  - MTS : nombre max de successeurs ("Most total successors)
  - LST : date de début la plus récente ("Latest start time")
  - GRPW : maximum de temps d'activité écoulé ("Greatest rank positional weight")
  - WRUP : taux pondéré d'utilisation des ressources et des précédences maximaux ("Weighted resource utilisation ratio and precedence")
  - LFT : date de fin la plus proche ("Latest finish time")
  - MSLK : écart minimum entre date de complétion au plus tôt et date de remise (cf. Davis et Patterson [13]).

Performances :

- sérielles : 9.13 % d'écart avec l'optimum (Boctor [5])

- parallèles : 2.89% de l'optimum avec les problèmes de Patterson (Bedworth et Bailey [3]).

D'après **R. Kolisch** [29], les meilleures méthodes approchées sont celles avec des règles de décisions sérielles (R. Kolisch [28]) pour des problèmes de grandes tailles et des jeux d'essais qui sont seulement modérément contraints en terme de capacités de ressources. Les méthodes parallèles, par contre, sont meilleures pour les problèmes "difficiles" (avec des contraintes de ressources serrées).

3.3 Bilan de la revue de littérature

La recherche est avancée sur le domaine du problème de gestion de projet avec contraintes de ressources. Il est important de connaître ces méthodes pour pouvoir s'en servir dans de nouvelles approches comme celle que nous allons définir au prochain chapitre. La méthode de recherche arborescente de Demeulemeester [15] est présentement la plus performante. Cette approche est différente de la notre, mais nous allons reprendre les techniques qu'elle utilise notamment dans le calcul des bornes.

Notre formulation mathématique ressemble à celle du problème d'atelier ("job shop") de Roundy et al. [42] présentée dans la partie sur les décompositions. Une telle approche par décomposition n'est peut être pas aussi rapide que les méthodes arborescentes, mais devrait néanmoins apporter de nouvelles performances de résolutions des problèmes GRCPSP de grandes tailles pour des types d'objectifs divers.

Chapitre 4

Méthodes et outils

Le problème de gestion de projet avec contraintes de ressources, peut être résolu avec la décomposition de Dantzig-Wolfe en séparant les contraintes liées à l'ordonnancement de celles liées aux ressources. La méthode revient à générer des ordonnancements du projet sans contraintes de ressources à l'aide d'un sous-problème et à les confrontrer aux contraintes de ressources dans le problème maître.

Chaque horaire du sous-problème permet de construire une colonne de la matrice des contraintes du problème maître à l'aide du profil de consommation de ressources qui lui est associé. Le problème maître recherche alors la colonne optimale par rapport aux ressources, et à la date de fin des activités du projet.

Comme le nombre de ces colonnes est très grand, nous procédons par génération de colonnes, en générant des colonnes avec le sous-problème. Le problème maître utilise ces colonnes pour calculer une combinaison linéaire convexe, solution linéaire du problème, dont les variables duales associées serviront au sous-problème pour rechercher une colonne de coût marginal minimum. Dès que le sous-problème ne trouve plus de colonne de coût marginal négatif, la solution du problème est optimale, mais en nombres réels.

Cette solution fournit une borne inférieure et une solution à partir de laquelle la solution optimale en nombres entiers peut être trouvée par évaluation et séparation progressive (appelé aussi recherche arborescente ou encore branchement). Le schéma de la figure 4.1 récapitule le fonctionnement de cette méthode. Le premier module

de l'algorithme est le programme linéaire qui fournit une solution au branchement. Le branchement va relancer le programme linéaire à chaque fois qu'il aura construit des noeuds à explorer.

Les prochaines sections présentent en détail cette méthode et son application à notre problème de gestion de projet. Ainsi nous détaillons la méthode de base dans les sections 4.1, à 4.4 à savoir la formulation mathématique, le sous-problème, le problème maître et le branchement. Dans la section 4.5, nous proposons, construisons et testons numériquement différentes techniques pour améliorer la borne inférieure de la méthode de base. Nous faisons le même travail pour la borne supérieure dans la section 4.6 en y ajoutant la présentation avec test d'une méthode de coupe de l'arbre de recherche. Enfin la section 4.7 est une présentation de deux techniques d'accélération du problème maître.

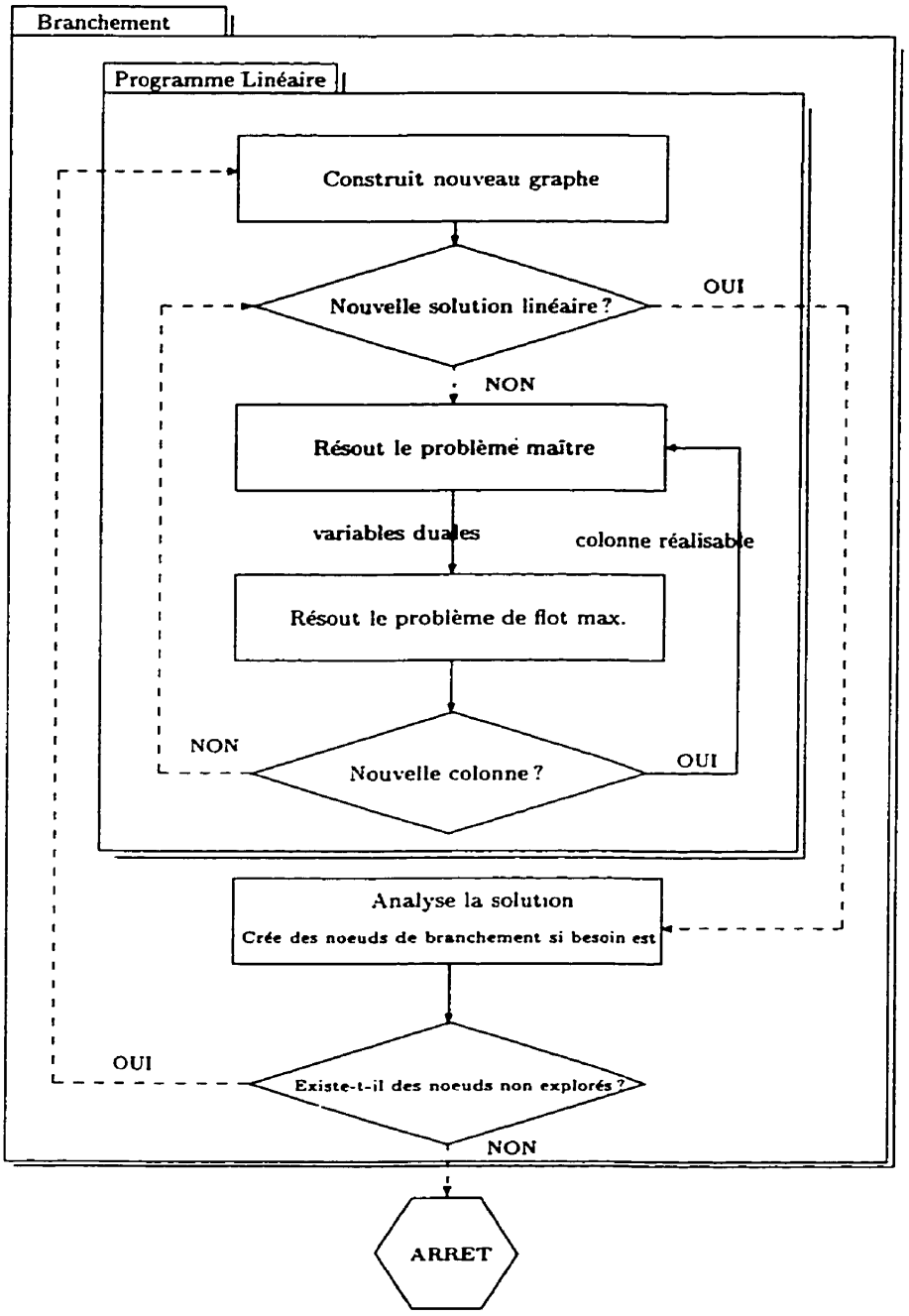


Figure 4.1: *Algorithme de résolution*

4.1 Présentation du modèle mathématique

Parmi les méthodes exactes en programmation linéaire passées en revue précédemment dans le chapitre 3, celles qui paraissent les plus prometteuses sont les méthodes de décomposition. Ces dernières résolvent des problèmes d'ordonnancement, problèmes qui sont moins complexes que le problème de gestion de ressources avec contraintes de ressources (cf. section 3.1.3).

Les méthodes de résolution existantes utilisent la relaxation lagrangienne et une méthode de sous-gradients. Ici nous proposons d'exploiter la décomposition du programme linéaire selon la méthode de Dantzig et Wolfe [12] comme l'a décrit **S. Gélinas** [22]. Le problème considéré est la gestion de projet avec contraintes de ressources et fenêtres de temps. La fonction objectif est une combinaison linéaire des dates de début des activités. Le sous-problème est un problème de flot maximum mais différent de celui de l'article de Roundy et al. [42] qui est spécifique aux problèmes d'atelier ("job shop").

L'autre différence provient du type de décomposition. Ici nous avons choisi celle de Dantzig-Wolfe car elle est plus simple à mettre en place que la décomposition lagrangienne qui nécessite un calcul hardu des coefficients de Lagrange.

Le problème maître est un programme linéaire en nombres entiers de grande taille. La relaxation linéaire du problème est résolue par génération de colonne. Le résultat du programme linéaire est ensuite affiné par un branchement sur les intervalles de temps (fenêtres de temps) qui modifie le graphe du sous-problème pour pouvoir trouver une solution entière.

Nous allons voir comment le problème mathématique est formulé, puis décomposé pour ensuite proposer des méthodes de résolutions du sous-problème et du

problème maître pour la fonction objectif classique de minimisation de la date de fin du projet ($\min C_{max}$).

Le problème peut être décomposé selon la méthode de Dantzig et Wolfe en un problème **maître**, qui ne prend en compte que les contraintes de ressources, en fonction des horaires proposés par un sous-problème qui lui prend en compte la préséance des tâches et leur continuité dans le temps.

4.1.1 Définition du programme linéaire en nombres entiers

La formulation que nous utilisons est celle présentée par S. Gélinas [22]. Elle est équivalente à la formulation déjà rencontrée dans Pristker [40] pour les variables, et Roundy et al. [42] pour la décomposition des contraintes.

Paramètres :

u, v : $u, v = 1, \ldots, N$	opérations ou activités du projet,
k : $k = 1, \ldots, K$	type de ressources,
p_u : $u = 1, \ldots, N$	durée de l'opération u,
r_u : $u = 1, \ldots, N$	temps au plus tôt pour commencer l'opération u,
d_u : $u = 1, \ldots, N$	temps au plus tard pour terminer l'opération u,
R_k : $k = 1, \ldots, K$	quantité de ressources disponibles pour le type k,
r_{uk} : $u = 1, \ldots, N$ $k = 1, \ldots, K$	quantité de ressources du type k utilisées par l'opération u,
$A = \{(u, v) \mid u \in \Gamma^-(v)\}$:	relations de précédences,
$T = \max_u d_u$:	horizon du projet,
t : $t = 1, \ldots, T-1$	période de temps,

$c_u^t : u = 1, \ldots, N$ $t = 1, \ldots, T-1$	coût de commencement de l'opération u à l'instant t.

Variables :

$$x_u^t = \begin{cases} 1 \text{ si l'opération u commence au temps t ou avant,} \\ 0 \text{ sinon.} \end{cases}$$

$$\forall u = 1, \ldots, N \text{ et } t = 1, 2, \ldots, T-1.$$

Problème :

$$\min \sum_{u=1}^{N} \sum_{t=r_u}^{d_u - p_u} c_u^t (x_u^t - x_u^{t-1}) \tag{4.1}$$

$$\text{s. c. :} \quad x_u^t \leq x_u^{t+1} \quad u = 1, \ldots, N \text{ et } t \in [r_u \,;\, d_u - p_u - 1], \tag{4.2}$$

$$x_v^t \leq x_u^{t-p_u} \quad \forall (u, v) \in A \text{ et } t \in [r_u + p_u \,;\, d_u], \tag{4.3}$$

$$\sum_{u=1}^{N} r_{uk} (x_u^t - x_u^{t-p_u}) \leq R_k \quad k = 1, \ldots, K \text{ et } t \in [0 \,,\, T-1], \tag{4.4}$$

$$x_u^t = 0 \quad u = 1, \ldots, N \text{ et } t < r_u, \tag{4.5}$$

$$x_u^t = 1 \quad u = 1, \ldots, N \text{ et } t \geq d_u - p_u, \tag{4.6}$$

$$x_u^t \in \{0 \,;\, 1\} \quad u = 1, \ldots, N \text{ et } t \in [r_u \,;\, d_u - p_u]. \tag{4.7}$$

L'objectif (4.1) minimise la somme des coûts de démarrage des activités. Si l'opération u commence en t, l'expression $(x_u^t - x_u^{t-1})$ prend la valeur 1 en un seul temps t. Les contraintes (4.2) (contraintes de continuité) assurent que si l'opération u est commencée en t_0, elle l'est aussi pour tout $t \geq t_0$. Les contraintes (4.3) représentent les préséances des tâches ou activités, tandis que les contraintes de ressources sont décrites par les contraintes (4.4). (4.5) et (4.6) assurent que les opérations sont effectuées dans les fenêtres de temps. Elles peuvent être remplacées en définissant seulement les variables à l'intérieur de chaque fenêtre de temps, les autres étant inexistantes, seule la contrainte suivante subsiste :

$$x_u^{d_u - p_u} = 1 \; \forall u = 1, \ldots, N.$$

$$x_u^{r_u} = 0.$$

Enfin, (4.7) traduit la nature binaire des variables x_u^t.

Valeurs des paramètres c_u^t :

Soient w_u un poids défini par activité u, et d'_u l'heure au plus tard pour terminer l'opération u sans retard. Il est possible de choisir plusieurs fonctions objectifs :

- somme pondérée des temps : $c_u^t = w_u(t + p_u)$,
- somme pondérée des retards : $c_u^t = w_u \max\{0, t + p_u - d'_u\}$,
- fonction quelconque du temps de complétion des opérations,
- fonctions objectifs du type min-max (par exemple : C_{max}, T_{max}, L_{max}).

Il est possible de modéliser toute fonction objectif combinaison linéaire ou polynomiale des dates de début des activités. La fonction financière NPV (Net Present Value) ne peut pas être utilisée dans notre modèle, car elle possède une fonction exponentielle. Néanmoins, s'il est possible de se contenter d'une approximation linéaire de la fonction, le problème peut être envisagé. (Cette fonction est utilisée pour ordonnancer un projet dont on veut minimiser les paiements d'interets dus aux investissements que chaque activité nécessite.)

4.1.2 Décomposition de Dantzig-Wolfe

Cette partie présente la décomposition tel que présentée par S. Gélinas [22]. Cependant, nous proposons une reformulation du sous-problème pour l'exploiter comme un problème de flot maximum.

Données :

H :	ensemble des horaires réalisables,
h : $h \in H$	horaire réalisable par rapport aux préséances,
k : $k = 1, \ldots, K$	type de ressources,
$]t-1,\ t]$: $t \in [0, T-1]$	période de temps t,
c_h : $h \in H$	coût de l'horaire h,
n^t_{hk} : $t \in [0, T-1]$,	quantités de ressources du type k utilisées,
$h \in H$, $k = 1, \ldots, K$	à la période t dans l'horaire h.

Variables :

$$y_h = \begin{cases} 1 \text{ si l'horaire h est choisi,} \\ 0 \text{ sinon}, h \in H. \end{cases}$$

Problème maître :

$$\min \sum_{h \in H} c_h y_h \tag{4.8}$$

$$\text{s. c. :} \quad \sum_{h \in H} n^t_{hk} y_h \leq R_k \quad \forall t \in [0, T-1] \text{ et } k = 1, \ldots, K \tag{4.9}$$

$$\sum_{h \in H} y_h = 1 \tag{4.10}$$

$$y_h \in \{0\,;\,1\} \quad \forall h \in H. \tag{4.11}$$

L'horaire de coût minimum est choisi (4.8), à condition que celui-ci respecte les contraintes de ressources (4.9). Les contraintes (4.10) et (4.11) assurent qu'un et un seul horaire est choisi.

La contrainte (4.11) est relaxée lors des itérations de la génération de colonne pour devenir la contrainte suivante :

$$0 \leq y_h \leq 1 \ \forall h \in H.$$

Calcul des coûts réduits :

Le problème maître permet de pondérer les horaires sur lesquels il travaille et qui sont générés par le sous-problème.

Soient $\alpha_k^t \geq 0$ et λ les variables duales associées aux contraintes (4.9) et (4.10) respectivement, le sous-problème doit fournir un horaire réalisable au vu des préséances et de coût marginal minimum c_{h^*}.

$$c_{h^*} = \min_{h \in H} \{c_h + \sum_{k=1}^{K} \sum_{t=0}^{T-1} \alpha_k^t n_{hk}^t - \lambda\}$$

Ce qui revient à minimiser :

$$\sum_{u=1}^{N} \sum_{t=r_u}^{d_u - p_u} \tilde{c}_u^t x_u^t,$$

$$\text{avec } \tilde{c}_u^t = c_u^t - c_u^{t+1} - \sum_{k=1}^{K} r_{uk}(\alpha_k^{t+p_u} - \alpha_k^t)$$

Sous-problème :

$$\min \sum_{u=1}^{N} \sum_{t=r_u}^{d_u - p_u} \tilde{c}_u^t x_u^t$$

s. c. : (4.2), (4.3), (4.5), (4.6), (4.7)

Ce programme linéaire est un problème de fermeture minimale sur un graphe $G(N, A)$ de N nœuds avec l'ensemble des arcs A. Une fermeture sur G est l'ensemble S des nœuds, sous-ensemble de N, tel que pour tout nœud de la fermeture, les successeurs appartiennent aussi à la fermeture. On désigne la valeur de la fermeture S comme étant la somme des valeurs de ses nœuds. Une fermeture est dite minimale (resp. maximale) si sa valeur est la plus grande (resp. petite) parmi les valeurs de toutes les fermetures possibles.

4.1.3 Reformulation du sous-problème

Nous allons montrer comment identifier ce sous-problème à un problème de flot maximum. Il est nécessaire, pour cela, de le transformer en problème de fermeture maximale. Nous avons par conséquent choisi de réécrire le sous-problème avec de nouvelles variables. (La formulation précédente conduit à un problème de fermeture à coût minimale qui n'est pas identifiable à un problème de flot maximum.)
Posons pour $u = 1, \ldots, N$ et $\forall t$:

$$z_u^t = \begin{cases} 1 \text{ si l'opération u n'est pas commencée à l'instant t,} \\ 0 \text{ sinon} \end{cases}$$

Autrement écrit :

$$z_u^t = 1 - x_u^t \quad u = 1, \ldots, N \ \forall t$$

Le problème se réécrit de la façon suivante :

$$\min \sum_{u=1}^{N} \sum_{t=r_u}^{d_u - p_u} \bar{c}_u^t (1 - z_u^t) \tag{4.12}$$

$$\text{s. c. :} \quad z_u^t \geq z_u^{t+1} \quad u = 1, \ldots, N \text{ et } t \in [r_u \,;\, d_u - p_u - 1], \tag{4.13}$$

$$z_v^t \geq z_u^{t-p_u} \quad \forall (u, v) \in A \text{ et } t \in [r_u + p_u \,;\, d_u], \tag{4.14}$$

$$z_u^t = 1 \quad u = 1, \ldots, N \text{ et } t < r_u, \tag{4.15}$$

$$z_u^t = 0 \quad u = 1, \ldots, N \text{ et } t \geq d_u - p_u, \tag{4.16}$$

$$z_u^t \in \{0 \,;\, 1\} \quad u = 1, \ldots, N \text{ et } t \in [r_u \,;\, d_u - p_u]. \tag{4.17}$$

L'objectif (4.14) se réécrit :

$$\sum_{u=1}^{N} \sum_{t=r_u}^{d_u - p_u} \bar{c}_u^t - \max \sum_{u=1}^{N} \sum_{t=r_u}^{d_u - p_u} \bar{c}_u^t z_u^t$$

Soit le nouveau sous-problème :

$$\max \sum_{u=1}^{N} \sum_{t=r_u}^{d_u - p_u} \bar{c}_u^t z_u^t \tag{4.18}$$

$$\text{s. c. :} \quad z_u^{t+1} - z_u^t \leq 0 \;\; u = 1, \ldots, N \text{ et } t \in [r_u \,;\, d_u - p_u - 1], \tag{4.19}$$

$$z_u^{t-p_u} - z_v^t \leq 0 \;\; \forall (u,v) \in A \text{ et } t \in [r_u + p_u \,;\, d_u], \tag{4.20}$$

$$z_s - z_u^t \leq 0 \;\; u = 1, \ldots, N \text{ et } t < r_u, \tag{4.21}$$

$$z_u^t - z_T \leq 0 \;\; u = 1, \ldots, N \text{ et } t \geq d_u - p_u, \tag{4.22}$$

$$z_u^t \leq 1 \;\; u = 1, \ldots, N \text{ et } t \in [r_u \,;\, d_u - p_u] \tag{4.23}$$

$$z_s = 1, \;\; z_T = 0. \tag{4.24}$$

Cette formulation est celle d'un problème de fermeture maximale (*maximum closure*) sur un graphe $G(N, A)$. La variable z_u^t représente un nœud du graphe G. On associe à chaque nœud (u, t) une valeur $\bar{c}_u^t$ de signe quelconque. Si $z_u^t = 1$, le nœud associé appartient à la fermeture. Il est possible à partir de cette formulation de dessiner le graphe G. Chaque activité est associée à un sous-ensemble de nœuds (u, t) où $t \in [r_u, d_u - p_u]$. Ces nœuds sont reliés par des arcs de capacité infinie, d'un nœud (u, t) vers son voisin $(u, t-1)$ d'après la contrainte (4.19). Enfin la contrainte (4.20) nous permet de construire les arcs de préséance entre activités : pour toute relation (u, v) dans le diagramme potentiel-tâche, les arcs suivants sont créés : $(u, t - p_u) \rightarrow (v, t) \forall t \in [r_u + p_u; d_u]$.

Tachefine [49] a redémontré après Picard [39] que le problème dual de fermeture à coût maximal est équivalent à un problème de flot maximum. Nous allons le montrer à nouveau avec la formulation du problème.

Formulation duale

Soient $\mu_{u,v}^t$ les variables duales associées aux préséances (4.22), f_u^t celles associées aux tâches (4.19), et α_u^t celles associées aux variables z_u^t (4.23).

Les contraintes (4.21) et (4.22) représentent l'existence ou la non existence de variables associées à une activité suivant que l'on considère une date dans la fenêtre

de temps ou non de l'activité. Elle ne sont pas prisent en compte dans le dual, puisqu'elles n'ont d'influence que sur le nombre de variables du problème primal.

$$\min_{\alpha_u \geq 0} \sum_{u=1}^{N} \sum_{t=r_u}^{d_u - p_u} \alpha_u^t \tag{4.25}$$

$$\text{s. c. :} \quad \tilde{c}_u^t - \sum_{v \in \Gamma_u^+} \mu_u^{t,v} - f_u^{t-1} + \sum_{v \in \Gamma_u^-} \mu_v^{t,u} + f_u^t \leq \alpha_u^t \tag{4.26}$$

$$u = 1, \ldots, N, \quad t \in [r_u; d_u - p_u] \tag{4.27}$$

$$f_u^t \geq 0, \; u = 1, \ldots, N, \quad t \in [r_u; d_u - p_u] \tag{4.28}$$

$$\mu_u^{t,v} \geq 0, \; u = 1, \ldots, N, \; v \in \Gamma_u, \; t \in [r_u; d_u - p_u]. \tag{4.29}$$

Soient les variables suivantes :

$$\mu_u^{t,s} = \sum_{v \in \Gamma_u^+} \mu_u^{t,v} + f_u^{t-1} - \sum_{v \in \Gamma_u^-} \mu_v^{t,u} - f_u^t \text{ si } \tilde{c}_u^t > 0$$

$$\mu_t^{t,u} = \sum_{v \in \Gamma_u^-} \mu_v^{t,u} + f_u^t - \sum_{v \in \Gamma_u^+} \mu_u^{t,v} - f_u^{t-1} \text{ si } \tilde{c}_u^t \leq 0$$

Le problème dual précédent s'écrit alors :

$$\min_{\alpha_u \geq 0} \sum_{u=1}^{N} \sum_{t=r_u}^{d_u - p_u} \alpha_u^t \tag{4.30}$$

$$\text{s. c. :} \quad \tilde{c}_u^t - \mu_u^{t,s} \leq \alpha_u^t \text{ si } \tilde{c}_u^t > 0 \tag{4.31}$$

$$-\mid \tilde{c}_u^t \mid - \mu_T^{t,u} \leq \alpha_u^t \text{ si } \tilde{c}_u^t \leq 0 \tag{4.32}$$

$$f_u^t \geq 0, \; u = 1, \ldots, N, \quad t \in [r_u; d_u - p_u] \tag{4.33}$$

$$\mu_u^{t,v} \geq 0, \; u = 1, \ldots, N, \; v \in \Gamma_u, \; t \in [r_u; d_u - p_u]. \tag{4.34}$$

La solution optimale de ce problème dual est obtenue pour les valeurs de α_u^t suivantes :

$$\alpha_u^t = 0, \quad \text{si } \bar{c}_u^t \leq 0$$

$$\alpha_u^t = \bar{c}_u^t - \mu^{*t,s}_u, \quad \text{si } \bar{c}_u^t > 0,$$

où $\mu^{*t,s}_u$ est la solution optimale du problème PF suivant :

$$\max \sum_{u,t:\bar{c}_u^t>0} \mu_u^{t,s} \tag{4.35}$$

$$\text{s. c. :} \quad \mu_u^{t,s} \leq \bar{c}_u^t \ \text{ si } \bar{c}_u^t > 0 \tag{4.36}$$

$$\mu_T^{t,u} \leq \mid \bar{c}_u^t \mid \ \text{ si } \bar{c}_u^t \leq 0 \tag{4.37}$$

$$\mu_u^{t,s} + \sum_{v\in\Gamma_u^-} \mu_v^{t,u} + f_u^t - \sum_{v\in\Gamma_u^+} \mu_u^{t,v} - f_u^{t-1} = 0 \ \forall u,t : \bar{c}_u^t > 0 \tag{4.38}$$

$$\mu_T^{t,u} + \sum_{v\in\Gamma_u^+} \mu_u^{t,v} + f_u^{t-1} - \sum_{v\in\Gamma_u^-} \mu_v^{t,u} - f_u^t = 0 \ \forall u,t : \bar{c}_u^t \leq 0 \tag{4.39}$$

$$f_u^t \geq 0, \ u = 1,\ldots,N, \ \ t \in [r_u; d_u - p_u] \tag{4.40}$$

$$\mu_u^{t,v} \geq 0, \ u = 1,\ldots,N, \ v \in \Gamma_u, \ t \in [r_u; d_u - p_u] \tag{4.41}$$

$$\mu_u^{t,s} \geq 0, \ \mu_T^{t,u} \geq 0. \tag{4.42}$$

Le programme PF est la formulation d'un problème de flot maximum défini sur un graphe G^e construit comme une extension du graphe G. Le graphe G^e est obtenu par l'ajout d'une source s et d'un puits T à l'ensemble des nœuds N de G. La source s est reliée dans G^e à tout nœud $(u,t) \in N$ de valeur $\bar{c}_u^t > 0$ par un arc (u,t) de capacité égale à $\bar{c}_u^t$. De même tout nœud $(v,t) \in N$ de valeur $\bar{c}_v^t \leq 0$ est relié au puits T dans G^e par un arc $((v,t),T)$ de capacité égale à $\mid \bar{c}_v^t \mid$. Les capacités des arcs $((u,t),(v,t))$ déjà présents dans le graphe G sont considérées infinies.

4.2 Construction et résolution du sous-problème

4.2.1 Utilisation du sous-problème

Le sous-problème est un problème de fermeture à coût maximum sur un graphe. Ce graphe contient un nœud (u, t) pour chaque opération u et chaque temps t.

Il y a 2 types d'arcs :

- les arcs $(u, t) \rightarrow (u, t+1)$ sont associés aux contraintes de continuités
- et les arcs $(u, t) \rightarrow (v, t - p_v)$ sont associés aux contraintes de préséances.

Les coûts $\bar{c}_u^t$ sont placés sur les nœuds (u, t). Le problème de fermeture à coût maximum consiste à choisir un ensemble de nœuds de coût maximum, tel que les successeurs appartiennent aussi à cet ensemble. Les nœuds qui font partie d'un tel ensemble peuvent être séparés des autres nœuds par une coupe dans le graphe.

La coupe sépare les temps où les opérations ne sont pas commencées des temps où les opérations sont commencées ou complétées. Des coûts négatifs incitent à effectuer les opérations tôt dans l'horaire et des coûts positifs incitent à retarder les opérations. Ces coûts sont ajustés à chaque appel du problème maître selon les besoins de ce dernier.

Picard [39] a montré, et nous venons de le voir, que le problème dual du problème de fermeture à coût maximum est un problème de flot maximum. Un nœud source et un nœud puits sont ajoutés au réseau. On construit les arcs suivants :

- Si $\bar{c}_u^t > 0$, alors le nœud source relie le nœud (u, t)
- Si $\bar{c}_u^t < 0$, alors le nœud puits est relié par le nœud (u, t).

où $|\bar{c}_u^t|$ devient une capacité sur ces arcs.

La fermeture recherchée devient une coupe maximale du problème de flot. Elle est donnée par l'algorithme de flot maximum et est interprétée comme l'horaire du projet (dates de début des activités).

4.2.2 Construction du réseau

Il est nécessaire de calculer les fenêtres de temps des activités lorsque celles-ci ne sont pas précisées dans la définition du problème. Il est utile dans tous les cas de recalculer ces fenêtres dans l'intérêt de les resserrer au maximum. La taille des fenêtres joue en effet un rôle primordial dans la taille du sous-problème, et par conséquent dans les temps de calculs de la méthode. De plus, la taille des fenêtres joue un rôle important dans le branchement comme nous le verrons plus loin.

Par défaut, les fenêtres sont toutes identiques à la fenêtre type : $[0, T]$. Il faut par conséquent les calculer pour minimiser leur taille. Le calcul des fenêtres de temps peut se faire dans un premier temps à partir du problème de gestion de projet sans contraintes. Nous verrons plus loin qu'il est plus intéressant de tenir compte des contraintes de ressources afin de les resserrer. Considérons le graphe potentiel-tâche décrit au chapitre 2 pour représenter le projet. Les contraintes de précédences et l'information sur la durée de chaque activité permettent de calculer rapidement les fenêtres de temps. Ce calcul est réalisé avec l'algorithme de Bellman décrit ci-dessous, en parcourant le diagramme potentiel-tâche du puits vers la source, afin de trouver les membres de droite des fenêtres puis dans le sens inverse, pour déterminer les membres de gauche.

Début $r_0 := 0$; <u>marquer</u> le sommet 0 ;

Tant que il existe des sommets non marqués <u>faire</u>

Si il n'existe pas de sommet j non marqué dont tous les prédécesseurs sont marqués

Alors Ecrire : "le graphe a un circuit"

Sinon Soit j un tel sommet ;

$r_j := \max_{i \in \Gamma^-(j)}(r_i + p_i)$;

marquer j

Fin Tant que

Fin

Le calcul des dates de fins est analogue, en partant du nœud puits du diagramme potentiel-tâche.

4.2.3 Modélisation de la date de fin du projet

L'objectif choisi pour tester la méthode est de minimiser la durée du projet ($\min C_{\max}$). Cet objectif est simple à modéliser et offre des jeux d'essais nombreux dans la littérature. Il permet enfin d'interprêter facilement les coûts réduits. Ces derniers sont directement proportionnels aux variables duales du problème maître.

Voici la démarche à suivre pour cet objectif avec notre modèle :

1. créer une activité fictive de durée nulle
2. lui affecter la fenêtre de temps suivante : $[MpMtime; T]$ où $MpMtime$ est le temps de completion du projet à capacité infinie et T , l'horizon du projet (données du problème).
3. affecter à cette activité toutes les activités sans successeurs comme des prédécesseurs.
4. affecter à cette activité le puits comme unique successeur.

5. affecter une consommation nulle sur les ressources en tout temps pour cette activité
6. construire le problème avec cette nouvelle activité

Fonction de coût

La fonction de coût du graphe temporel s'écrit comme suit :

$$c_u^t = 0 \ \forall u = 1, \ldots, N; \ \forall t \in [r_u, d_u - p_u]$$
$$c_{N+1}^t = t \ \forall t = MpMtime, \ldots, T$$

C'est cette modélisation que nous adoptons dans la suite de ce mémoire.

4.2.4 Les algorithmes de flot maximum

Tachefine [49] a adapté des algorithmes de chemins augmentant au problème de calcul de la fermeture maximale sur un graphe. Nous résumons et commentons ici les notions.

Les algorithmes de chemins augmentant sont fondés sur l'idée de déterminer à chaque itération une chaîne augmentante de s à t et d'y acheminer le maximum de flot possible (chemin de la source vers le puits dont la capacité des arcs n'est pas saturée). Un système d'étiquetage sur les nœuds est initialisé par une recherche en largeur d'abord en partant du puits. Elle coûte $O(m)$ où m est le nombre d'arcs. Ensuite, partant du nœud source, l'algorithme construit une chaîne en cherchant des arcs admissibles jusqu'au puits. Une fois la chaîne trouvée, il y fait passer le maximum de flot possible (dépendant de la capacité minimum parmi les arcs de la

chaîne). L'algorithme s'arrête lorsqu'il ne peut plus trouver de chaîne augmentante.

Dans la même approche des algorithmes de chemin augmentant, il existe une famille d'algorithmes dits de Pré-Flot. Ils diffèrent de l'algorithme vu ci-avant, par leur technique de progression du flot le long de la chaîne augmentante. Au lieu d'utiliser le principe de chaîne augmentante, ils cherchent à pousser le flot simultanément à partir de la source jusqu'au puits, d'où le nom de pré-flot qui est maintenu sur chaque arc du graphe. Il existe plusieurs approches pour sélectionner les nœuds d'où il fait passer le flot :

1. sélection des nœuds actifs suivant un ordre "FIFO" (premier entré premier sorti),
2. sélection des nœuds actifs les plus éloignés du puits en priorité (plus grande étiquette de distance).

Le premier choix correspond à l'algorithme de Goldberg appelé *FIFO Preflow Algorithm*. Cet algorithme a une complexité en $O(n^3)$. Le choix numéro 2 conduit à un algorithme développé par Goldberg et Tarjan appelé *Highest Label Preflow Algorithm* et sa complexité est en $O(n^2\sqrt{m})$.

Le but du sous-problème étant de déterminer une coupe minimale sur un graphe, il est inutile de continuer le calcul du flot une fois cette coupe trouvée. Tachefine [49] a donc proposé des améliorations de ces algorithmes pour arrêter la procédure dès que cette dernière a trouvé une coupe minimale.

Difficultés rencontrées

Le graphe doit être connexe (on doit pouvoir rejoindre chaque noeud depuis le

puits et depuis la source), sinon la recherche en largeur d'abord est invalide. C'est pourquoi il est nécessaire de relier tous les nœuds susceptibles d'être des sources ou des puits fictifs (i.e. pas d'arcs entrants, ou pas d'arcs sortants). Ce détail a eu de lourdes conséquences sur l'avancement de nos travaux. La solution que nous proposons est de créer des arcs de capacité nulle reliant la source aux nœuds du graphe qui ne possèdent pas d'arcs entrants et de relier le puits par un nœud n'ayant pas d'arcs sortants. Lors de la recherche, l'étiquetage se fait en remontant les arcs (destination (puits) vers origine (source)) et ainsi tous les nœuds sont accessibles.

D'après Tachefine [49], l'algorithme *Highest Label Preflow Algorithm* peut résoudre des problèmes de fermeture maximale en moyenne en 15 secondes sur des graphes de 150 000 nœuds et environ 1 500 000 arcs. Or ce problème de fermeture intervient comme sous-problème dans le problème de gestion de projet avec contraintes de ressources. Sa résolution rapide conditionne la réussite de cette approche. Avec l'algorithme retenu, le sous problème peut être résolu une centaine de fois en moins de 30 minutes, ce qui laisse un espoir quant au temps de calcul du problème d'une centaine d'activités.

4.3 Construction et résolution du problème maître

4.3.1 Méthode de résolution

Le problème maître est un problème linéaire avec des variables réelles comprises entre 0 et 1. Il peut être résolu par l'algorithme primal du simplexe. Des logiciels comme Cplex (Ilog) résolvent de tels problèmes de façon rapide. Ce problème est du type :

$$\min c^t x$$

$$\text{s.c. } \mathrm{Ax} \leq \mathrm{b}$$

$$\mathbb{1}^t x = 1$$

$$x \geq 0$$

où $A \in \mathbb{N}^{m \times n}$ est la matrice des colonnes générées par le sous-problème. Le nombre de lignes m de la matrice A reste constant et égal au nombre de contraintes de ressources, soit $K \times T$ où K est le nombre de ressources et T l'horizon du projet. Nous prendrons ce nombre de l'ordre de plusieurs centaines pour un problème de 30 activités et 4 types de ressources. Nous verrons qu'il est possible de le réduire par la suite. Le nombre de colonnes est fonction du nombre d'itérations effectuées par le sous-problème. Ce dernier construit une seule colonne par itération. Nous verrons que l'on peut augmenter ce nombre par la suite.

Initialisation du problème

Au départ, le problème maître ne contient pas de colonne et la matrice A est vide. Les variables duales sont donc nulles, et la première colonne construite par le sous-problème est un horaire qui correspond à minimiser la durée du projet sans contraintes de ressources. Cette colonne est introduite dans le problème maître. Si les contraintes de ressources ne sont pas violées, alors la colonne est optimale pour le problème maître et donc le traitement s'arrête. Sinon le problème est infaisable car le flot de cette colonne est contraint égal à 1 puisqu'il n'y a pas d'autres colonnes.

Pour éviter cette impasse dans le démarrage de la génération de colonnes, il faut créer une colonne artificielle pour l'introduire dans les cas d'infaisabilité du problème maître (Cette colonne est construite à partir des contraintes ou rangées du problème maître). Cette colonne ne possède pas de contributions sur les lignes de contraintes, mais elle participe à la contrainte de convexité (4.10). Lors de l'introduction de cette colonne la résolution passe en *Phase I*, c'est à dire que les coûts des colonnes déjà présentes sont annulés et la colonne artificielle se voit affectée d'un coût non nul.

Ce prétraitement vise à accélérer les itérations de la résolution du simplexe. Le seul but étant de minimiser le flot sur la colonne artificielle pour qu'elle ne serve que de variable d'écart sur la contrainte de convexité. Dès que le flot est annulé sur cette colonne (dès qu'il y a suffisamment de colonnes dans le problème maître pour trouver une solution réalisable en nombres réels) la résolution passe en *Phase II*. La colonne artificielle est alors supprimée de la matrice A.

Description des colonnes

Chaque colonne est générée par le sous-problème et doit être vue comme un horaire du projet réalisable par rapport aux contraintes de préséances. On attache par conséquent à chaque colonne une date de début de chaque activité du projet. Si une seule colonne est retenue par le problème maître, son flot sera égal à 1 et ce sera la colonne optimale du problème avec ressources. L'horaire optimal en nombres réels sera celui attaché à cette colonne. Les cœfficients des colonnes sont les consommations d'une ressource à une date donnée de l'ensemble des activités en cours de réalisation du projet. Ces cœfficients sont fabriqués à partir de l'horaire de la colonne et des consommations de chaque activité par unité de temps qui sont données.

4.3.2 Améliorations de la résolution

A première vue, il est souhaitable de pouvoir générer plusieurs colonnes par itérations du sous-problème. Une façon d'accélérer les itérations du problème maître est envisageable grâce aux perturbations sur le membre de droite b des contraintes de ressources. Enfin la *Phase I* peut être évitée si on part avec une colonne réalisable qui serait une borne supérieure du problème. Cette borne supérieure servirait en même temps à resserrer l'horizon du projet en diminuant ainsi le nombre de lignes de contraintes (matrice A). Toutes ces améliorations seront présentées plus loin.

4.4 Description de la méthode de branchement

La recherche arborescente est amorcée lorsque la solution optimale en nombres réels est connue. Elle intervient chaque fois que les itérations de génération de colonne sont arrêtées avec une solution optimale en nombres réels, pour proposer une nouvelle fenètre de temps sur une des activités du projet. La méthode de branchement consiste à détecter parmi les colonnes possédant un flot non nul les activités dont la date de début diffère d'une colonne à l'autre. Ceci signifie que parmi les propositions d'horaire pour cette tâche, il en existe un qui ne fait pas partie de l'horaire optimal. Nous devons donc séparer le problème en divisant les propositions d'horaire pour cette tâche en deux groupes.

4.4.1 La décision de branchement

Un nœud de branchement est créé en choisissant une activité à brancher et un intervalle associé qui possède au moins une date de début proposée par la solution en nombres réels. Un nœud complémentaire est créé simultanément en prenant la même activité et l'intervalle de temps complémentaire de l'intervalle de temps du premier nœud.

La décision revient finalement à choisir une activité u et une date pivot t_u de sa fenêtre de temps $[r_u,\ d_u - p_u]$ autour de laquelle deux intervalles sont crées : $G_u = [r_u,\ t_u]$ et $D_u = [t_u + 1,\ d_u - p_u]$. Cette décision permet de créer deux nœuds de branchement : $N_{G_u} = \{u;\ t \in G_u\}$ à gauche et à droite : $N_{D_u} = \{u;\ t \in D_u\}$.

Il faut pouvoir adopter une méthode de décision efficace afin de choisir judicieusement les nœuds sur lesquels effectuer le branchement. Cette décision porte sur le choix de l'activité et le pivot associé.

Choix de l'activité

Dans un premier temps la décision sera du type "premier candidat" (first fits). Dès qu'une activité possède deux dates de débuts différentes, elle est choisie pour constituer le nœud de branchement. Une méthode plus efficace consisterait à choisir l'activité qui répond à un critère comme :

- Plus large fenêtre,
- Plus grande violation,
- Appartient au chemin critique,
- Plus grand écart entre les dates proposées.

Choix du pivot

La méthode la plus simple est de prendre le deuxième horaire proposé lors du parcours des horaires solutions du problème linéaire, pour le prendre comme pivot. La position du pivot dans la fenêtre n'est pas contrôlée, c'est pourquoi cette méthode s'apparente à un choix aléatoire. Il existe d'autres méthodes plus adaptées au problème, parmi les suivantes :

- médiane : le pivot est la médiane de l'ensemble croissant des horaires pour la tâche considérée.
- barycentre : le pivot est une date calculée avec les flots associés à chaque horaire de la tâche considérée. ($t = \sum_{h \in H} y_h t_h^i$, où i est l'activité choisie, et y_h est le flot associé à la colonne qui propose l'horaire t_h^i pour l'activité i.)

4.4.2 La progression dans l'arbre

Par défaut nous avons choisi la technique de "profondeur d'abord". Plus tard, il sera possible d'appliquer des scores sur les nœuds fabriqués pour orienter la progression sur tel ou tel nœud. Dans cette optique, on peut imposer des coupes au

branchement en anticipant sur les effets des restrictions de fenêtre de temps imposées. Si l'activité choisie appartient au chemin critique et que le nœud de droite engendre un retard qui dépasse la borne supérieure, alors ce nœud doit être éliminé.

Les tests sur les méthodes de branchement sont présentés dans le chapitre 5 (versions B0 à B13). La méthode des médianes (version B5) a été testée une fois pour calculer la date pivot. La méthode du barycentre a fait l'objet de nombreux tests. Quant au tri des activités, nous avons utilisé les fenêtres de plus grande largeur (B0 à B5), de plus petite largeur (B6), de plus grand écart entre les dates (B7), de plus petit écart (B8), ou d'un mélange de deux critères (B9 et B10). Nous avons aussi étudié (B11 et B12) la marge des activités (la marge est définie dans la section 4.9). Toutes ces versions utilisent la méthode la plus performante du programme linéaire présentée dans la section 4.12.

4.5 Les bornes inférieures

Dans le cadre de la résolution de problèmes de projets de durée minimale ($\min C_{\max}$), il est possible de se servir de la solution optimale du problème linéaire en nombres réels égale à Z^*_{lp} comme d'une borne inférieure pour ne générer que des colonnes de coût supérieur ou égal à Z^*_{lp}.

Si la solution optimale en nombres réels est supérieure à la solution du problème sans contraintes de ressources ($Z^*_{lp} > Z_0$), alors il est légitime de supprimer les colonnes de coût inférieur à Z_{lp} car elles ne sont pas solutions du problème en nombres entiers. Ce raffinement du domaine de recherche des solutions nous permet de resserrer la borne inférieure de la fenêtre de temps de la dernière activité du projet de manière à forcer le sous-problème à ne plus reproduire de telles colonnes. En effet les prochains horaires qu'il va générer ne pourrons faire démarrer la dernière activité plus tôt que Z^*_{lp} (la date de fin du projet), par conséquent le coût de la

colonne associée sera supérieur ou égal à Z^*_{lp}.

Remarquons qu'au premier abord, le fait de retarder la dernière tâche n'impose aucunement les fenêtres de temps des autres activités du projet et revient donc à pénaliser les colonnes de coût inférieur à Z^*_{lp} par un coût égal à Z^*_{lp} sans avoir à les interdire de la liste des colonnes candidates. En pratique, cette méthode devrait néanmoins permettre au sous-problème de ne plus générer des colonnes de trop faible coût car ces dernières étant susceptibles de violer fortement les contraintes de ressources (contrairement à des colonnes où les activités sont plus étalées dans le temps) la combinaison des variables duales et de la pénalité devrait les rendre indésirables au problème maître.

Cette méthode d'épuration des colonnes et de resserrage de la dernière fenêtre de temps, si elle s'avère efficace, peut être exécutée à plusieurs reprises à partir de la première solution en nombres réels.

Au fur et à mesure que des nouvelles solutions linéaires sont trouvées, la fenêtre va se resserrer et les colonnes générées auront un coût plus proche de la colonne optimale.

Cette méthode nous assure de faire progresser la borne inférieure de Z_0 à une valeur $Z^*_{lp_i}$ où i représente la $(i-1)$ème itération du problème linéaire. Il reste à prouver que cette borne reste bien une borne inférieure du problème en nombres entiers.

4.5.1 Démonstration de la validité de la méthode

Par hypothèse, nous admettons sans le démontrer que $Z^*_{lp_0} \leq Z^*$ (hypothèse $\mathcal{H}_0$), où $Z^*_{lp_0}$ est la solution en nombres réels du problème de la première itération donc sans aucune restriction sur les colonnes noté P_0. (ceci est vrai puisque le

problème linéaire en nombres réels est une relaxation du problème en nombres entiers.) Soit P_1 le problème linéaire en nombres réels pour la deuxième itération de notre méthode. Si P_1 admet une solution optimale $Z^*_{lp_1}$, cette valeur est elle une borne inférieure de Z^* ?

Soit l'hypothèse à démontrer $\mathcal{H}_1$: $Z^*_{lp_0} \leq Z^*_{lp_1} \leq Z^*$. La solution du problème P_1 est une combinaison convexe de colonnes pour minimiser la durée du projet notée $Z^*_{lp_1}$. $Z^*_{lp_1} \geq Z^*_{lp_0}$ par construction du nouveau polyèdre des colonnes. Appelons R_1 l'ensemble des colonnes réalisables pour le problème maître en nombres entiers. Ces colonnes réalisables ont un coût supérieur à $Z^*_{lp_1}$ qui est obtenue en relaxant l'intégrité. Or Z^*, la colonne optimale en nombre entier du problème, fait partie de cet ensemble R_1. Donc son coût est supérieur ou égal à $Z^*_{lp_1}$. Donc $\mathcal{H}_1$ est vraie.

En supposant que l'hypothèse $\mathcal{H}_p$: $Z^*_{lp_{p-1}} \leq Z^*_{lp_p} \leq Z^*$ est vraie, qu'en est-il de l'hypothèse $\mathcal{H}_{p+1}$?

La solution en nombres réels $Z_{lp_{p+1}}$ est une combinaison convexe de colonnes pour minimiser la durée du projet. L'ensemble des colonnes candidates définit un polyèdre dans lequel se trouve un sous ensemble R_{p+1} qui contient les colonnes réalisables en nombres entiers. Ce sous-ensemble est construit comme une restriction de R_p aux colonnes de coût supérieur ou égal à $Z^*_{lp_p}$. Dans ce domaine de recherche, la relaxation linéaire du problème égale à $Z^*_{lp_{p+1}}$ est par définition meilleure que les colonnes de l'ensemble R_p (elles sont plus contraintes), donc $Z^*_{lp_p} \leq Z^*_{lp_{p+1}}$. Or $Z^*_{lp_p} \leq Z^*$ donc il n'existe pas de colonnes solution entières appartenant à l'intersection de R_p et R_{p+1}, donc la colonne optimale en nombres entiers du problème appartient à R_{p+1}. Or $Z^*_{lp_{p+1}}$ est la solution du problème de gestion de projet sans les contraintes d'intégrités dont la solution est Z^* donc $Z^*_{lp_{p+1}} \leq Z^*$. Par conséquent $\mathcal{H}_{p+1}$ est vraie. Par récurrence sur p, il est légitime de généraliser la preuve pour toutes les itérations de la méthode.

4.5.2 Implémentation

La méthode consiste à tester la première solution linéaire en nombres réels avec la colonne solution du problème sans contraintes. Si $Z^*_{lp} > Z_0$ alors le processus d'élimination des colonnes de coût inférieur à Z^*_{lp} peut être lancé. La fenêtre de temps de la dernière activité u est ensuite resserrée ($r_u = Z^*_{lp}$) de manière à retarder la fin du projet à une date supérieure ou égale à Z^*_{lp}. Il ne faut pas interdire la date Z^*_{lp} car elle est peut être optimale en nombres entiers. Tant que le processus est réalisable, il est possible de le réitérer. Le critère d'arrêt de ces itérations peut être un test sur le progrès effectué sur la borne inférieure obtenue ou bien sur un nombre d'itérations maximum.

Remarque :

Dans le cas de la minimisation de la durée du projet avec des données entières, la solution optimale est entière donc il est possible d'arrondir à l'entier supérieur la borne LP trouvée. ($r_u = \lceil Z^*_{lp} \rceil$ où u est l'activité fictive de fin du projet).

4.5.3 Tests numériques

Protocole expérimental

Les tests ont été effectués sur les 480 problèmes de Kolisch avec 30 activités et 4 ressources. Les itérations sont répétées tant que la solution linéaire croît d'un écart minimum de 0.001 et que le nombre d'itérations ne dépasse pas 20. Le gain sur la borne inférieure coûte cher en temps de calcul étant donné qu'il engendre beaucoup d'itérations pour certains problèmes. Mais la borne inférieure est souvent améliorée pour réduire jusqu'à 90% la valeur du "gap" (écart entre Z_0 et Z^*).

Représentations graphiques

La figure 4.2 représente l'écart entre Z_{lp_0} et Z_{lp} pour les 110 problèmes les plus difficiles de KSD (voir Chapitre 5).

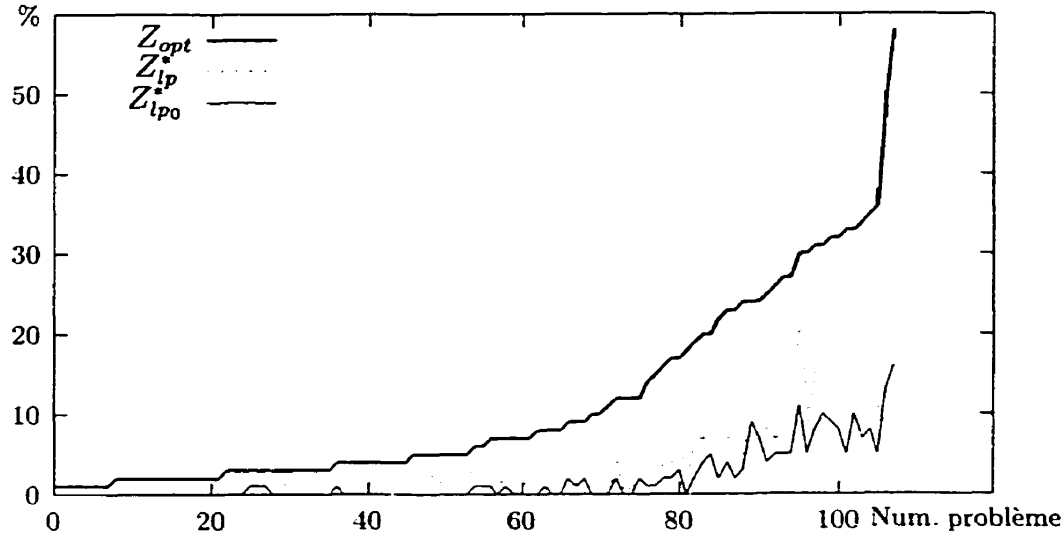


Figure 4.2: *Amélioration de la borne inférieure*

La ligne de base est la valeur de la durée du projet pour le problème sans contraintes de ressources (solution du diagramme potentiel-tâche). Les valeurs reportées sur les courbes sont les écarts entre les makespan des solutions trouvées par la méthode et cette valeur de base. La courbe supérieure en traits pleins représente la valeur de Z_{opt} pour chaque problème tandis que la courbe inférieure en trait plein représente les valeurs de Z_{lp_0}. La courbe en traits pointillés qui représente la nouvelle solution Z_{lp}, est toujours au dessous de la courbe de la solution optimale en nombres entiers, mais elle est globalement au dessus de la courbe des solutions Z_{lp} dessinée dans le graphe ci-dessus. Ce qui montre la progression de la borne inférieure. Des résultats plus détaillés figurent dans le chapitre 5 où la technique présentée est appliquée pour les versions V4 et V12 de la méthode.

4.5.4 Extension de la méthode

La méthode a été accélérée et améliorée grâce à la nature des problèmes qui veut que la valeur de l'objectif soit un entier. En effet, la borne inférieure donnée par la relaxation linéaire est arrondie à l'entier immédiatement supérieur pour resserrer la fenêtre et réévaluer les colonnes.

Dans un but de dimunution du temps de calcul, il est également possible d'adopter la méthode de borne inférieure en l'arrêtant plus tôt que dans les tests. Mais il est possible aussi d'imposer cette modification au sous-problème si on connaît déjà une borne inférieure meilleure que le plus long chemin du diagramme potentiel-tâche (solution optimale sans contraintes de ressources). Il suffit de s'inspirer des heuristiques qui recherchent une borne inférieure pour effectuer le calcul avant de définir les fenêtres de temps.

La meilleure heuristique actuelle est celle de Mingozzi et al. [33] dont voici l'algorithme :

Soit L la liste des activités non ordonnancées ordonnées dans l'ordre croissant de leur nombre de "sœurs". Où une "sœur" pour une activité i est une activité j telle que i et j peuvent être ordonnancées en parallèle sans violation de contraintes de précédences ou de ressources.(dans l'ordre croissant de leur durée p_i).

Initialiser LB := date de fin au plus tôt des activités en cours de réalisations

Tant que L est non vide

- Sélectionner la première activité j de la liste L
- LB := LB + p_j
- Enlever l'activité j et ses "sœurs" de la liste L

Fin tant que

Nous n'avons pas effectué de tests avec cette borne mais les heuristiques définies dans la revue de littérature (cf. 3.1.2) ont pu être reprises pour construire les bornes LB1 et LB2 ci-dessous.

4.5.5 Proposition : calcul des fenêtres avec les bornes LB1 et LB2

Le calcul des fenêtres de temps pour chaque activité a jusqu'à maintenant été effectué en ne tenant compte que des contraintes de préséances. Il est possible de tenir compte des contraintes de ressources, en s'inspirant de la borne inférieure fondée sur le chemin critique, définie par Stinson [47]. Celle-ci a été présentée dans la revue de littérature de ce mémoire (cf. 3.1.2).

Voici une adaptation pour le resserrage des fenêtres avec la borne LB2 :

Initialiser i à la deuxième activité du chemin critique

Initialiser $\Pi K_{k,t}$ à R_k pour toute date t et type de ressource k

Tant que l'activités $i \neq$ Puits **faire**

Choisir j, un prédécesseur de i sur le chemin critique

Pour chaque valeur de t de l'intervalle $[r_j, r_j + p_j]$

$\Pi K_{k,t} = \Pi K_{k,t} - r_{kj}$ pour tout type de ressource k

Fin pour

Pour chaque u prédécesseur de i différent de j, **faire**

Pour chaque type de ressource $k = 1, \ldots, K$

Initialiser $e_u^k = 0$

Pour chaque date t_u de l'intervalle $[r_u,\ r_i - p_u]$, **faire**

Pour chaque t de l'intervalle $[t_u,\ t_u + p_u]$, **faire**

Si $\Pi K_{k,t} \geq r_{ku}$ **Alors** incrémenter e_u^k de 1

Fin pour

Fin Pour

$r_i = \max(r_i,\ r_i + p_u - e_u^k)$

Fin pour

Fin pour

Choisir un nouveau i successeur de l'activité i sur le chemin critique

Fin Tant que

Cet algorithme permet de resserrer toutes les fenêtres de temps des activités dont les prédécesseurs peuvent entrer en conflit de ressource. Une amélioration de cette méthode, serait de parcourir chaque chemin critique, car il peut exister plusieurs chemins critiques (au plus $n!$).

Il peut enfin être plus efficace si on calcule une borne supérieure du profil de consommation de ressource, au fur et à mesure que l'on progresse dans le chemin critique.

Dans ce cas, la disponibilité des ressources dépend des activités du chemin critique dont on fixe les dates de débuts à leur date de début au plus tôt mais aussi des activités qui permettent de calculer la borne inférieure. On ne peut pas définir leur date de début, mais on peut connaître la consommation moyenne qu'elles vont engendrer. Si la consommation de toutes les activités antécédantes à une activité du chemin critique est supérieure à la disponibilité de la ressource, alors le chemin critique est trop court. Ceci définit une borne inférieure fondée exclusivement sur les ressources (appelée borne LB1).

La borne fondée sur les ressources se calcule en faisant une recherche en largeur d'abord sur le diagramme potentiel-tâche en partant du puits visant à calculer la consommation cumulée dans le temps de chaque type de ressource des prédécesseurs de l'activité courante. Notons $\bar{r}_i^k$ la consommation cumulée de la ressource k associée à l'activité i.

Algorithme de calcul de la borne inférieure LB1 :

Initialiser $\bar{r}_i^k$ à 0 pour toute activité i et tout type de ressource k

Initialiser D et $\bar{D}$ à l'ensemble vide.

Pour chaque activité i **faire**

Poser a=i

Tant que a est non nul

Fabriquer D l'ensemble des prédécesseurs de i

Tant que D est non vide **faire**

Choisir j dans D et l'en sortir ($D := D - \{j\}$)

Si j n'est pas marqué

- marquer j
- ajouter j à $\bar{D}$
- $\bar{r}_i^k = \bar{r}_i^k + r_j^k * p_j$

Fin de si

Fin tant que

Choisir une activité l dans $\bar{D}$ et l'en sortir

Poser a = l

Fin tant que

Fin pour

La borne inférieure s'exprime ainsi :

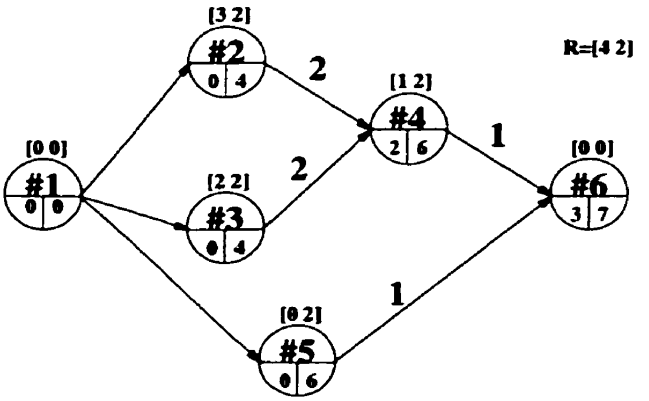
$$r_i = \lceil \max_{k=1,\ldots,K} \{ \frac{\bar{r}_i^k}{R_k} \} \rceil$$

pour toute activité i du projet.

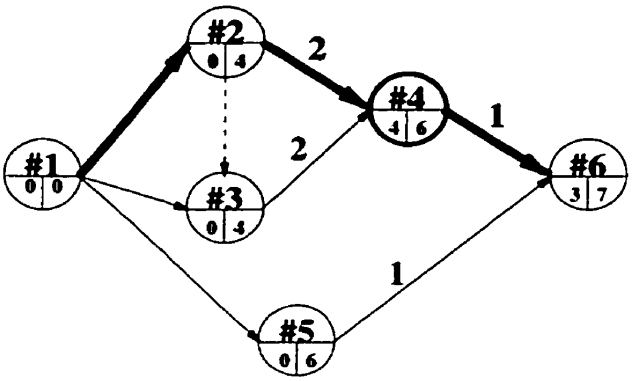
Exemple :

Sur la figure 4.3 est présenté un exemple de projet avec 4 tâches et 2 ressources. La première étape de l'algorithme consiste à calculer le chemin critique du projet. Les traits gras représentent le chemin critique $\{1, 2, 4, 6\}$. En sélectionnant l'activité 4, il apparaît un conflit de ressources entre les activités 2 et 3. Ce qui suggère d'après la borne inférieure de Stinson de retarder le chemin critique de 2 unités de temps. Donc le membre de gauche de la fenêtre de temps de l'activités 4 passe de la valeur 2 (ancienne valeur du chemin critique) à la valeur 4 (chemin critique + durée du conflit). Cette première exploration conduirait à trouver une borne inférieure pour le problème égale à 5. Mais en prenant compte de la borne inférieure sur les ressources, il est possible de constater que la consommation en ressources des activités 2, 3 et 4,

entre l'instant 0 et l'instant 5 vaut 10, ce qui est égal à la valeur de la disponibilité de la ressource numéro 2. Donc aucune activité qui consomme cette ressource ne peut être ordonnancée entre les dates 0 et 4. Donc, l'activité 5 ne peut pas être ordonnancée entre les instants 0 et 4 ce qui va retarder le chemin critique courant d'une unité de temps (la durée de l'activité 5). Donc le nouveau membre de gauche de la fenêtre de temps de l'activité 6 est égal à 6.



Tache #4, Ressource 1



Tache #6, Ressource 2

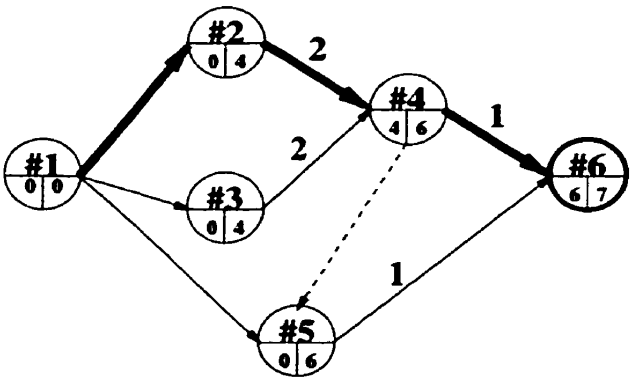


Figure 4.3: *Exemple de calcul de borne inférieure*

4.5.6 Essais numériques de la borne LB1

La borne LB1 définie sur les ressources apporte beaucoup d'économie en terme d'itérations et de temps de calcul. Ses effets sont répercutés sur le sous-problème en diminuant la taille du graphe, et sur le problème maître en resserrant la borne inférieure. La taille de l'ensemble des colonnes admissibles est donc réduite mais le nombre d'itérations s'en voit augmenté. En effet les problèmes sont résolus à l'optimalité en nombre réels, avec davantage d'itérations que lorsque le domaine est plus large (sans LB1).

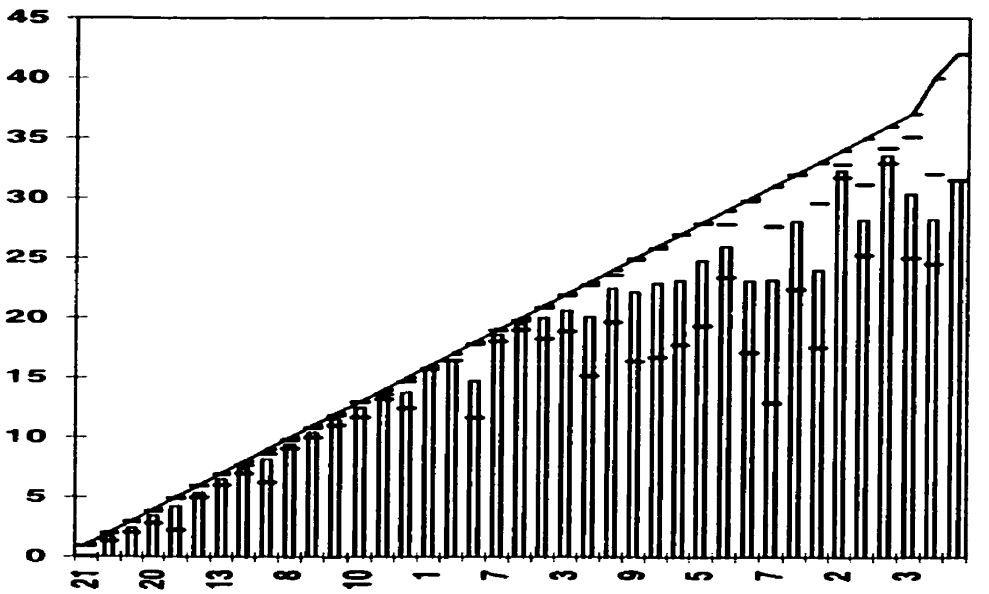


Figure 4.4: *Performance de la borne inférieure LB1*

Le graphe 4.4 représente la différence entre le gap de la méthode de base et celui obtenu avec l'heuristique LB1. En abscisse figure la distribution des problèmes en fonction de la valeur croissante de l'écart d'intégrité associé à la borne inférieure Z_0. Ce dernier varie entre 0% et 41 %. La distribution du nombre de problème en fonction des valeurs de l'écart varie entre 1 et 21 problèmes. Les rectangles représentent la valeur moyenne de l'écart d'intégrité de la solution avec la borne inférieure LB1 pour chaque ensemble de la distribution (lire les valeurs de l'écart en ordonnées). Les petites barres horizonnales représentent la valeur minimale et la valeur maximale de l'écart avec la borne LB1 en fonction des problèmes du sous-ensemble considéré. La droite représente la valeur de l'écart d'intégrité sans la borne LB1. Comme le montre ce graphique, la borne LB1 joue un rôle important pour les jeux d'essais dont l'écart entre la borne inférieure Z_0 et la solution optimale est supérieur à 21%. De plus amples détails sont présentés au chapitre 5 où les versions V11 et V12 adoptent l'heuristique LB1.

4.6 Les bornes supérieures

4.6.1 Heuristique pour calculer la borne supérieure

La borne supérieure de la méthode de base est l'horizon du projet c'est à dire la date ultime à laquelle le projet doit être terminé. Cette valeur est parfois très grande, car elle est calculée comme le plus long chemin du diagramme potentiel-tâche sans fenêtres de temps. Il est donc intéressant de calculer une borne supérieure du problème avant de commencer les calculs par génération de colonne. Nous présentons ici une méthode qui est satisfaisante puisqu'elle assure un résultat dont l'écart avec la valeur optimale est inférieur à 10% de cette dernière et ceci dans des temps de calculs très courts (quelques secondes pour un problème de 30 activités et 4 ressources).

La méthode la plus efficace est une méthode dite de liste ou sérielle. Elle fut proposée par Kelley en 1963. Une itération de cette méthode consiste à sélectionner

une activité parmi les J du projet et à l'ordonnancer à la date la plus tôt pour laquelle elle ne viole pas les contraintes de ressources. Cette itération est répétée tant qu'il reste des activités non ordonnancées.

Les critères de sélection des activités les plus performants sont les suivants :

- LST(latest start time)départ au plus tard : $j^* = \min_j\{v(j) = d_j - p_j\}$,
- LFT(latest finish time)fin au plus tard : $j^* = \min_j\{v(j) = d_j\}$.

où $v(j)$ est la fonction qui évalue le degré de priorité de l'activité j pour être ordonnancée.

Il est possible aussi de prendre le critère MTS (most total successors) maximum de successeurs. Ce critère permet de ne pas avoir à calculer les parties droites des fenêtres de temps. Mais ses performances sont moins intéressantes.

Algorithme de la méthode sérielle

Avant de présenter l'algorithme, il est nécessaire de définir les variables utilisées.

J :	nombre d'activités du projet
n :	itération courante
S_n :	ensemble des taches ordonnancées à la $n^{\text{ième}}$ itération
D_n :	ensemble des activités candidates à l'itération n
$\Pi K_{r,t} : t = 1, \ldots, T$ $r = 1, \ldots, R$	profil des ressources sur l'horizon du projet
$v(.)$:	fonction de règle de décision.

L'algorithme de la méthode comme il l'est présenté par R. Kolisch [29] est décrit ci-après.

Heuristique de calcul de borne supérieure :

Initialisation : $n := 1, \quad S_n := \emptyset$;

Tant que $|S_n| < J$ **Faire** l'itération n

Début – Construire D_n et $\Pi K_{r,t}, \ t = 1, \ldots, T, \ r \in R$;

- $j^* := \min_{j \in D_n}\{j; v(j) = \inf_{i \in D_n}\{v(i)\}\}$;
- $r_j^* := \max\{t_i + p_i; i \in \Gamma^-(j^*)\}$;
- $t_j^* := \min\{t; \ r_{j^*} \leq t \leq d_{j^*} - p_{j^*} + 1, \ k_{j^* r} \leq \Pi K_{r,\tau}, \ \tau = t, \ldots, t + p_{j^*} - 1, \ r \in R\}$;
- $S_{n+1} := S_n \cup \{j^*\}$;
- $n := n + 1$;

Fin Tant que

Arrêt

Remarque : $\Gamma^-(j)$ représente l'ensemble des prédécesseurs de l'activité numéro j.

Avantages de la méthode

Dans un premier temps la méthode heuristique permet de trouver une borne supérieure pour l'objectif de l'ordre de 10% au dessus de la valeur de l'objectif optimal si la fonction objectif est la minimisation de la durée du projet. Cette borne supérieure peut servir pour plusieurs paramètres de construction du problème de décomposition :

- cette borne supérieure définit un nouvel horizon du projet, donc il contribue à diminuer le nombres de lignes de contraintes du problème maître.
- c'est aussi une borne pour les fenêtres de temps des activités dont les membres de droites peuvent être recalculés. Ceci diminue le nombre de nœuds associés au graphe du sous-problème.

- le résultat de l'heuristique fournit un horaire réalisable, donc une colonne solution des contraintes du problème maître. Introduire cette colonne dans le problème maître permet de rester réalisable dès les premières itérations et de ne plus avoir recours à la *Phase I* (colonne artificielle) pour démarrer des itérations.
- la solution peut être optimale ce qui économise une itération du sous-problème. (le problème maître va arrêter les itérations).

Lors de tests présentés dans le chapitre 5, nous appellerons V0 la version de base de la méthode de décomposition où la seule borne supérieure utilisée sera l'horizon T donné dans les jeux d'essais. Toutes les autres versions utiliseront la méthode de borne supérieure présentée ci-avant.

4.6.2 Une méthode de coupe : calcul dynamique de la borne supérieure

Principe d'application de la coupe

L'idée est de rechercher une solution réalisable pour s'en servir de borne supérieure. Cette recherche peut s'effectuer à chaque itération du sous-problème en vérifiant pour chaque itération si la colonne générée est réalisable en nombres entiers. Si c'est le cas c'est donc une nouvelle borne supérieure. Dans le calcul de la date de fin du projet, la date de début de l'activité fictive "puits" devient donc le nouvel horizon sur lequel les fenêtres de temps peuvent être calculées.

Ce resserrage des fenètres peut éliminer permet de réduire le domaine de branchement. Il faut par conséquent vérifier que les noeuds de branchement non encore explorés sont toujours compatibles avec le nouveau fenétrage. Pour palier à

ce problème, il suffit "délaguer" le branchement que l'on veut explorer pendant le retour en arrière.

Du point de vue de l'implémentation, il faut détecter la coupe avant d'évaluer le nœud. Autrement dit, il faut pouvoir détecter que le fenêtrage du nœud que l'on envisage d'explorer est incompatible avec la nouvelle borne que l'on s'est fixée. Pour cela deux conditions doivent être vérifiées :

1. le nœud à brancher appartient au chemin critique,
2. le début de sa nouvelle fenêtre est supérieur à la date de début de la colonne qui constitue la borne supérieure.

Si ces deux conditions sont vérifiées alors il faut couper la branche ; sinon, le nœud est explorable.

Appartenance d'une tâche au chemin critique

Il suffit de calculer les marges des activités sur les fenêtres des successeurs pour identifier les tâches critiques. Celles dont les marges sont nulles sont des tâches critiques. La marge de l'activité i s'écrit $m_i = (T_i - t_i)$ avec :

$$t_i = r_i + p_i$$

$$T_i = \min_{j \in \Gamma_i^+} r_j$$

Essais numériques

Dans la pratique, les conditions à vérifier pour couper une branche reviennent à vérifier si la fenêtre proposée par la décision est incluse ou non dans les nouvelles

fenêtres calculées avec la nouvelle borne supérieure. Apparemment, les gains sont importants, puisque pour un problème de 4 activités et 2 ressources, le nombre d'itérations passe de 139 à 55 avec seulement 14 nœuds de branchement au lieu de 44 sans cette réduction. Dans la pratique, il apparaît que pour les gros problèmes (plus de 20 activités et 4 ressources) cette borne supérieure n'est détectée que tardivement dans l'exploration. Notre travail s'est donc concentré davantage sur une amélioration de la résolution du problème linéaire avant d'implémenter de telles coupes du branchement.

4.7 Diminution du nombre d'itérations

Dans le but d'accélérer le programme linéaire, nous avons mis au points différentes techniques fondées sur la structure du problème maître et sur l'interface entre le problème maître et le sous-problème.

4.7.1 Exploitation du problème maître

Observation :

Les variables duales ne tiennent pas compte de la durée des activités dont l'horaire engendre les violations de contraintes.

Autrement dit, si une activité viole une contrainte, le coût marginal résultant de cette violation est affecté sur une seule variable duale (sur une des lignes du problème maître) de façon non contrôlée. Les variables duales des contraintes voisines elles aussi saturées prennent la valeur 0. Or la position de cette valeur est à l'origine du décalage des dates de début des activités. Il peut donc être intéressant de synchroniser ce décalage avec les durées des activités concernées.

Proposition : perturbation des membres de droite

Seule la variable duale de la ligne de contrainte qui se trouve saturée en premier lors de la résolution du problème linéaire prend une valeur non nulle. Donc, si on force les lignes les plus tardives à être saturées en premier, ce sont elles qui prendront la variable duale et les autres lignes saturées elles aussi mais correspondant à une date inférieure auront des variables duales associées nulles. Ceci est obtenu en diminuant légèrement les quantités de ressources disponibles durant les différentes périodes avec des perturbations qui augmentent avec la date. Du point de vue du sous-problème, une tâche engendrant une saturation, et donc une variable duale, sera retardée jusqu'à la date de la dernière ligne saturée. Donc, elle sera décalée d'une durée supérieure ou égale à sa durée de traitement. Les contraintes de ressources suivantes :

$$t_0 : r_0^0 y_0^0 + r_1^0 y_1^0 + ... \leq R$$
$$t_1 : r_0^1 y_0^1 + r_1^1 y_1^1 + ... \leq R$$
$$...$$
$$t_H : r_0^H y_0^H + r_1^H y_1^H + ... \leq R$$

deviennent :

$$t_0 : r_0^0 y_0^0 + r_1^0 y_1^0 + ... \leq R - \epsilon_0$$
$$t_1 : r_0^1 y_0^1 + r_1^1 y_1^1 + ... \leq R - \epsilon_1$$
$$...$$
$$t_H : r_0^H y_0^H + r_1^H y_1^H + ... \leq R - \epsilon_H$$

où

$$0 \leq \epsilon_0 < \epsilon_1 < ... < \epsilon_H << R$$

Preuve d'optimalité

Il reste à prouver que ce nouveau problème maître est équivalent au problème initial.
Soit (P) le problème initial :

$$\min c^T x$$

$$Ax - b \leq 0$$

$$\mathbb{1}^T x - 1 = 0$$

$$x \geq 0$$

c est le vecteur de IR^n des coûts des n colonnes introduites dans le problème maître.

b est le vecteur de IR^m où $m = K \times T$ des disponibilités par unité de temps des K ressources sur l'horizon T.

A est la matrice de $IR^{m \times n}$ des colonnes introduites dans le problème.

b est le vecteur $\mathbb{1}$ représente le vecteur de IR^n constitué de 1 partout.

Soit x^* un optimum **régulier** de (P), c'est à dire tel que le problème suivant admette une solution de valeur nulle :

$$\min c^T y$$

$$Ay \leq 0$$

$$\mathbb{1}^T y = 0$$

Alors pour ce minimum x^*, il existe des multiplicateurs $\lambda \in IR$ et $\mu \in IR^m$ satisfaisant aux conditions dites de Lagrange Kuhn Tucker qui s'écrivent :

$$c + \lambda^* \mathbb{1} + \mu^{*T} A = 0$$

$$\mu^* \geq 0$$

$$\mu^{*T}(Ax^* - b) = 0$$

$$Ax^* - b \leq 0$$

$$\mathbb{1}^T x^* - 1 = 0$$

Il est possible d'aller plus loin en remarquant que les gradients des fonctions de contraintes $\{\mathbb{1}$ et $A_i; i \in I(x^*)\}$ sont linéairement indépendants.

Cette condition de normalité qui assure l'existence d'un ensemble unique de multiplicateurs bornés est la condition de Mangasarian-Fromovitz qui s'écrit pour (P) :

$\mathbb{1}$ est non nul

il existe un $y^* \in \mathbb{R}^n$ tel que :

$$A_i y^* < 0 \; i \in I(x^*)$$

$$\mathbb{1}^T y^* = 0$$

où I est l'ensemble des indices de contraintes saturées.

Cette condition assure que, pour tout $\epsilon > 0$, le programme perturbé (Pmf_ϵ) suivant :

$$\min c^T y$$

$$A_i y \leq -\epsilon \;\; i \in I(x^*)$$

$$\mathbb{1}^T y = 0$$

est réalisable. On peut montrer qu'il est aussi fini.

Nous allons maintenant utiliser ces résultats pour montrer que le problème perturbé associé à (P) admet une solution finie proche de x^*.

Soit (P_ϵ) le problème perturbé :

$$\min c^T x$$
$$Ax - b \leq \epsilon$$
$$\mathbb{1}^T x - 1 = 0$$
$$x \geq 0$$

Pour le problème (P_ϵ) les conditions de Lagrange Kuhn Tucker s'écrivent :

$$c + \lambda \mathbb{1} + A^T \mu = 0$$
$$\mu \geq 0$$
$$\mu^T (Ax^* - b + \epsilon) = 0$$
$$Ax^* - b + \epsilon \leq 0$$
$$\mathbb{1}^T x^* - 1 = 0$$

Ceci prouve que pour les lignes de contraintes pour lesquelles la contrainte n'est pas saturée, les conditions sont équivalentes au premier problème (P) et donc les valeurs x^* , λ^* et μ^* ne vont pas être modifiées par ces contraintes. Il faut en revanche étudier le cas des contraintes suivantes :

$$I(x^*) = \{i; (Ax^* - b)_i = 0\}$$

où I est l'ensemble des indices correspondant aux contraintes d'inégalités saturées par cette solution.

Définissons un chemin implicite :

$$x(t) = \varphi^{-1}(\xi(t)) \quad x(0) = x^*$$

où l'application $\varphi(.) : IR^n \rightarrow IR^n$ est définie comme suit :

$$\varphi_j(x) = \mathbb{1}^T x$$

$$e_j^T x \quad j \in K$$

et ξ son inverse :

$$\xi(t) \in R^n \quad \xi(0) = \varphi(x^*) = \xi^* \quad t \in (0, t^*)$$

à tout chemin $x(t)$ correspond le chemin $\xi(t) = \varphi(x(t))$.
d'où :

$$c^T x(t) = c^T x(0) + t c^T x'(0) + o(t)$$

soit $y \in IR^n$ le chemin implicite $x(.)$ associé a y permet d'écrire :

$$c^T x(t) = c^T x^* + t c^T y + o(t)$$

et

$$Ax(t) = Ax^* + tAy + o(t)$$

D'après ces résultats le problème (P_ϵ) peut se réécrire :

$$\min c^T y$$

$$tAy + \epsilon \leq 0$$

$$\mathbb{I}^T y = 0$$

Quel que soit ϵ, pour t assez petit, on a $tA_i y^* < A_i y^* \leq -\epsilon$, $\forall i \in I(x^*)$ où y^* est la solution optimale du problème (Pmf_ϵ).

Donc y^* est la solution optimale du problème (P'_ϵ) restriction de (P_ϵ) aux contraintes saturées de (P).
(P'_ϵ)

$$\min c^T x$$

$$A_i x - b \leq -\epsilon \quad i \in I(x^*)$$

$$\mathbb{I}^T y = 0$$

$x(t^*) = \xi(t^*)$ est la solution optimale de (P'_ϵ).

Or nous avons vu que les contraintes non saturées restent insaturées au voisinage de x^*. Donc il est possible de construire une solution optimale pour (P_ϵ) avec la solution optimale $x(t^*)$ de (P'_ϵ). Donc (P_ϵ) admet une solution finie. Mais quelle est sa valeur comparée à x^* ?

D'après les conditions d'optimalité de Lagrange Kuhn Tucker, les contraintes saturées de (P) ne sont plus saturées par la solution de (P_ϵ). Les conditions sont :

$$0 < t < 1 \text{ et } \epsilon \geq 0$$

L'écart sur l'objectif est :

$$c^T x(t^*) = c^T x^* + t^* c^T y^* + o(t^*)$$
$$c^T x(t^*) - c^T x^* \leq c^T y^* + o(t^*)$$

avec $c^T y^* \to 0$ lorsque $\epsilon \to 0$.

Donc la perturbation ne vient modifier que de façon négligeable le résultat numérique de la fonction objectif.

En généralisant ces écritures, il est possible de choisir un vecteur de perturbations ϵ distribué sur l'ensemble des contraintes d'inégalités et qui respecte les conditions d'optimalité. Le choix peut donc être modifié en fonction du profil des multiplicateurs μ souhaité, sachant que pour chaque vecteur ϵ correspond un unique vecteur μ, que l'on peut calculer grâce à l'égalité de Lagrange Kuhn Tucker :

$$\mu^T(Ax^* - b) = 0$$

Ainsi dans la pratique, si deux contraintes saturées j et k s'écrivent à l'optimalité :

$$\sum_{i=1}^{n} a_{ki}x_i^* = b$$

$$\sum_{i=1}^{n} a_{ji}x_i^* = b$$

Il est correct de choisir une perturbation $\epsilon = (\epsilon_i)_{1 \leq i \leq m}$ pour s'assurer que la contrainte k soit saturée et pas la contrainte j en affectant à k et j, les nouvelles contraintes :

$$\sum_{i=1}^{n} a_{ki}x_i \leq b - \epsilon_k$$

$$\sum_{i=1}^{n} a_{ji}x_i \leq b - \epsilon_j$$

$$\text{avec } 0 \leq \epsilon_k < \epsilon_j << b$$

Résultats numériques

En suivant les directives proposées nous avons testé la résolution de 480 problèmes de 30 activités et 4 ressources générés aléatoirement, en perturbant les contraintes de manière à saturer les contraintes les plus tardives avec $\epsilon = 1e^{-10}$.

Nous avons observé les résultats des 200 tests les plus significatifs parmi 480 (les 280 restants sont résolus à l'optimalité par la relaxation linéaire très rapidement).

Tableau 4.1: *Résultats des tests avec et sans perturbations*

ΔItérations			**Δ Flots**		
min	max	moyenne	min	max	moyenne
-62	54	1.17	-9	4	0
-50 %	36 %	0.96 %	-70 %	44 %	-0.31 %

Le tableau 4.1 représente l'écart du nombre d'itérations et l'écart du nombre de colonnes fractionnaires pour 200 problèmes différents de 30 activités et 4 ressources. Ces problèmes ont étés résolus sans perturbation ($\epsilon = 0$), puis avec une perturbation ($\epsilon = 1e^{-10}$). Les écarts calculés prennent pour référence la résolution sans perturbations. Par exemple l'écart minimum de -62 pour les itérations est la différence entre le nombre d'itérations sans perturbations et avec perturbations, autrement dit il y a un problème parmi les 200 testés pour lequel la perturbation a coûté 62 itérations de plus que la méthode sans perturbations. L'écart sur le flot montre qu'en moyenne, la solution linéaire possède le même nombre de colonnes fractionnaires avec ou sans perturbations.

Conclusion

Le résultat n'est pas prometteur pour certaines instances. Le nombre d'itérations n'est pas réduit pour la majorité des problèmes. Donc cette technique de perturbation ne semble pas efficace pour accélérer la résolution, même si les colonnes respectent davantage les durées des tâches en conflit.
Le seul effet concluant est que ce processus permet au problème de quitter la *Phase I* dans certains cas plus rapidement en créant dès le départ des colonnes de coût important. Ensuite la *Phase II* est aussi longue voire plus longue que sans les perturbations.

Il serait plus intéressant lorsque le problème est en *Phase II* de revenir au problème initial ou bien de choisir un autre vecteur de perturbations qui permette non plus de repousser les variables duales à la fin, mais de les déclencher à des endroits précis où l'on souhaite enlever un conflit de ressource.

Il est aussi possible de les décomposer en une combinaison linéaire sur les contraintes. Ceci ne peut être réalisé par la résolution du problème maître avec le simplexe car les variables duales sont des solutions linéaires extrêmes du problème dual. Par conséquent quand la consommation de ressource est excédée durant une tâche, la variables duale se retrouvera sur une seule contrainte et pas répartie linéairement sur plusieurs contraintes, durant cette tâche.

En effet, l'objectif dual revient à minimiser une expression du type $\sum_j b_j \mu_j$ où les membres b_j sont égaux pour un grand nombre d'indices j, donc les solutions sont dégénérées et la solution sera toujours un point extrême des solutions possibles. A partir de cette expression, on peut supposer qu'un choix judicieux des perturbations ϵ_j pourrait orienter la solution pour que la valeur de la variable duale soit affectée à une variable précise mais il est impossible de générer une solution qui ne soit pas un point extrême.

Le dual du problème (P) s'écrit :

$$\max \lambda - \mu^t b$$

$$\lambda \mathbb{I} - \mu^t \mathrm{A} \leq \mathrm{c}$$

$$\mu \geq 0$$

Une perturbation plus élaborée consisterait sans doute à déterminer les cas où pour l'indice j du vecteur μ il existe un palier dans l'objectif et dans les contraintes. C'est à dire que dans le voisinage de j, b_j et $\bar{a}_{ji}$ sont constants, (i est l'indice de la colonne nouvellement générée) où $\bar{a}_{ji} = \sum_{h \in H} y_h^* n_{hj}$.

Il existe j et $\mathrm{V_j}$ proche de j tel que :

$$\forall t \in \{j, \ldots, V_j\}$$

$$b_t = b_j \text{ et } \bar{\mathrm{a}}_{\mathrm{ji}} = \bar{\mathrm{a}}_{\mathrm{ti}}$$

Dans ce cas, la perturbation devraît permettre de repousser les tâches de toute cette zone en prenant une valeur maximale sur le terme b_t tel que t soit maximal.

Aucun test n'a été fait dans ce sens car, nous allons le voir plus loin, il est possible d'accélérer les itérations sans les perturbations, et cette approche a un handicap majeur : celui de créer des instabilités numériques au niveau du problème maître. Dans le chapitre 5, la version V1 est un test réalisé avec une perturbation de valeur $\epsilon = 1.10^{-10}$.

4.7.2 Génération de colonne : combinaisons de points intérieurs

Les colonnes générées par le sous-problème ne respectent pas les contraintes de ressources car l'information fournie par le problème maître n'est pas suffisante pour construire un réseau fidèle aux contraintes de ressources. En effet les variables duales telles qu'elles sont exploitées dans le modèle agissent souvent comme des pénalités sur un ensemble d'activités en conflit, ce qui entraîne la construction d'un nouvel horaire dans lequel les tâches en conflit débuteront à une autre date mais le conflit sera simplement "déplacé" à un autre moment du projet. Il n'y a pas vraiment de décalage entre les activités en conflit, comme l'impose la construction de la colonne optimale. Cette partie vise à trouver des techniques prenant en compte cette notion de décalage dans la construction des colonnes ajoutées au problème maître. La piste suivie étant de mieux exploiter le sous-problème et les variables duales fournies par le problème maître.

Observation du sous-problème

En observant le travail du sous-problème, nous nous apercevons que la colonne générée par le sous-problème qui correspond à une fermeture maximale sur un graphe, est une solution parmi d'autres. Autrement dit, la fermeture n'est pas unique. Donc à partir d'une solution, il est possible de générer plusieurs colonnes de même coût. Si elles apparaissent identiques au vu du sous-problème elles peuvent respecter avec plus ou moins de succès les contraintes du problème maître.

Proposition et essais numériques

Les tests ont d'abord été effectués avant la procédure de lecture de la fermeture :

Tableau 4.2: *Résultats de la méthode de base*

	Temps (sec.)	**Itérations**	**Flots**
Moyenne	19.26	84.68	13.37
Max.	257.6	836	82
Min.	0.1	3	0

Le tableau 4.2 présente les résultats des résolutions en nombres réels de 43 problèmes de 30 activités et 4 ressources. La dernière colonne présente le nombre de colonnes de flot non nul dans la solution en nombres réels. Tests avec la procédure de lecture de la fermeture :

Tableau 4.3: *Résultats de la méthode avec l'analyse de la fermeture*

	Temps (sec.)	**Itérations**	**Flots**
Moyenne	190.43	29.81	16.15
Max.	2821.4	250	85
Min.	0.1	3	0

Le tableau 4.3 présente les résultats pour les mêmes problèmes et de la même manière que dans le tableau précédent.

Il est à noter que les résultats du calcul de la date de fin du projet restent les

mêmes, la méthode de calcul est inchangée ; c'est seulement sa rapidité d'exécution qui change. Le nombre d'itérations est largement diminué avec cette méthode au dépens du temps de calcul qui augmente.

Il est difficile de conclure sur le nombre de colonnes fractionnaires de la solution car il varie de façon irrégulière en fonction du type de problèmes.

Limites et défauts de cette amélioration

Cette approche permet de réduire le nombre d'itérations mais ne permet pas de diminuer le temps de calcul. En effet le nombre de colonnes générées est très important car dans cette expérience nous avons permis la génération de toutes les colonnes possibles à partir de chaque itération du sous-problème. Il serait intéressant de réduire le nombre de colonnes en adoptant une sélection de ces dernières. Cette sélection peut être fondée sur une heuristique simple comme un nombre maximum ou bien mieux, le profil des variables duales.

Extension de la méthode : recherche locale

Il est possible d'intégrer au générateur de colonne une heuristique qui recherche des solutions autour de la colonne optimale construite par le sous problème. Ces solutions deviendront des colonnes candidates pour l'itération suivante du problème maître.

Ces colonnes ne sont pas nécessairement solutions entières mais étant construites sur la base d'un décalage des tâches en conflit de ressources, il est possible qu'elles respectent davantage certaines contraintes en comparaison avec la colonne optimale courante qui le plus souvent déplace le conflit de ressource sans l'éliminer pour autant.

Proposition

En adoptant la même approche que précédemment, il faut ne pouvoir retenir qu'une partie des candidats énumérés par lecture de la coupe du sous problème. Une première sélection serait de ne retenir que les décalages égaux à la durée de la tâche concernée. Si une activité i viole une contrainte à l'instant t, le problème vise à pousser le début de la tâche jusqu'à l'instant $t+1$. Si maintenant les nœuds $(i,t),(i,t-1),\ldots,(i,t-k)$ sont candidats pour un décalage de l'horaire optimal, les candidats retenus seront les nœuds $(i,t-u)$ tels que $u \leq t-p_i+1$. Ainsi nous nous assurons que l'activité i sera décalée totalement par rapport à la colonne optimale et ne viendra pas violer la contrainte à l'instant $t+1$.

Implémentation et essais numériques

Lors de la construction des alternatives sur les colonnes, il faut choisir les candidats correspondant aux dates u telles que $u \leq t+1-p_i$ parmi les nœuds (i,t_i) proposés par la lecture de la fermeture. Les tests semblent montrer qu'il n'y a pas de temps gagné sur la génération de colonne ni d'économie faite sur le nombre d'itérations du problème maître. Il reste trop de colonnes générées pour certains problèmes. Il faut donc trouver un moyen pour en générer encore moins. Voici le tableau récapitulatif des résultats après avoir choisi les colonnes avec des décalages au moins égaux aux durées des activités candidates.

Tableau 4.4: *Résultats de la méthode avec analyse de fermeture restreinte*

	Temps (sec.)	**Itérations**	**Flots**
Moyenne	196.94	32.75	16.1
Max.	3010.2	267	83
Min.	0.1	3	0

Le tableau 4.4 présente les résultats des calculs sur les problèmes vus précédemment. Le nombre d'itérations n'est pas beaucoup plus élevé que dans la première tentative, et le temps de calcul n'est pas diminué ; il est même plus grand dans certains cas.

Amélioration : sélection sur la valeur du décalage

Voyons maintenant si nous ne gardons que la colonne qui admet un décalage de la durée exacte de l'activité. Cette dernière technique semble être un bon compromis entre le temps de calcul et le nombre d'itérations comme le montre le tableau 4.5.

Tableau 4.5: *Résultats de la méthode avec analyse de fermeture très restreinte*

	Temps (sec.)	**Itérations**	**Flots**
Moyenne	41.79	53.2	14.67
Max.	610	492	84
Min.	0.1	3	0

D'après le tableau 4.5, les temps de calculs sont toujours plus importants qu'avec la méthode de base bien que le nombre d'itérations du problème maître ait diminué. Il faut savoir d'où provient cette dépense de temps de calcul. Après observation, il s'avère que c'est le problème maître qui ralentit le processus car il est en moyenne 10 fois plus gros que dans la version de base de la méthode (lorsque nous ne générons qu'une colonne par itération) ceci provient du fait qu'en moyenne, 10 colonnes sont générées à la fois et ajoutées dans le problème maître.

Il est évident que cela pénalise les problèmes difficiles. Donc il faut trouver un moyen de générer moins de colonnes par itérations et/ou des colonnes plus intéressantes pour le problème maître. Pour cela il est possible de prendre de décisions

heuristiques lorsqu'un décalage concerne la même date.

Amélioration : sélection heuristique

Si deux activités différentes doivent débuter à la même date dans la colonne optimale, mais peuvent être décalées dans le temps toutes les deux, alors il est possible de générer deux colonnes supplémentaires qui avancent la date début de chaque activité. Or nous ne pourrions décider de garder qu'une colonne parmi les deux en choisissant, soit le décalage le plus important, soit l'activité qui a les plus faibles variables duales. Cette dernière proposition est tirée du fait que l'activité dont les variables duales sont élevées viole de façon importante les contraintes, donc le problème maître peux avoir tendance à la repousser plus tard. Elle peut en effet violer plusieurs contraintes liées à plusieurs activités, donc il est préférable de la repousser plus tard, alors qu'une activité débutant en même temps mais dont les variables duales sont plus faibles viole sans doute moins les contraintes et peut être avancée sans violer autant de contraintes que l'activité considérée en premier.

Dans le cas où les variables duales seraient identiques, nous avons alors à faire à des activités qui violent les contraintes de la même manière au vu du problème maître. Il suffit donc de décaler celle de plus forte durée ou bien encore de prendre l'ordre lexicographique pour décider.

Implémentation de la méthode heuristique

Lors de la lecture de la fermeture, la construction des candidats au décalage est effectuée activité par activité. Il est donc difficile de trier ces candidats par date de décalage.

On connaît en revanche le coût réduit associé à chaque nœud donc il est possible de comparer ces valeurs pour décider quel nœud sera retenu. (en effet le coût réduit représente exactement les violations de contraintes des activités dans le temps dans le cas de la minimisation du makespan).

Lors de la construction des arcs associés aux coûts réduits, il faut construire un tableau récapitulant les coûts réduits positifs par date et donner un score à chaque nœud égal à dix fois son coût réduit (pour augmenter la précision des comparaisons). Si le nœud courant possède un score inférieur à celui du tableau pour la date courante étudiée, alors ce coût est sauvegardé dans le tableau et dans le nœud, sinon il est effacé du nœud. De cette manière les nœuds de score non nul sont ceux qui possèdent le coût réduit positif le moins grand. Il peut donc être retardé.

Cette technique permet de ne choisir qu'un seul décalage parmi plusieurs candidats à conditions que ces derniers aient des coûts réduits positifs et de valeurs distinctes sur le nœud frontière à la coupe optimale du sous-problème. Elle ne permet pas de distinguer deux activités de même coût réduit positifs.

Tableau 4.6: *Itérations et temps de calcul avec et sans heuristique*

	base	**sans heur.**	**avec heur.**
Itérations	405	405	405
Sous Problème	51.4	69.2	57.3
Problème Maître	46.9	363.5	134.3
Nb. colonnes	405	3046	974
Nb. total d'itr.	688	405	541

Le tableau 4.6 montre le coût en temps de calculs perdu lors des itérations du problème maître. Ceci est dû au trop grand nombre de colonnes générées. Alors

que le sous-problème ne souffre pas trop des calculs liés à la recherche des candidats au décalage. Il faudrait par conséquent traîter plus précisément les colonnes alternatives avant de les introduire dans le problème maître. On ne peut cependant pas réduire encore le nombre de colonnes car la moyenne actuelle est de 2 colonnes par itérations. Il faut en revanche que la colonne alternative soit plus intéressante pour le problème maître.

Tableau 4.7: *Résultats de la méthode avec heuristique*

	Temps (sec.)	**Itérations**	**Flots**
Moyenne	26.74	72.56	13,25
Max.	397.5	707	82
Min.	0.1	3	0

D'après le tableau 4.7, le nombre d'itérations est inférieur à celui de la méthode de base mais le temps de calcul en est toujours pénalisé. Donc cette amélioration reste insuffisante et inacceptable.

Les colonnes alternatives générées ne sont pas assez intéressantes pour apporter un gain de temps au problème maître. L'idéal serait de savoir à quel moment de l'itération cette colonne peut servir de "raccourçi" au problème maître et de la supprimer ensuite pour ne pas engorger le simplexe.

Observations et conclusion

Pour faire des observations sur le comportement des colonnes générées il faudrait pouvoir analyser la constitution du flot optimal afin d'identifier quelle sorte de

colonne est préférable pour la résolution du programme linéaire. Dans un premier temps il est possible d'identifier si les colonnes alternatives font partie des colonnes solutions du problème linéaire ainsi que le nombre de colonnes alternatives qui ont un flot non nul lors des itérations du problème maître.

Le résultat de cette observation est que les petits problèmes ne génèrent pas de colonnes alternatives et ne font donc pas apparaître ces flots.

Le résultat des tests montrent que ces colonnes sont utilisées lors des itérations du problème maître, mais le nombre maximum de colonnes alternatives dans le flot est 1. Le tableau 4.8 présente les occurrences des colonnes alternée dans le flot.

Tableau 4.8: *Proportion des colonnes alternatives dans la solution linéaire*

Nom	Itr.	Flot	Col. alt.	Itr. opt.
41-4	262	67	1	756
45-4	64	37	1	1034
9-4	315	67	1	546
9-4	319	67	1	idem
9-4	321	67	1	idem
9-4	325	65	1	idem

Notons que, d'après le tableau 4.8, la proportion de colonnes alternatives dans le flot dépend du nombre de ces colonnes générées par itérations c'est à dire que dans le cas où nous générons beaucoup de colonnes alternatives, le flot des itérations, y compris le flot optimal, contient ces colonnes.

Ce test peut être un indicateur pendant les itérations pour continuer ou abandonner la méthode de génération de colonnes alternatives.

Le problème d'engorgement du problème maître par un excès des colonnes alternatives dans la matrice peut être réglé en partie en éliminant des colonnes au cours de la résolution de manière à ne garder qu'un nombre limité de colonnes. La difficulté de cette méthode réside dans le choix à faire pour sélectionner les colonnes que nous pouvons enlever sans compromettre la convergence vers l'optimalité. Elle doivent être assez peu intéressantes pour ne pas être reproduites par le sous-problème. Ce perfectionnement reste à faire.

4.7.3 Stabilisation des coûts réduits

Présentation du problème

Lorsque l'on étudie le comportement des variables duales au cours des itérations on peut constater deux points importants :

- les variables duales sont affectées à un temps donné parmi les périodes de temps pendant lesquelles se produit une saturation des ressources,
- la répartition des valeurs des variables duales est discontinue par rapport aux dates du projet.

Le graphe 4.5, représente les coûts réduits d'une tranche de l'horizon du projet "j30-10-1.sm" de la bibliothèque KSD. Ils concernent les nœuds d'une activité de l'intervalle de temps [21, 26] pendant 45 itérations. En abscisse figurent les itérations et en ordonnées les valeurs relatives de coûts réduits. Chaque courbe représente la variation relative du coût réduit associé à une variable x_u^t du problème où t est le temps. Les variations montrent que les coûts réduits prennent une valeur positive à certaines itérations pour repousser l'activité u d'une unité de temps. Les coûts ne sont pas synchronisés entre deux itérations ce qui pourrait permettre l'activité de reprendre la date de début qu'elle ne devait pas prendre juste avant. De plus les coûts réduits ne repoussent les activités que d'une seule unité de temps car on ne

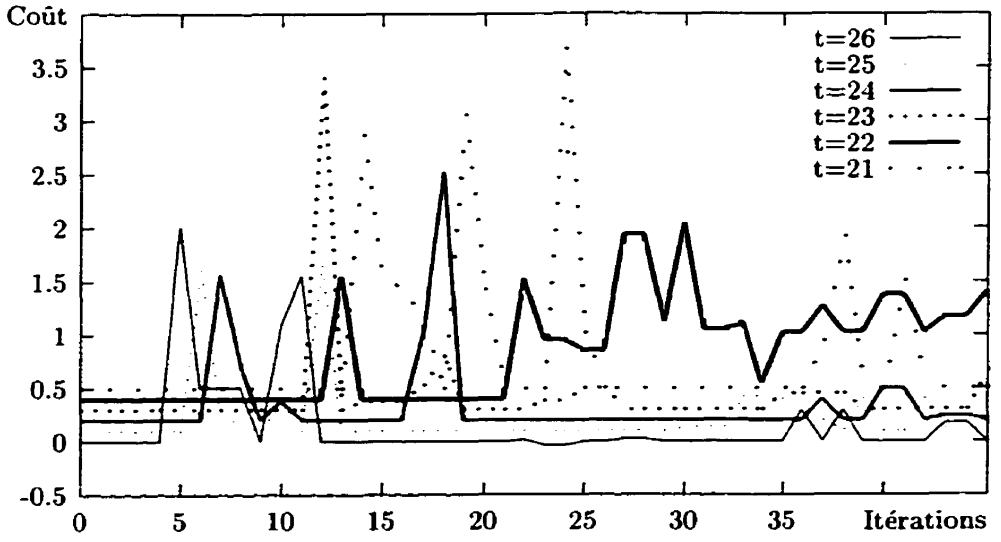


Figure 4.5: *Evolution des coûts réduits avec la méthode de base*

retrouve pas de coûts actifs sur deux dates successives pendant une itération donnée. Cette illustration montre qu'il est possible aux colonnes d'être entraînées dans un phénomène d'oscillations.

Proposition : Mémorisation des variables duales

Ayant constaté que les variables duales oscillent et ne sont pas réparties sur l'ensemble des contraintes de ressources saturées, nous proposons ici de formuler une pondération sur les coûts réduits transmis au sous-problème afin de tenir compte des itérations antérieures et du profil de leur variables duales.

Soit Π^k les variables duales de la k^{ieme} itération. On définit $\lambda^1 = \Pi^1$, et λ^{k+1} à la $k+1^{\text{ieme}}$ itération de la façon suivante :

$$\lambda^{k+1} = (1-\rho)\lambda^k + \rho\Pi^{k+1}$$

où $0 < \rho \leq 0.2$.

Il faut vérifier si les coûts pondérés de cette manière permettent au sous-problème de générer des colonnes de coût marginal négatif. Sinon la résolution va s'arrêter avant l'optimalité. Pour les cas où cela arrive il faut augmenter la valeur du cœfficient ρ.

Au cours des itérations, lorsque le problème ne génère plus de colonnes intéressantes, il faut revenir au problème non pénalisé et vérifier si l'optimalité est atteinte ou pas.

Pour un cœfficient de 0.1, le sous-problème est souvent incapable de générer des colonnes de coût réduit négatif. Il faut donc souvent réitérer la génération de colonne avec les variables duales normales, ce qui représente beaucoup de temps de calculs. Une solution serait de produire des colonnes alternatives au cours des itérations, ce qui augmente la probabilité de générer une colonne de coût réduit négatif à chaque itération. Le temps économisé à ne pas relancer le sous-problème est cependant perdu lors de la résolution du problème maître à cause de l'excédant de colonnes générées. Cette méthode n'est pas infaillible et peut cumuler les retards (répétition de la génération de colonne + excès de colonnes dans le problème maître).

Une meilleure méthode consiste à réitérer le sous-problème non plus en revenant aux valeurs normales des variables duales mais cette fois en réitérant la formule de calcul des variables augmentées.

Cette technique semble réduire par 2 le nombre d'itérations du problème maître sans augmenter le temps de calcul car nous ne générons que quelques colonnes par itérations. La seule chose qu'il reste à définir est le critère d'arrêt, c'est à dire un critère permettant de reconnaître que la solution optimale est atteinte.

L'arrêt des itérations était décidé initialement dès que le sous-problème ne

générait plus de colonnes de coût réduit négatif. Maintenant ceci n'est plus un critère d'arrêt mais un signal pour lancer des itérations successives du générateur de colonnes. Dans un premier temps, nous nous contenterons d'arrêter ces itérations au bout de dix tentatives au cours desquelles le sous-problème n'a pas pu générer de colonnes de coût réduit négatif. Au bout de ces 10 tentatives, on relance le générateur sans la pondération sur les variables duales. Si le sous-problème ne peut toujours pas générer de colonnes, alors l'optimalité est atteinte. Sinon, les itérations vont reprendre.

Dans la pratique ce test n'est exécuté qu'à l'optimalité ou bien très proche (à deux itérations de l'optimalité).
La procédure a pour effet de répartir les variables duales sur l'ensemble des contraintes violées.

Le graphe 4.6, représente les coûts réduits d'une tranche de l'horizon du projet "j30-10-1.sm" de la base de données KSD. Elles concernent les nœuds d'une activité de l'intervalle de temps [21, 26] pendant 45 itérations.

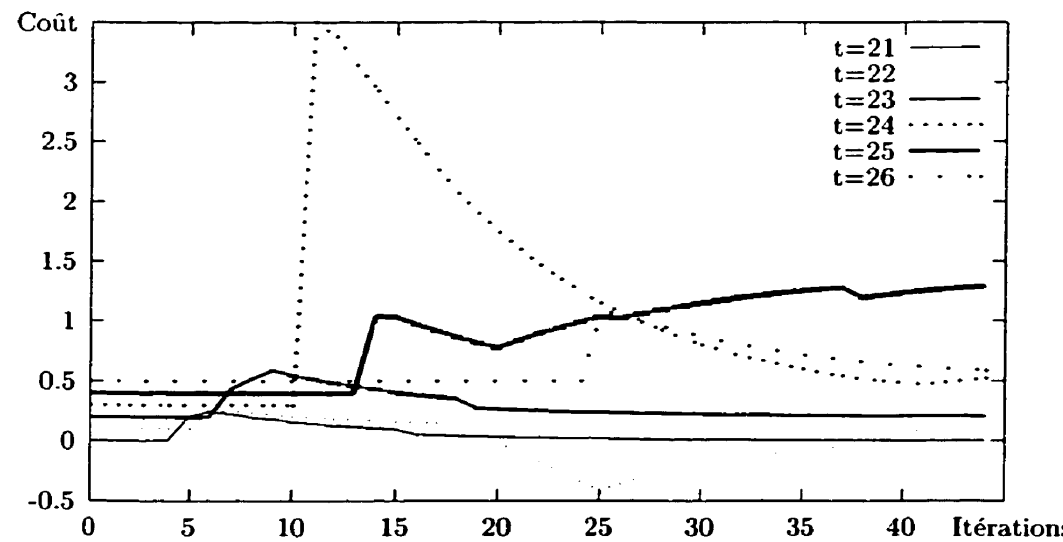


Figure 4.6: *Evolution des coûts réduits avec la méthode de pondération*

En abscisse figurent les itérations du sous-problème de 0 à 45. (Il y a autant d'itérations du sous-problème mais 20 de moins pour le problème maître.) En ordonnées figurent les valeurs relatives des coûts réduits. Chaque courbe représente la variation des coûts réduits en fonction des itérations pour une variable x_u^t du problème où t est le temps. Cette méthode permet de synchroniser les coûts réduits sur plusieurs variables pendant une itération. Par exemple entre les itérations 6 et 17, les courbes de paramètre t égal à 21, 22 et 23, ont le même profil. Cela signifie que les coûts réduits appliqués sur les variables x_u^{21}, x_u^{22} et x_u^{23} seront positifs pendant toutes ces itérations, ayant pour effet de repousser l'activité u pour la faire commencer à la date 24 au plus tôt. Cet exemple montre de quelle manière la pondération stabilise les coûts réduits dans le problème et permet ainsi d'eviter les oscillations.

Graphiquement, l'évolution des coûts réduits semble être plus stable au fur et à mesure des itérations. Dans la pratique, le nombre d'itérations est divisé par quatre. Cette technique apporte une accélération significative de la résolution en terme de temps et de nombre d'itérations. Ce qui laisse croire qu'il est important d'approfondir ce perfectionnement par rapport à ce qu'on a vu jusqu'à maintenant.

La figure 4.7 représente l'évolution de la moyenne du nombre d'itérations du problème maître pour calculer la solution en nombres réels des 480 problèmes KSD de 30 activités et 4 ressources, en fonction du paramètre ρ. Il semble que $\rho = 0.1$ soit la meilleure valeur pour minimiser cette moyenne. Les tests de cette méthode sont détaillés au chapitre 5 par les versions V5, V6 et V13.

Autre version de la stabilisation

La loi de récurrence définie précédemment possède le défaut de converger vers la valeur des variables duales originales de façon asymptotique. Autrement dit, les

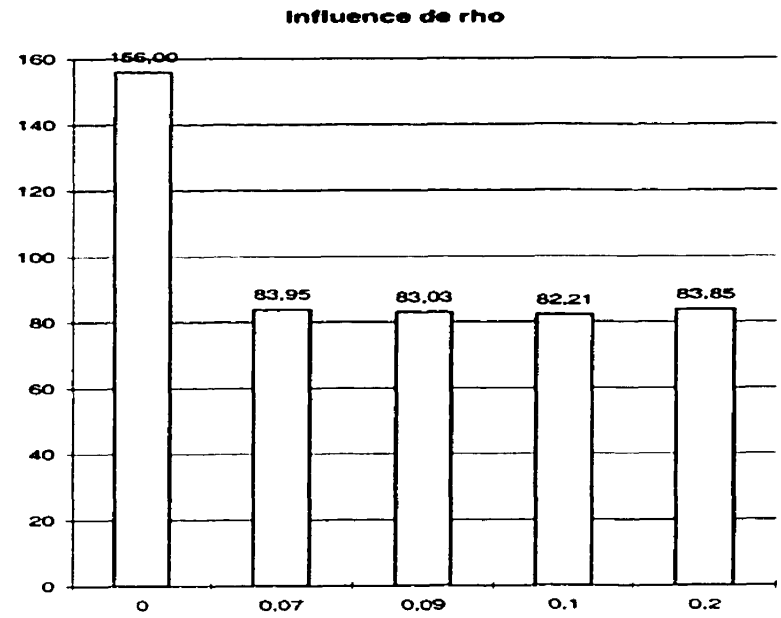


Figure 4.7: *Influence de ρ sur le nombre d'itérations*

pseudo-variables duales convergent vers les vraies variables duales en un nombre infini d'itérations. Ceci peut entraîner des itérations répétitives dans le cas où le sous-problème ne trouverait pas de colonnes de coût réduit négatif. Il arrive alors que le probème s'arrête alors qu'il n'est pas à l'optimalité car les variables duales ne sont traduites correctement entre le problème maître et le sous-problème. La formulation qui suit est la transformation de la première définition des pseudo-variables de manière à retomber sur les vraies variables duales après un nombre fini d'itérations.

$$\lambda^{k+1} = (1-\rho)^q \lambda^{k+1-q} + \sum_{i=1}^{q} (1-\rho)^{i-1} \rho \Pi^{k+2-i}$$

où $5 \leq q \leq 10$ et $0 < \rho < 1$.

Pour simplifier les calculs, nous imposons, $\lambda^{k+1-q} = 0$. On peut vérifier que les pseudo-variables retombent sur les variables duales originales. Car au bout de $q-1$

itérations du sous-problème, les pseudo-variables sont égales à :

$$\lambda^{k+q} = \sum_{i=1}^{q} (1-\rho)^{i-1} \rho \Pi^{k+1}$$
$$= \rho(\sum_{i=1}^{q} (1-\rho)^{i-1}) \Pi^{k+1}$$

Cette dernière valeur est proportionnelle à la vraie valeur de la variable duale, et permet donc au sous-problème de générer une colonne optimale au bout d'un nombre fixe d'itérations $(q-1)$. Nous appelerons q la taille du tampon de stockage des variables. Cette dernière technique d'amélioration est testée par les versions V7 à V10 puis V14 au chapitre 5.

Chapitre 5

Résultats numériques

Ce chapitre présente les jeux d'essais qui existent dans la littérature et ceux qui ont été utilisés pour effectuer les tests numériques. Il récapitule ensuite les expériences effeectuées sur notre méthode.

5.1 La génération des problèmes

De nombreux exemples numériques de projet avec contraintes de ressources existent dans la littérature dont voici un bref apercu. Nous présenterons ensuite les jeux que nous avons utilisé pour tester notre méthode. Nous n'avons pas pu trouver de jeux d'essais comportant des fenêtres de temps. C'est pourquoi nous avons développé des méthodes de calcul des ces fenêtres avant de lancer nos calculs par génération de colonnes.

5.1.1 Les problèmes types de Patterson

En 1984 Patterson [37] a fait une liste de 110 problèmes types de la littérature, dont voici les caractéristiques :

- nombre d'activités : entre 7 et 50
- nombre de types de contraintes par activité : entre 1 et 3
- utilisation des contraintes : tout ou rien pour 103 problèmes (binaires $\neq$ continues)

De nombreux articles ont fondés l'analyse de leurs résultats numériques sur ces problèmes : Patterson et Huber [35], Davis et Heidorn, Davis (1969), Davis et Patterson [11], Davis, Talbot et Patterson (1978).

5.1.2 Les problèmes de "KSD"

Kolisch, Sprecher, et Drexl [28] proposent en 1995 un générateur appelé *Pro-Gen* qui permet la génération de 1216 instances (jeux d'essais) du problème à un seul mode ou plusieurs modes.

Définition des modes de problèmes :

- *single mode* : une activité possède :
  - une durée propre,
  - un besoin propre en ressource d'un certain type.
- *multi-mode* : il existe plusieurs façons de réaliser les activités du projet. Pour une activité il peut y avoir différentes durées, différentes demandes en ressources en fonction du mode choisi.

Types de problèmes générés :

Pour le problème à mode unique, il est possible de générer 480 jeux d'essais différents en changeant les paramètres suivants :

- $C \in IN$ (complexité) : nombre moyen d'arcs sortants et(ou) entrants par nœuds (en général C est voisin de 2)
- RF $\in]0;1[$ (facteur de ressource) : portion moyenne du besoin en ressource d'une activité

- RS $\in]0;1[$ (pouvoir de la ressource) : rapport entre la disponibilité de la ressource et la demande en ressource.

D'autres données sont nécessaires pour construire un graphe, voici les principales :

- nombre d'activités,
- durée des activités,
- nombre de ressources,
- capacité des ressources,
- nombre de types de ressources par activité,
- nombre de successeurs,
- nombre d'activités à la fin du projet,
- nombre de prédécesseurs.

Remarque :

Plusieurs paramètres ont de l'influence sur la difficulté combinatoire d'un jeu d'essai. Par exemple, le temps de calcul diminue lorsque la complexité C du graphe augmente. Voici d'autres facteurs influents :
R : nombre de ressources
S1 : nombre d'activités au début du projet
J : nombre d'activités.

Le tableau 5.1 présente l'influence des paramètres du générateur sur le temps de calcul de résolution des problèmes par une recherche arborescente.
Parmi les 3 variables d'entrées, le pouvoir de ressource (RS) est le facteur le plus influent sur le temps de calcul. Par exemple, 47 problèmes sur 120 dont RS valait

Tableau 5.1: *Types de problèmes et temps de caculs*

C $\nearrow$	$\Rightarrow$	temps $\searrow$
RF $\nearrow$	$\Rightarrow$	temps $\uparrow$
RS $\nearrow$	$\Rightarrow$	temps $\downarrow$
R $\nearrow$	$\Rightarrow$	temps $\uparrow$
J $\nearrow$	$\Rightarrow$	temps $\uparrow$

0.2 n'ont pas pu être résolus en moins d'une heure avec la recherche arborescente de Demeulemeester [17].

Il reste maintenant à tester le comportement de notre méthode sur ces problèmes.

5.2 Tests effectués avec les problèmes de KSD

Les jeux de donnés de KSD m'ont permis de mesurer les performances de la méthode de génération de colonnes et de vérifier l'exactitude de ses calculs avec les solutions optimales trouvées par les méthodes de recherche arborescente de la littérature. Ces tests ont pu être effectués sur les problèmes de 30 activités et 4 ressources, les autres (60, 90 et 120 activités) ne sont encore résolus qu'à l'aide de méthodes heuristiques.

5.2.1 Performances et bornes

Les bornes inférieure et supérieure conditionnent directement la taille du problème. Elles ont par conséquent une influence sur le nombre d'itérations de la méthode et le temps de calcul.

Si la borne supérieure est facile à obtenir et de relativement bonne qualité, la borne inférieure est encore loin d'être acceptable pour pouvoir envisager un branchement rapide, comme le montrent les résultats qui suivent.

Le graphe 5.1 présente les résultats de calcul de bornes par rapport à la solution optimale en nombres entiers pour les 480 problèmes KSD de 30 activités et 4 ressources.

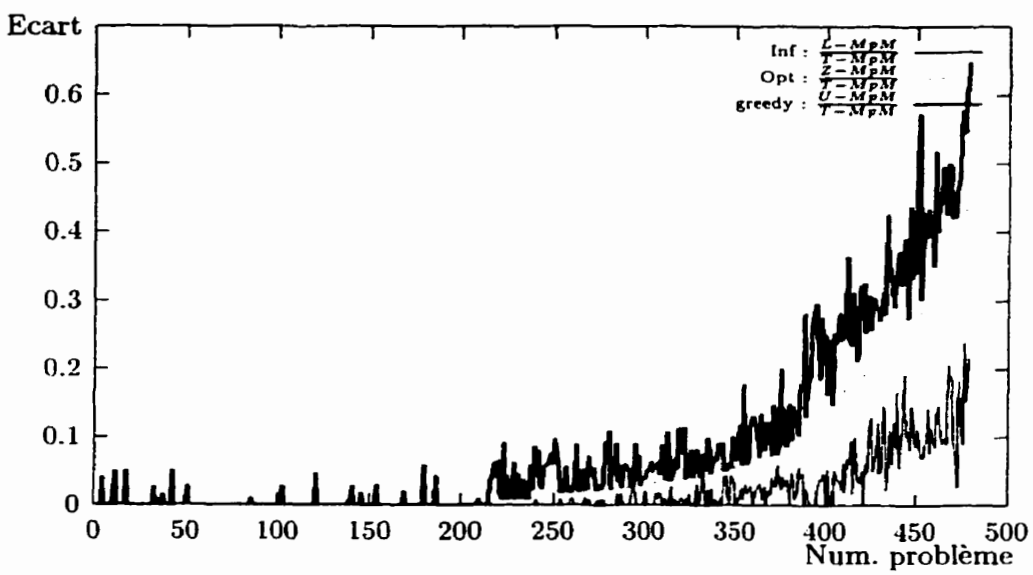


Figure 5.1: *Comportement des bornes*

La figure 5.1 représente trois courbes en ratio des bornes inférieure en trait fin (**Inf**), solution optimale en trait pointillés (**Opt**) et borne supérieure en trait gras (**greedy**), des 480 problèmes ordonnés selon les "écarts croissants" où ce qu'on appelle "Ecart" s'écrit :

$$\text{Ecart} = \frac{\text{Z} - \text{MpMtime}}{\text{T} - \text{MpMTime}}$$

où Z est la solution optimale du problème, *MpMtime* est la solution du problème sans contraintes de ressources, et T est l'horizon du projet. Cet indicateur permet

de classer les problèmes par ordre de difficulté. En effet plus l'écart est grand, plus la solution optimale est éloignée en terme de coût de la solution sans contraintes de ressources.

La borne supérieure est calculée par la méthode sérielle tandis que la borne inférieure est calculée par plusieurs itérations du problème linéaire.

Le calcul de la borne inférieure telle que présentée ici demande un grand nombre d'itérations de la part du programme linéaire ce qui retarde d'autant la résolution du problème avant même de commencer le branchement.

5.2.2 Résultats des tests sur la relaxation linéaire

Les tableaux qui suivent présentent tous les tests effectués sur la relaxation linéaire avec les jeux de données de KSD.

Il est plus pratique de travailler sur la moyenne des 480 problèmes pour comparer les versions du programme. Faisons néanmoins un bilan des problèmes en fonction de leur difficulté. Parmi les 480 problèmes de KSD, pour 197 problèmes, le programme linéaire trouve la solution optimale en nombres entiers en 2 itérations du problème maître et 0.2 secondes ; pour les 283 restants, il faut employer la méthode de branchement pour trouver la solution optimale.

Tableau 5.2: *Description des versions de la méthode*

Num. version	**Option**	**Paramètres**
V0	Version de base	BI=MpmTime BS=T
V1	Borne sup. empirique Perturbations	BS=110%Z_{opt} $\epsilon = 1.10^{-10}$ BI=MpmTime
V2	Stabilisation des variables duales Borne sup. heuristique	$\rho = 0.1$ BI=MpmTime BS=Z_{Greedy}
V3	Stabilisation des variables duales	$\rho = 0.2$ BI=MpmTime BS=Z_{Greedy}
V4	Borne Inf. itérative	BI=LB0 BS=Z_{Greedy}
V5	Stabilisation des variables duales	$\rho = 0.09$ BI=MpmTime BS=Z_{Greedy}
V6	Stabilisation des variables duales	$\rho = 0.07$ BI=MpmTime BS=Z_{Greedy}
V7	Stabilisation des variables duales version avec tampon	$\rho = 0.1$ $q = 5$ BI=MpmTime BS=Z_{Greedy}
V8	Stabilisation des variables duales version avec tampon	$\rho = 0.1$ $q = 10$ BI=MpmTime BS=Z_{Greedy}

Tableau 5.3: *Description des versions de la méthode (bis)*

Num. version	**Option**	**Paramètres**
V9	Stabilisation des variables duales version avec tampon	$\rho = 0.09$ $q = 10$ BI=MpmTime BS=Z_{Greedy}
V10	Stabilisation des variables duales version avec tampon	$\rho = 0.2$, $q = 10$ BI=MpmTime BS=Z_{Greedy}
V11	Application de LB1	BI=LB1 $\rho = 0.1$, $q = 5$ BS=Z_{Greedy}
V12	Application de LB1 avec les iterations du PL	BI=LB1+LB0 mode <u>nice</u> $\rho = 0.1$, $q = 5$ BS=Z_{Greedy}
V13	Stabilisation des variables duales	$\rho = 0.15$ BI=LB1 BS=Z_{Greedy}
V14	Stabilisation des variables duales version avec tampon	$\rho = 0.15$ $q = 10$ BI=MpmTime BS=Z_{Greedy}
V15	Stabilisation des variables duales version avec tampon	$\rho = 0.15$ $q = 10$ BI=LB2 BS=Z_{Greedy}

Tableau 5.4: *Description des versions de la méthode (ter)*

Num. version	Option	Paramètres
V16	Stabilisation des variables duales version avec tampon	$\rho = 0.15$ $q = 10$ BI=LB1+LB2 BS=Z_{Greedy}
V17	Stabilisation des variables duales version avec tampon	$\rho = 0.15$ $q = 10$ BI=LB0+LB1+LB2 BS=Z_{Greedy}

Notations et remarques :

Dans la version V1, trois problèmes sont arrêtés anormalement car il y a un problème d'instabilité numérique à cause des perturbations. ϵ est la valeur de la perturbation du membre de droite du problème maître (cf. 4.6.1). BI signifie Borne Inférieure et BS, Borne Supérieure. PL signifie Programme Linéaire et Mpmtime est la borne inférieure calculée par l'algorithme de Bellman (cf. 4.3). ρ est le cœfficient de stabilisation des colonnes (cf. 4.6.3) et q la taille du tableau tampon (cf. 4.6.3). La mention "mode nice" indique que les calculs n'ont pas utilisé toutes les ressources disponibles de l'ordinateur ce qui peut résulter en des temps de calculs plus longs que prévus. LB0 est le nom de la pocédure itérative pour améliorer la borne inférieure (cf. 4.4) ; LB1 est la borne étendue de Stinson (cf. 4.4.5)

Le tableau 5.5 représente les résultats de calculs du programme linéaire (sans le branchement pour 480 problèmes de KSD avec 30 activités et 4 ressources. Pour chaque version du programme, sont présentés le temps de résolution en secondes, le

nombre d'itérations du problème maître, le nombre de colonnes fractionnaires dans la solution optimale, et le saut d'intégrité. Ces quatre mesures sont calculées en moyenne sur les 480 problèmes. Le saut d'intégrité noté "GAP" est calculé d'après la formule suivante :

$$\mathrm{GAP} = 100 * \frac{\mathrm{Z_{opt}} - \mathrm{Z_{lp}}}{\mathrm{Z_{opt}}}$$

Le symbole σ correspond à l'écart type de la mesure sur l'ensemble des 480 problèmes testés. Pour chaque mesure X_i du jeux i parmi n de moyenne $\bar{X}$:

$$\sigma^2 = \frac{1}{n}\sum_{i=1}^{n}(X_i - \bar{X})^2$$

Tableau 5.5: *Résultats des versions V0 à V17*

Versions		**Temps (sec.)**	**Itr.**	**F**	**GAP**
V0	moy. (σ)	86.5 (245.14)	156 (330)	18 (27.77)	7.63 (10.51)
V1	moy. (σ)	62.8 (240.12)	157.41 (330.3)	18.25 (28.2)	7.69 (9.89)
V2	moy. (σ)	24.14 (65.12)	82.21 (153.06)	16.07 (27.36)	7.62 (10.51)
V3	moy. (σ)	24.39 (68.33)	83.85 (154.0)	16.12 (27.20)	7.62 (10.51)
V4	moy. (σ)	386.6 (1523.28)	551.86 (1504.1)	18.02 (30.84)	6.03 (8.54)
V5	moy. (σ)	27.39 (69.01)	83.03 (155.2)	16.10 (27.23)	7.62 (10.51)
V6	moy. (σ)	24.98 (69.2)	83.95 (1154.9)	16.16 (27.12)	7.62 (10.51)
V7	moy. (σ)	33.83 (87.32)	105.81 (193.10)	16.16 (27.50)	7.62 (10.51)
V8	moy. (σ)	30.88 (86.32)	98.56 (192.10)	16.49 (27.55)	7.62 (10.51)
V9	moy. (σ)	28.46 (83.42)	93.27 (183.2)	15.43 (26.30)	7.28 (10.22)
V10	moy. (σ)	27.50 (77.80)	91.32 (179.37)	16.46 (27.72)	7.62 (10.51)
V11	moy. (σ)	50.87 (214.42)	112.57 (268.29)	17.72 (31.10)	6.95 (9.23)
V12	moy. (σ)	79.88 (344.25)	171.95 (433.50)	15 (26.45)	5.90 (8.49)
V13	moy. (σ)	35.39 (139.23)	87.79 (195.86)	17.16 (30.72)	6.95 (9.23)
V14	moy. (σ)	30.17 (84.4)	93.75 (183.69)	16.45 (27.59)	7.62 (10.51)
V15	moy. (σ)	27.78 (78.3)	90.93 (182.0)	16.49 (27.77)	7.27 (10.11)
V16	moy. (σ)	41.83 (175.5)	97.54 (226.89)	17.39 (30.91)	6.95 (9.22)
V17	moy. (σ)	142.89 (687.24)	250.1 (2285.38)	15.16 (26.09)	5.81 (8.42)

Chaque ligne présente la valeur moyenne (moy.) de la mesure et son écart type entre parenthèses (σ). L'écart type permet d'avoir une idée sur l'écart de chaque mesure avec la moyenne parmi les 480 jeux d'essais effectués. Par exemple, la valeur moyenne du temps de calcul de la version V12 est de 79.88 secondes avec un écart type de 344.25. Cette mesure est très instable, d'après la valeur de l'écart type (en effet le temps de calcul est supérieur à 5000 secondes pour un des jeux).

La version de base V0 procède selon la description des sections 4.1, à 4.4 de ce mémoire. Sa borne inférieure est égale à la durée minimale du projet sans contraintes de ressources, tandis que sa borne supérieure est l'horizon du projet donné par le problème. A partir de cette version, nous avons testé différentes améliorations.

- accélérations du problème maître V1 à V10,
- bornes inférieures V4, V11 à V13, V15 à V17,
- bornes supérieures V1 à V17.

Remarquons tout d'abord que la colonne F qui correspond à la mesure du nombre de colonnes fractionnaires, présente à peu près toujours les mêmes résultats, ce qui signifie que les méthodes employées n'ont pas d'influence sur cette caractéristique de la solution linéaire en nombres réels. Les perturbations pour accélérer le problème maître (V1) n'ont pas donné de bons résultats et ont été abandonnées. En revanche les techniques de stabilisation des variables duales ont été testées avec les versions V2, V3, V5 pour la méthode sans tampon (cf. 4.7) puis V7 à V10 avec tampon. Ces tests montrent que la version la plus rapide est celle sans tampon avec un cœfficient $\rho = 0.1$, mais la version avec tampon est plus stable comme nous le verrons lors du branchement. Les versions V11 à V17 utilisent la stabilisation avec tampon qui est la plus efficace à savoir : $\rho \in [0.1, 0.2]$ et $q \in [5, 10]$. Cette configuration nous a permis d'expérimenter rapidement les idées d'améliorations des bornes. La borne inférieure LB0 (cf. 4.5) a été testée dans la version V4. Elle est très couteuse en temps de calcul mais apporte des gains sur le saut d'intégrité. La borne inférieure LB1 a été utilisée dans les versions V11 et V13 avec deux stabilisations différentes.

Elle apporte de meilleurs résultats que la version de base sans trop augmenter le temps de calcul. Toutefois la borne obtenue est supérieure à LB0. et que la version V4. Nous obtenons une borne inférieure meilleure que la version V4 lorsque LB1 est combinée à LB0 dans la version V12. La borne inférieure LB2 a été testée avec la version V15 puis combinée avec les bornes LB1 et LB0 dans les versions V16 et V17. La borne LB2 est supérieure à LB1 et la combinaison de LB1 et LB2 n'est pas mieux que LB1 seule. En revanche elle apporte un gain en temps de calcul car la taille du graphe du sous-problème est réduite. En outre, la combinaison des trois bornes inférieures LB0, LB1 et LB2 apporte des gains tant en temps de calcul qu'en qualité de la borne en comparaison avec les autres versions. La borne supérieure est utilisée dans les versions V2 à V17. Dans la version V1, on utilise le résultat connu de la solution optimale à laquelle on ajoute 10%. Cette approximation apporte une certaine accélération mais elle n'est pas rigoureuse du point de vue expérimental. Nous l'avons par conséquent remplacée par le calcul d'une solution approchée avec la méthode "greedy" sans perte d'efficacité. Cette méthode a également permis de réduire le nombre d'itérations de la résolution en évitant le passage en *phase I* décrite à la section 4.3 car la solution "greedy" est une solution réalisable du problème maître.

La meilleur configuration du programme linéaire, c'est-à-dire celle qui offre le meilleur compromis entre nombre d'itérations et qualité de la borne inférieure, est la suivante :

- système de stabilisation de colonnes avec tampon $\rho = 0.1$, $q = 10$,
- application de la borne inférieure LB1 et LB2 sur les fenêtres de temps.

Cette configuration assure le calcul de la solution en nombres réels le plus rapidement possible et avec un saut d'intégrité minime. Il doit donc être effectué lors du premier nœud d'exploration de notre algorithme.

Toujours d'après ces résultats numériques, la méthode de perturbations (V1) est à abandonner, quant à la méthode d'analyse de la fermeture, (cf. 4.6.2), elle demanderait des perfectionnements pour ne pas ralentir le problème maître. Notons que le tableau 5.5 ne présente pas de résulats sur cette méthode car ils sont récapitulés au chapitre 4.

Ces tests montrent que, le saut d'intégrité est de l'ordre de 6% en moyenne sur les problèmes de KSD, même en utilisant le système d'itérations successives coûteux en temps de calcul (V4 et V12). Ces tests montrent aussi que dans le meilleur des cas, il faut en moyenne 24.14 secondes pour résoudre un problème en nombres réels. Ce qui est assez long en comparaison avec les méthodes de recherche arborescente avec retour en arrière (cf. chapitre 3). Ces dernières résolvent les problèmes en nombres entiers en 28.97 secondes, en moyenne, ce qui prouve leur supériorité pour ces instances.

Il est néanmoins possible de comparer les performances de cette méthode avec la leur si nous considérons que le saut d'intégrité qu'ils prennent au départ (au premier nœud) de leur recherche arborescente est inférieure ou égal à celui de la version V16 (LB1+LB2) soit 6.95% alors que nous pouvons garantir un meilleur saut avec notre méthode qui est de 5.81% (version V17). C'est une amélioration qui nous avantage par rapport à leur approche même si le temps de calcul pour y arriver est plus élevé.
En effet ce calcul peut être exploité pour résoudre les problèmes de grandes tailles que ne peuvent résoudre les méthodes de recherche arborescente (à partir de 60 activités).

Nous avons testé l'influence de la taille du problème sur le temps de calcul du problème linéaire et voici quelques résultats.

Les tableaux 5.6, 5.7 et 5.8 présentent les résultats pour un jeu généré avec les

paramètres C, RF et RS définis en 5.1.2. La taille des problèmes est 30, 60, 90 et 120. Or le temps de calculs croît lentement en fonction de cette taille.

Le problème linéaire est à la base de notre méthode c'est pourquoi nous lui avons consacré beaucoup de temps pour le perfectionner. La partie qui suit présente les tests effectués sur la méthode de séparation et d'évaluation progressive qui utilise le programme linéaire.

Tableau 5.6: *Influence de la taille du projet*

C=1.5		RF=0.75		RS=0.5	
Taille	**Sous Prob.**	**Prob. Maître**	**Itr.**	**Col.**	**f**
30	2.4	0.5	26	84	11
60	5.1	0.9	28	92	12
90	7.3	0.8	27	90	14
120	19.7	1.6	32	109	16

Tableau 5.7: *Influence de la taille du projet(bis)*

C=2.1		RF=0.75		RS=0.5	
Taille	**Sous Prob.**	**Prob. Maître**	**Itr.**	**Col.**	**f**
30	4.1	3.8	72	257	28
60	11.2	2.8	44	156	23
90	9.6	1.6	30	98	20
120	32	6.6	54	197	24

5.2.3 Résultats des tests sur le branchement

Le temps de calcul de cette opération dépend directement du nombre de nœuds de branchements et le nombre de nœuds de branchement dépend de nombreux

Tableau 5.8: *Influence de la taille du projet et des ressources*

C=1.8		RF=0.25		RS=0.5	
Taille	**Sous Prob.**	**Prob. Maître**	**Itr.**	**Col.**	**f**
30	0.8	0.1	14	28	4
60	3.0	0.1	17	37	12
90	3.3	0.1	16	37	8
120	8.7	0.7	39	36	9

paramètres qui sont liés au programme linéaire et à la solution fractionnaire qu'il fournit. Il dépend également des techniques employées pour fabriquer les fenêtres de temps. C'est pourquoi nous avons testé plusieurs règles de décisions et de sélection des fenêtres pour construire les nœuds de branchement sur un problème choisi dans la librairie KSD afin de connaître quelle est la plus efficace en terme de nombre de nœuds de branchements.

Les tests suivants (tableau 5.9) ont été effectués sur le problème "j30-11-10.sm". Ce nom signifie que le problème possède 30 activités et 4 ressources et que le facteur de ressource est de 0.75, le pouvoir de ressource est de 0.70 et la complexité est de 1.5 (cf. 5.1 La génération des problèmes).

Remarques et Notations :

L'option "barycentre" consiste à calculer la date pivot comme le barycentre des dates solutions du programme linéaire par opposition avec "médiane" qui cacule la date au centre de la fenètre de temps de l'activité considérée (cf. 4.7.1). La notation "maxTW" (resp. "minTW") dans le tableau 5.9 correspond au critère de

Tableau 5.9: *Résultats des versions de branchement*

Version	Option	Param. ρ	N	Itr.	Temps (sec.)
B0	barycentre & maxTW	$\emptyset$	346	2345	80.4
B1	barycentre & maxTW	0.1	34	276	14.3
B2	barycentre & maxTW	0.15	37	245	13.2
B3	barycentre & maxTW	0.2	16	142	8.1
B4	barycentre & maxTW	0.3	185	1057	36.0
B5	médiane & maxTW	0.1	903	5838	324.2
B6	bary. & minTW	0.2	334	1794	101.3
B7	bary. & maxEcart	0.2	792	4031	243.8
B8	bary. & minEcart	0.2	>1000	> 8043	>300
B9	bary. & maxE. & maxTW	0.2	61	477	21.2
B10	bary. & minE. & maxTW	0.2	25	192	9.4
B11	bary. & maxMarge	0.2	338	2056	101.7
B12	bary. & minMarge	0.2	32	223	12.3
B13	bary. & $\emptyset$	0.2	>5600	>650	>200

sélection des nœuds pivots. C'est-à-dire que l'on choisit parmi les nœuds pivots candidats celui de l'activité dont la fenêtre de temps est la plus large (resp. petite). "maxEcart" (resp. "minEcart") signifie que le critère de sélection est l'activité dont les dates de début proposées sont les plus éloignées (resp. proches).

Le critère "maxMarge" (resp. "minMarge") revient à choisir l'activité dont la marge est la plus grande (resp. petite) (cf. 4.7.3).

L'ensemble des tests se décompose par le choix de la méthode de calcul de la date pivot d'une part et par le choix du critère de sélection des activités candidates au branchement d'autre part. La version B5 utilise le calcul de la date pivot avec la technique de la médiane. Mais cette option n'est pas meilleur que toutes les versions

de B0 à B4 qui utilise, à la place, la technique du barycentre. Donc nous avons abandonné la technique de la médiane pour ce concentrer sur des tests avec le barycentre.

Remarquons que les options sur le programme linéaire ont une influence sur la performance du branchement. Dans les versions B0 à B4, nous avons gardé les mêmes options de branchement (même critère de sélection, même calcul de pivot). Il s'avère que la stabilisation des colonnes permet de réduire efficacement le nombre de nœuds de branchement. Dans cette optique, les versions B1 à B4 testent l'influence du paramêtre ρ sur le nombre de nœuds de branchement. La meilleur valeur de ce paramêtre ($\rho = 0.2$) a été utilisé par la suite. Quant au critère de sélection, d'après le test B13, il est nécessaire de prendre un critère de sélection pour ne pas faire diverger le branchement. Ce critère a été défini comme l'activité dont la fenêtre de temps est la plus large pour les versions B0 à B5 puis B9 et B10. La version B6 utilise le critère inverse (fenêtre la plus petite) mais il n'améliore pas la résolution. Le critère additionnel sur les écarts de dates à été testé seul dans la version B7 puis en combinaison avec les fenêtres les plus larges (B9) mais n'a pas donné de bons résultats. Le critère minEcart est inefficace lorsqu'il est utilisé seul (B8). Il produit de bons résultats lorsqu'il est combiné avec maxTW (B10) mais reste inférieure à maxTW seul (B3). Les tests avec les critères maxMarge et minMarge (versions B11 et B13) ne sont pas meilleurs que maxTW.

La meilleure configuration pour la résolution en nombres entiers est la version B3 :

- programme linéaire : stabilisation de colonnes avec buffer ($\rho = 0.2$ $q = 10$), en utilisant les bornes supérieure et inférieure heuristiques,
- choix du pivot : barycentre des dates fractionnaires,
- choix de l'activité sur laquelle brancher : plus large fenêtre de temps.

Le branchement est nécessaire pour 283 problèmes sur 480. Parmi eux, 174

problèmes possèdent un nombre de colonnes fractionnaires compris entre 2 et 13 et les 109 restants possèdent entre 13 et 125 colonnes fractionnaires. Pour cette dernière catégorie, le branchement nécessite beaucoup de nœuds avant de trouver l'optimum. On peut considérer qu'ils ne peuvent pas être résolus par notre méthode. Pour les résoudre, il faudrait pouvoir réduire le saut d'intégrité et le nombre de colonnes fractionnaires.

5.2.4 Résultats des tests sur d'autres fonctions objectifs

Objectif : minimum des temps d'attente ("Flow min")

Si nous choisissons de minimiser la somme des dates de débuts de chaque activité, la résolution devient très longue. L'objectif revient à minimiser la somme des temps d'attentes des activités entre leur date au plus tôt et leur date de début. L'objectif s'écrit :

$$\min \sum_{i=1}^{N} t_i - r_i$$

où t_i est la date de début de l'activité i.

Cet objectif se traduit dans le graphe du sous-problème par la construction d'arcs partant de chaque nœud (i, t) $(\forall i,\ \forall t \in [r_i, d_i - p_i])$ vers le puits. Cette construction augmente la taille du graphe ce qui ralentit l'algorithme de flot maximum.

Tableau 5.10: *Résultats de la méthode avec minimisation des retards*

Itérations	**Temps (secondes)**	
	S. Prob.	**Pb. Maître**
267	166.9	8.1

Le tableau 5.10 présente les temps de calcul d'un problème résolu par le programme linéaire avec l'objectif de juste à temps. Il a fallu 267 itérations et un temps

de 166.9 secondes au sous-problème soit 0.62 secondes par itération. Avec l'objectif de minimisation de la date de fin du projet, il faut seulement 0.036 secondes par itérations pour trouver la solution du même problème. Cet objectif n'est pas très facile à traîter par notre méthode. Il n'est pas possible de tester la performance car il n'existe pas de solutions connues pour les problèmes de KSD avec cette fonction objectif.

Objectif : Juste à temps

En définissant des dates de fins souhaitées pour chaque activité il est possible de calculer une solution pour laquelle l'ordonnancement vise à respecter au mieux ces dates de fins. L'objectif s'écrit :

$$\min \sum_{i=1}^{N} |t_i - \bar{d}_i|$$

où t_i est la date de début de l'activité i et $\bar{d}_i$ la date de fin souhaitée.

Le calcul des dates de fins souhaitées pour chaque activité i peut être effectué à partir des fenêtres de temps du diagramme potentiel-tâche :

$$\bar{d}_i = r_i + p_i + \alpha p_i$$

où α est tel que : $1 < \alpha \leq 3$.

En ce qui concerne l'implémentation de cet objectif, le graphe du sous-problème possède des arcs de capacité $\bar{d}_i - t$ de la source vers tout nœud (i, t) tel que $t < \bar{d}_i$, et des arcs de capacité $t - \bar{d}_i$ d'un nœud (i, t) où $t > \bar{d}_i$ vers le puits.

Objectif : minimum de retard

En utilisant les mêmes données de date de fins souhaitées par activités, il est possible de ne considérer que le retard maximum des activités avec leur date de fins.

$$\min \max_i \{|t_i - \bar{d}_i|\}$$

Cet objectif présente l'avantage de pouvoir resserrer les fenêtres des deux côtés lorsqu'une borne supérieure est connue.

La construction du sous-problème se fait de la même façon que pour l'objectif de juste à temps, mais il ralentit moins les calculs car la taille du graphe décroît après la première solution réelle.

Chapitre 6

Conclusion

Dans ce mémoire, vous a été présenté le travail sur les problèmes de gestion de projet avec contraintes de ressources et fenêtres de temps utilisant une méthode exacte de décomposition.

La résolution est fondée sur un programme linéaire qui est la décomposition de Dantzig-Wolfe d'un programme linéaire en nombre entiers. Ce programme linéaire est résolu par génération de colonnes à l'aide d'un sous-problème de flot maximum et un problème maître résolu par la méthode du simplexe. Une combinaison linéaire des colonnes générées par le sous-problème assure le respect des capacités de ressources par les activités du projet. Cette solution en nombres réels constitue une borne inférieure de la solution optimale du programme linéaire en nombres entiers. Elle constitue le point de départ d'une méthode d'évaluation et de séparation pour rechercher une solution en nombres entiers. Ce branchement, fondé sur des restrictions sur les fenêtres de temps, applique des modifications de données initiales afin de restreindre le problème linéaire jusqu'à ce qu'il génère une colonne solution en nombres entiers optimale.

Nous avons réalisé le programme informatique de cette méthode en utilisant un algorithme de flot maximum qui est adapté au calcul de fermeture maximale sur un graphe. Nous avons créé un programme pour intégrer cet algorithme au problème maître qui est résolu par Cplex. Enfin, nous avons raccordé ce programme linéaire à un algorithme de recherche arborescente pour effectuer le calcul de branchement sur les fenêtres de temps.

Les tests effectués sur le programme linéaire à partir de la banque de jeux d'essais fournie par R. Kolisch et A. Sprecher de l'Université de Kiel en Allemagne ont permis d'évaluer la performance du logiciel. La résolution de problèmes difficiles a montré certaines faiblesses du programme linéaire quant à la qualité de la borne inférieure qu'il fournit. Or la borne inférieure joue un rôle majeur dans la taille du branchement, donc la méthode de base ne suffit pas pour résoudre des problèmes difficiles (contraintes de ressources serrées).

Cette observation de la méthode de base nous a conduit à proposer et valider les perfectionnements du programme linéaire suivants :

1. calcul heuristique d'une borne supérieure,
2. calcul itératif d'une borne inférieure,
3. calcul puis resserrage heuristique des fenêtres de temps,
4. génération de plusieurs colonnes par le sous-problème,
5. gestion des perturbations sur les contraintes du problème maître,
6. stabilisation des coûts réduits.

Ce travail d'amélioration de la résolution du programme linéaire a permis de réduire les temps de calcul et la taille des problèmes par quatre. Et parmi ces améliorations celle sur la stabilisation des coûts réduits pourrait être testée pour d'autres méthodes de génération de colonne. Elle surpasse les méthodes de perturbations et l'analyse de la fermeture pour générer plusieurs colonnes (ou points intérieurs), tout en répondant aux objectifs de ces techniques. Son principe consiste simplement à mémoriser les violations rencontrées pour les éviter lors des itérations suivantes. En complément aux améliorations apportées au programme linéaire, nous avons aussi perfectionné le branchement. Il s'agit premièrement d'une méthode de coupe en utilisant la borne supérieure et les fenêtres de temps, et deuxièmement de

plusieurs règles de décisions de branchement sur les dates de début de la solution fractionnaire. Les tests du problème en nombres entiers ne sont pas actuellement significatifs car l'écart entre la solution réelle et la solution optimale en nombres entiers est trop importante. Il faudrait en effet réduire la moyenne de cet écart au dessous de 4% pour envisager des résultats intéressants. (la moyenne est actuellement de 6% environ). Il est donc nécessaire d'améliorer la résolution du programme linéaire avant de travailler sur le branchement.

La meilleure méthode pour résoudre les problèmes de gestion de projet avec contraintes de ressources est une recherche arborescente développée par E. Demeulemeester [17]. Si les performances de la génération de colonnes sont loin de concurrencer cette méthode pour les problèmes de petite taille (30 activités et 4 ressources), elle offre en revanche un outil de base plus souple permettant de modéliser de nombreux problèmes d'ordonnancements de projet. La résolution des gros problèmes reste une question à éclaircir étant donné que les méthodes de branchements ne présentent pas de résultats alors que la génération de colonnes peut en résoudre quelques-uns lorsque le programme linéaire est suffisant pour trouver la solution optimale en nombres entiers.

Les extensions qu'il est possible d'explorer sur la base de nos travaux sont nombreuses. Nous en distinguons quatre principales qui sont les suivantes. La méthode peut être testée sur les problèmes de gestion de projet avec contraintes de ressources et temps d'attentes minimums et maximums entre les tâches. Elle peut aussi être testée avec le calcul de la borne inférieure de Mingozzi [33] à la place de la borne de Stinson. Il est possible de modéliser des fonctions objectifs différentes de celles présentées dans ce mémoire. Enfin la méthode est peut-être intéressante avec un autre type de sous-problème que le flot maximum. Si nous admettons que l'étude du flot dans le graphe du sous-problème ne permet pas de générer de décalage de tâches en conflit de par sa nature fractionnaire. Or ce décalage des tâches constitue le principe primordial de la résolution des problèmes de ressources. Le programme linéaire est incapable de générer des colonnes solutions en nombre entiers dès que

les contraintes de ressources sont actives. Le travail du branchement est donc considérable puisqu'il est supposé restreindre le sous-problème de manière à ce que les dates de début ou de fin des fenêtres de temps des activités en conflits traduisent exactement la solution optimale. ($r_u = t_u^*$ ou $d_u - p_u = t_u^*$ pour chaque activité u en conflit de ressource où t_u^* est la date de début de la solution optimale en nombres entiers).

La méthode d'analyse de la fermeture du graphe du sous-problème est une tentative pour palier à ce handicap, mais elle n'a pas abouti à des résultats intéressants. Elle nécessite une analyse plus approfondie dans sa construction des colonnes alternatives, c'est pourquoi il est légitime de se demander si ce type de méthode ne viendrait pas remplacer ou compléter l'algorithme de flot maximum.

Bibliographie

[1] ALVAREZ-VALDES TAMARIT (1989). "Heuristic algorithms for resource-constrained project scheduling : A review and an empirical analysis", dans : R. Slowinsky et J. Weglarz *Advances in Project Scheduling*, Helsevier 113-134.

[2] BALAS, E. (1970). "Project Scheduling with Resource Constraints", *Applications of Mathematical Programming Techniques*, Carnegie-Mellon University, Pittsburgh.187-200.

[3] BEDWORTH, D.D. et BAILEY, J.E. (1982). "Integrated Production Control Systems - Management, Analysis, Design", *Wiley*, New York.

[4] BELL, C.E. HAN, S. (1991). "A new heuristic solution method in resource-constrained project scheduling", *Naval Research Logistics* 38, 315-331.

[5] BOCTOR, F.F. (1990). "Some Efficient Multi-Heuristic Procedures for Resource-Constrained Project Scheduling", *European Journal of Operational Research*, 49, 3-13.

[6] BOWMAN, E.H. (1959). "The Schedule-Sequencing Problem", *Operations Research* 621-624.

[7] BRAND, J.D. MEYER, W.L. and SHAFFER, L.R. (1964). "The Resource Scheduling Problem in Construction", Civil Engineering Studies, Report No. 5, Department of Civil Engineering, University of Illinois, Urbana, III.

[8] BURTON, M. (1967). "Some Mathematical Models for the Allocation of Limited resources to Critical Path Type Scheduling Problems" non publié, University of Illinois, Urbana.

[9] CARLIER, J. CHRÉTIENNE, R. (1988). "Problèmes d'ordonnancement" *ed. MASSON.*

[10] CARRUTHERS, J.A. BATTERSBY, A. (1966). "Advances in critical path methods", *Operational Research Quaterly* 17, 359-380.

[11] DAVIS, E.W. (1973). "Project scheduling under resource constraints : Historical review and categorization of procedures", *AIIE Transactions* 5/4, 297-313.

[12] DANTZIG, G. B. WOLFE, P. (1961). "The Decomposition Algorithm for Linear Programming", *Econometrica*, 9, N0. 4.

[13] DAVIS, E.W. PATTERSON, J.H. (1975). "A Comparison of Heuristic and Optimum Solutions in Resource-Constrained Project Scheduling", *Management Science*, 21, 944-955.

[14] DECKRO, R.F. WINKOFSKY, E.P. HERBERT, J.E. et GAGNON, R. (1991). "A decomposition approach to multi-project scheduling", *European Journal of Operational Research* 51, 110-18.

[15] DEMEULEMEESTER, E.L. HERROELEN, W.S. (1992). "A Branch-and-Bound Procedure for the Multiple Resource-Constrained Project Scheduling Problem", *Management Science*, Vol. 38, No.12.

[16] DEMEULEMEESTER, E.L. (1995). "Minimizing Resource Availibility Costs in Time-Limited Project Networks", *Management Science* Vol. 41, No. 10.

[17] DEMEULEMEESTER, E.L. HERROELEN, W.S. (1997). "New Benchmark Results for the Resource-Constrained Project Scheduling Problem", *Management Science* Vol.43, No. 11, pp 1485-1492.

[18] DESROCHERS, M. DESROSIERS, J. SOLOMON, M.M. (1992). "A New Optimization Algorithm for the Vehicle Routing Problem with Time Windows, *Operations Research* 40, 342-354.

[19] DOERCH, R.H. PATTERSON, J.H. (1977). "Scheduling a project to maximise its present value : a zero-one programming approach." *Management Science* 23, 882-889.

[20] ELMAGHRABY, S.E. (1968). "The One Machine Sequencing Problem Problem With Delay Costs" *Journal of Industrial Engineering.*

[21] FISHER, Marshall Lee (1970). "Optimal Solution of Resource Constrained Network Scheduling Problems" Technical Repot N0. 56, *Oprations Research Center*, MIT.

[22] GELINAS, Sylvie (1996). "Problèmes d'ordonnancement", working paper, (non publié) Ecole Polytechnique de Montréal.

[23] GOMORY, R.E. WADE, C.S. (1961). "Write-Up for Integer Programming Z, 7090, PK IPO2 and PK IPM2" *IBM Corp.*, Yorktown Heights, New York.

[24] GORENSTEIN, S. (1969). "An Algorithm for Project (Job) Sequencing with Resource Constraints", *Operations Research*, 834-850.

[25] HADLEY, G. (1964). "Project Planning and Manpower Schedulin" sec.8.8 of *Nonlinear and Dynamic Programming*, Addison-Welsey

[26] HU T.C. 1961, "Parallel Sequencing and Assembly Line Problems" *Operation Research.*

[27] JOHNSON, T.J.R. (1967). "An Algorithm for the Resource-Constrained, Project Scheduling Problem" non publié, Massachutsetts Institute of Technology.

[28] KOLISH, R. SPRECHER, A. and DREXL, A. (1995). " Characterization and Generation of a General class of Resource Constrained Project Scheduling Problems : Easy and Hard Instances", *Management Science* 41, 1693–1703.

[29] KOLISH, R. (1996). "Serial and parallel resource-constrained poject scheduling methods revisited : Theory and computation" *European Journal of Operational Research* 90, 320-333.

[30] LENSTRA, J.K. RINNOOY, Kan A.H.G. (1978). "Complexity fo scheduling under precedence constraints", *Operatios Research* 26, 22-35.

[31] LEON, V.J. BALAKRISHNAN, R. (1995). "Strength and adaptability of problem-space based neighbourhoods for resource constrained scheduling", *OR Spektrum* 17, 173-182.

[32] MANDEVILLE, D.W. (1965). "The Development of Network Analysis Resource Balancing Methods from Assembly Line Balancing Techniques", thèse MS non publiée, Purdue University, Lafayette, Indiana.

[33] MINGOZZI, A. MANIEZZO, V. RICCIARDELLI, S. and BIANCO, L. (1994). " An Exact Algorithm for Project Scheduling with Ressource Constraints Based on a New Mathematical Formulation", *Technical Report* No. 32, University of Bologna.

[34] OGUZ et BALA (1994). "A comparative study of computational procedures for the resource constrained project scheduling problem", *European Journal of Operational Research* 72, 406-416.

[35] PATTERSON, J.H. HUBER, W.D. (1974). "A Horizon-varying, 0-1 approach to project scheduling", *Management Science*, 20/6, 990-998.

[36] PATTERSON, J.H. ROTH, G.W. (1976). "Scheduling a project under multiple resource constraints : A zero-one programming approach", *AIIE Transactions* 8, 449-455.

[37] PATTERSON, J.H. (1984). " A comparison of exact approaches for solving the multiple constrained resource, project scheduling problem", *Management Science* Vol. 10, N0. 7.

[38] PATTERSON, J.H. et al. (1990). "A backtracking algorithm for solving scheduling problems", *European Journal of Operational Research*, 49, 68-79.

[39] PICARD, J.C. (1976). "Maximal closure of a graph and application to combinatorial problem", *Management Science* 22 No. 11.

[40] PRITSKER, A.B. et WATTERS, L. (1968). "A Zero-One Programming Approach to Scheduling with Limited Ressources", *The Rand Corporation*, RM-5561-Pr.

[41] PRITSKER, A.B. WATTERS, L.J. WOLFE, P.M. (1969). "Multiproject sheduling with limited resources : A zero-one programming approach", *Management Science*, V.16 N.1.

[42] ROUNDY, R.O. MAXWELL, W.L. HERER, Y.T. TAYUR, S.R. et GETZLER, A.W. (1991). "A Price-Directed Approach to Real-Time Scheduling of Production Operations", *IIE Transactions*, 23 (2), 149-160.

[43] ROY, B. (1966). "Prise en compte des contraintes disjonctives dans les méthodes du chemin critique", *RAIRO*, 38, 69-84.

[44] SAMPSON WEISS, E. (1993). "Local search techniques for the generalized resource constrained project scheduling problem", *Naval Research Logistics* 40, 365-375.

[45] SEPIL, C. ORTAC, N. (1997). "Performance of th heuristic procedures for constrained projects with progress payments", *Journal of the Operational Research Society*, 48, 1123-1130.

[46] SLOWINSKY, R. (1980). "Two approaches to problems of resource allocation ammong project activities-A comparative study", *J.Opl Res. Soc.* Vol. 31, p711-723.

[47] STINSON, J.P. DAVIS, E.W. KHUMAWALA, B.M. (1978). "Multiple resource-constrained scheduling using branch and bound", *AIIE Transactions* 10, 252-259.

[48] TALBOT, F.B. (1976). "An Integer programming algorithm for the resource constrained project scheduling problem", The Pennsylvania State University.

[49] TACHEFINE, B. (1996). " Planification optimale de la production dans une mine à ciel ouvert", Thèse de doctorat, Ecole Polytechnique de Montréal.

[50] TALBOT, F.B. (1978). "An efficient integer programming algorithm with network cuts fo solving resource-constrined scheduling problems",*Management Science*, Vol. 24, No. 11.

[51] WAGNER, H.M. GIGLIO, R.J. et GLASER, R.G. (1964). "Preventive Management Scheduling by Mathematical Programmig" *Management Science.*

[52] WEIST, J.D. (1963). "The Scheduling of Large Projects with Limited Resources", PhD thèse non publiée, Carnegie Institute of Technology.

[53] WEGLARZ, J. (1981). "Project Scheduling with Continuously-Divisible, Doubly Constrained Resources", *Management Science*, Vol. 27, No. 9, 1040-1053.