HAL
open science

Programmation linéaire en nombres entiers pour l'ordonnancement cyclique sous contraintes de ressources

Maria Ayala

To cite this version:

Maria Ayala. Programmation linéaire en nombres entiers pour l'ordonnancement cyclique sous contraintes de ressources. Automatique / Robotique. Université Paul Sabatier - Toulouse III, 2011. Français. NNT : . tel-00604537

HAL Id: tel-00604537

https://theses.hal.science/tel-00604537v1

Submitted on 29 Jun 2011

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

THÈSE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Délivré par *l'Université Toulouse III - Paul Sabatier*
Discipline ou spécialité : *Systèmes informatiques et Systèmes industriels*

Présentée et soutenue par *Maria Alejandra Ayala P.*
Le *15 Juin 2011*

Titre : *Programmation linéaire en nombres entiers pour l'ordonnancement cyclique sous contraintes de ressources*

JURY
Eric Sanlaville, Professeur à l'Université du Havre
Marc Sevaux, Professeur à l'Université de Bretagne-Sud
Christian Artigues, Chargé de recherche au CNRS
Claire Hanen, Professeur à l'Université de Paris-Ouest-Nanterre-La Défense
Benoit Dupont-de-Dinechin, Directeur de Développement Logiciel Kalray
Pascal Sainrat, Professeur à l'Université Paul Sabatier

Ecole doctorale : *Ecole Doctorale Systèmes*
Unité de recherche : *LAAS-CNRS*
Directeur(s) de Thèse : *Chrsitian Artigues*
Rapporteurs :
Eric Sanlaville, Professeur à l'Université du Havre
Marc Sevaux Professeur à l'Université de Bretagne-Sud

A Luis, mi mayor tesoro,
gracias por tantas alegrias juntos,
esta nueva meta es tuya. Te amo hijo.

A mis padres, especialmente mi Mamá,
quien ha esperado pacientemente este dia.
Gracias por las oraciones y el amor incondicional.
Los quiero mucho.

A Bernat, por cada dia compartido,
por cada palabra de aliento,
gracias por estar junto a mi y
cuidarme con tanto amor.
T'estimo molt amor meu.

A mis hermanos y sobrinos quienes han
esperado pacientemente y me han animado
en cada nuevo proyecto de mi vida.
Los quiero mucho.

Remerciements

Tout d'abord je voudrais remercier mon directeur de thèse M. Christian Artigues, qui a accepté de m'encadrer pendant ces années de recherche. Merci pour ses idées, pour son enthousiasme et pour sa confiance.

Je remercie M. Marc Sevaux et M. Eric Sanlaville d'être rapporteurs de ce travail. Merci beaucoup pour leurs observations et leurs remarques pertinentes, ce qui m'ont permis d'améliorer mon rapport.

Je remercie tout particulièrement Mme Claire Hanen et son ancienne doctorante Abir Benabid pour le travail que nous avons réalisé ensemble et l'échange d'idées qui a contribué à la réussite de cette thèse.

Je souhaite également adresser mes remerciements à M. Benoit Dupont-de-Dinechin pour sa disponibilité et pour l'intérêt qu'il a porté à mon sujet de thèse. Merci aussi M. Pascal Sainrat d'avoir fait partie de jury.

Mes remerciements les plus chaleureux à Bernat Gacias pour son aide, ses conseils et pour le regard critique et objectif qu'il a porté à chaque étape de ce travail. Merci de m' avoir supporté chaque jour, même quand j'étais invivable. T'estimo molt amor meu.

Un grand remerciement à mon adorable fils Luis, sa présence, son sourire et son amour m'ont encouragé chaque jour. Cette réussite est pour toi, tu es la personne qui me procure le plus de bonheur dans la vie.

Je n'oublie pas le soutien de ma famille, spécialement ma mère. Dans la distance ils sont restés toujours proches et aujourd'hui ils sont heureux pour moi.

Merci à mes amies : Mila qui m'a encouragé pour commencer ce projet et Malinda avec qui j'étais toujours complice.

Je voudrais remercier l'Université de los Andes, qui m'a donnée la possibilité de venir en France pour finir ma formation.

Merci à tous les membres du groupe MOGISA pour leur accueil et leur convivialité.

Résumé

Un problème d'ordonnancement cyclique consiste à ordonner dans le temps l'exécution répétitive d'un ensemble d'opérations liées par des contraintes de précédence, en utilisant un nombre limité de ressources. Ces problèmes ont des ap ! plications immédiates dans les systèmes de production ou en in ! formatiq ue parallèle. Particulièrement, ils permettent de modéliser l'ensemble des contraintes de précédence et de ressource à prendre en compte pour l'ordonnancement d'instructions dans les processeurs de type VLIW (Very Long Instruction Word). Dans ce cas, une opération représente une instance d'une instruction dans un programme.

L'ordonnancement d'instructions de boucles internes est connu sous le nom de pipeline logiciel. Le pipeline logiciel désigne une méthode efficace pour l'optimisation de boucles qui permet la réalisation en parallèle des opérations des différentes itérations de la boucle. Dans cette thèse, nous nous intéressons principalement au problème d'ordonnancement périodique qui est un cas particulier de l'ordonnancement cyclique et qui est également la base du pipeline logiciel. Le terme ordonnancement modulo désigne un ordonnancement périodique tel que l'allocation de ressources pour une opération donnée n'est pas modifiée d'une itération sur l'autre ! .

Pour résoudre le problème, nous nous intéressons aux formulations de programmation linéaire en nombres entiers, et notamment à la résolution du problème par des techniques de séparation, évaluation, génération de colonnes, relaxation lagrangienne et des méthodes hybrides. En particulier, nous proposons des nouvelles formulations basées sur des variables binaires représentant l'exécution d'ensembles d'instructions en parallèle. Enfin, les méthodes développées ont été validées sur des jeux d'instances industrielles pour des processeurs de type VLIW.

Mot(s)-clé(s) : décomposition de Dantzig-Wolfe - méthodes hybrides - ordonnancement modulo - programmation linéaire en nombres entiers - relaxation lagrangienne.

Abstract

Integer linear programming for the resource-constrained modulo scheduling problem

The resource-constrained modulo scheduling problem (RCMSP) is a general periodic cyclic scheduling problem, abstracted from the problem solved by compilers when optimizing inner loops at instruction level for very long instruction word parallel processors. Since solving the instruction scheduling problem at compilation phase in less time critical than for real time scheduling, integer linear programming (ILP) is a relevant technique for the RCMSP.

In this work, we are interested in the methods based on the integer linear programming for the RCMSP. At first, we present a study of the two classic integer linear formulation for the RCMSP. A theoretical evidence of the equivalence between the classic formulations is shown in terms of linear programming (LP) relaxation. Secondly, based on the ILP formulations for the RCMSP, stronger formulations for the RCMSP derived from Dantzig-Wolfe decomposition are presented. In these formulations, the number of variables can be huge, for this reason, we proposed a column generation scheme to solve their LP relaxations. We propose also the heuristics methods based on the Lagragian relaxation and decomposed software pipelining. The heuristic methods search the transformation of the classic integer linear programming for the RCMSP for the performance improvement in the time for the search of solutions. All formulations are compared experimentally on problem instances generated from real data issued from the STMicroelectronics ST200 VLIW processor family.

Key-words : integer linear programming, Dantzig-Wolfe decomposition, Lagrangian relaxation, modulo scheduling, heuristic methods.

Sommaire

Introduction **13**

1 Ordonnancement modulo sous contraintes de ressources **15**
1.1 Introduction 15
1.2 Problèmes d'ordonnancement cyclique classiques 16
1.3 Ordonnancement modulo sous contraintes de ressources 26
1.4 Conclusion 35

2 PLNE pour le RCMSP **37**
2.1 Introduction 37
2.2 Programmation linéaire en nombres entiers 38
2.3 Problème d'ordonnancement de projet sous contraintes de ressources 42
2.4 Programmation linéaire en nombres entiers pour l'ordonnancement modulo sous contraintes de ressources 44
2.5 Résultats expérimentaux 54
2.6 Conclusion 62

3 Décomposition de Dantzig-Wolfe pour le RCMSP **63**
3.1 Introduction 63
3.2 Décomposition de Dantzig-Wolfe et génération de colonnes 64
3.3 Formulations renforcées pour le RCMSP 67
3.4 Génération de colonnes pour le RCMSP 70
3.5 Résultats expérimentaux 72
3.6 Conclusion 77

4 Relaxation Lagrangienne et solutions réalisables **79**
4.1 Introduction 79
4.2 Relaxation Lagrangienne 80
4.3 Relaxation Lagrangienne de la formulation directe desagrégée 82
4.4 Heuristique et résultats expérimentaux 86
4.5 Conclusion 92

5 Heuristique hybride pour le RCMSP **93**
5.1 Introduction 93
5.2 Pipeline logiciel décomposé (DSP) 94
5.3 Formulation hybride pour le RCMSP 99
5.4 Résultats expérimentaux 100
5.5 Conclusion 106

Conclusion et perspectives **107**

Introduction

Un problème d'ordonnancement consiste à organiser dans le temps la réalisation de tâches, compte tenu de contraintes temporelles (délais, contraintes d'enchaînement) et de contraintes portant sur la disponibilité des ressources requises. Un problème d'ordonnancement cyclique consiste à ordonner dans le temps l'exécution répétitive d'un ensemble d'opérations liées par des contraintes de précédence, en utilisant un nombre limité de ressources. Ces problèmes ont des applications immédiates dans les systèmes de production ou en informatique parallèle. Particulièrement, ils permettent de modéliser l'ensemble des contraintes de précédence et de ressource à prendre en compte pour l'ordonnancement d'instructions dans le processeurs de type VLIW (Very Long Instruction Word). Dans ce cas, une opération représente une instance d'une instruction dans un programme. L'ordonnancement d'instructions de boucles internes est connu sous le nom de *pipeline* logiciel. Le *pipeline* logiciel désigne une méthode efficace pour l'optimisation de boucles qui permet la réalisation en parallèle des opérations des différentes itérations de la boucle. Dans cette thèse, nous nous intéressons principalement au problème d'ordonnancement périodique qui est un cas particulier de l'ordonnancement cyclique et qui est également la base du pipeline logiciel. Dans l'ordonnancement périodique, l'écart entre les dates de début de deux occurrences successives d'une même tâche est égal au temps de cycle ou période λ. Dans un intervalle de longueur λ, chaque opération est exécutée exactement une fois. Le principal indicateur d'un ordonnancement cyclique est ce temps de cycle ou période λ, car la minimisation de la période λ équivaut à améliorer le rendement ou débit de l'ordonnancement. Un des objectifs de cette thèse est d'étudier le problème de la minimisation de la période λ suivit de la minimisation d'un objectif secondaire qui généralement s'exprime en fonction des dates de début des opérations. Le terme *ordonnancement modulo* désigne un ordonnancement périodique tel que l'allocation de ressources pour une opération donnée n'est pas modifiée d'une itération sur l'autre.

Cette thèse est dédiée au problème d'ordonnancement modulo sous contraintes de ressources cumulatives, avec une application à l'ordonnancement d'instructions VLIW. Pour résoudre le problème, nous nous intéressons aux formulations de programmation linéaire en nombres entiers, et notamment à la résolution du problème par des techniques de séparation, évaluation, génération de colonnes, relaxation lagrangienne et des méthodes hybrides. En particulier, nous proposons des nouvelles formulations basées sur des variables binaires représentant l'exécution d'ensembles d'instructions en parallèle. Enfin, les méthodes développées ont été validées sur des jeux d'instances industrielles pour des processeurs de type VLIW. Ce manuscrit est structurée de la manière suivante :

Le chapitre 1 a pour objectif d'introduire le problème de l'ordonnancement cyclique de base et le problème de l'ordonnancement périodique. Un état de l'art sur les principaux problèmes

d'ordonnancement cyclique est présenté. Ensuite, nous focalisons sur le problème de l'ordonnancement modulo sous contraintes de ressources (RCMSP). Les principales méthodes pour la résolution de ce type de problèmes sont également détaillées. Nous concluons le chapitre avec la présentation de l'architecture des processeurs de la famille ST200 car les expérimentations ont été réalisées sur des problèmes d'ordonnancement d'instructions de ce type de processeur.

Le chapitre 2 aborde la recherche de bornes inférieures pour la période (λ) et pour le makespan ($Cmax$). Ces bornes sont obtenues à partir des formulations de programmation linéaire en nombres entiers (PLNE) pour le RCMSP. Nous présentons d'abord les notions de base de la PLNE et les formulations classiques décrites dans la littérature pour le problème d'ordonnancement de projets sous contraintes de ressources. Ces formulations sont en effet la base des formulations PLNE pour le RCMSP. Nous présentons en détail les deux formulations PLNE plus remarquables de la littérature. Ensuite, une étude théorique sur la qualité de leurs relaxations est présentée. Finalement, nous clôturons le chapitre avec la présentation des résultats expérimentaux pour la comparaison des bornes inférieures et des solutions optimales obtenues avec les formulations PLNE pour le RCMSP.

Le chapitre 3 est consacré à la présentation de nouvelles formulations renforcées pour le RCMSP basées sur la décomposition de Dantzig-Wolfe. Nous proposons d'appliquer la décomposition de Dantzig-Wolfe aux formulations PLNE du RCMSP présentées dans le chapitre précédent. Ces nouvelles formulations sont basées sur le concept d'ensembles de tâches réalisables. Les tâches d'un ensemble réalisable peuvent en effet être exécutées simultanément sans violer les contraintes de précédence ou de ressources. Un schéma de génération de colonnes est présenté pour la résolution des relaxations des nouvelles formulations afin d'obtenir de nouvelles bornes pour la période λ et pour l'objectif secondaire. Des expérimentations pour l'évaluation de la qualité de ces nouvelles bornes inférieures proposées sont presentées en fin de chapitre.

Le chapitre 4 décrit une autre méthode pour la recherche non seulement de bornes inférieures pour la période λ et pour l'objectif secondaire, mais également pour la recherche de solutions réalisables (bornes supérieures). Dans ce chapitre, nous proposons une relaxation lagrangienne à partir de la dualisation de la contrainte de ressources de la formulation PLNE directe désagrégée qui est présentée dans le deuxième chapitre. Nous proposons la résolution du problème dual à partir de la transformation du problème lagrangien en un problème de coupe minimale dans un graphe orienté. Ensuite, nous présentons une méthode pour la recherche de bornes supérieures qui intègre les résultats de la relaxation lagrangienne dans la formulation PLNE décomposée structurée (introduite dans le chapitre 2). Nous finissons le chapitre avec la présentation des expérimentations pour l'évaluation des bornes supérieures obtenues à partir de la relaxation lagrangienne.

Le chapitre 5 présente une approche hybride pour le problème du RCMSP. Le but est de proposer une nouvelle approche pour la recherche de solutions réalisables tout en minimisant la période λ. Cette approche est fondée sur le principe du pipeline logiciel décomposé. Il s'agit de résoudre le problème en deux étapes : le décalage d'instructions et la compactation de la boucle. Nous proposons une formulation qui articule le calcul du retiming pour la phase de décalage des instructions et la formulation PLNE décomposée pour la phase de compaction de la boucle. Des résultats expérimentaux sont presentés afin d'évaluer la qualité des solutions réalisables obtenues. Ce chapitre est le résultat du travail réalisé en collaboration avec Abir Benabid (LIP6, Université Pierre et Marie Curie) et Claire Hanen (LIP6, Université Paris Ouest Nanterre la Défense).

Chapitre 1

Ordonnancement modulo sous contraintes de ressources

Ce chapitre présente le problème de l'ordonnancement cyclique de base. Ce problème couvre une variété de problèmes dans différents domaines. Dans le cadre de cette thèse, nous nous intéressons au problème d'ordonnancement périodique qui est un cas particulier du problème d'ordonnancement cyclique. Nous nous sommes intéressé plus particulièrement au problème d'ordonnancement modulo sous contraintes de ressources. Ce problème apparaît en compilation pour les processeurs VLIW et la technique de "pipeline logiciel" qui permet l'exploitation du parallélisme inhérent aux structures de boucles. D'abord, nous présentons la formulation du problème d'ordonnancement modulo sous contraintes de ressources et nous décrivons ensuite l'architecture des processeurs de la famille ST200 puisque les expérimentations de cette thèse ont été réalisées sur des problèmes d'ordonnancement d'instructions de ce processeur.

1.1 Introduction

Un problème d'ordonnancement consiste à organiser dans le temps la réalisation de tâches, compte tenu de contraintes temporelles (délais, contraintes d'enchaînement) et de contraintes portant sur la disponibilité des ressources requises. Un problème d'ordonnancement cyclique consiste à ordonner dans le temps l'exécution répétitive d'un ensemble d'opérations liées par des contraintes de précédence, en utilisant un nombre limité de ressources. Les principaux domaines dans lesquels les problèmes d'ordonnancement cyclique peuvent apparaître sont, les problèmes de systèmes de production (Boussemart *et al.*, 2002; Cohen *et al.*, 1985; Hanen and Munier, 1994a; Christofides *et al.*, 1987; Dinechin, 2004; Eichenberger and Davidson, 1997; Hanen and Munier, 1994b; Kubale and Nadolski, 2005; Ramchandani, 2006; Rajagopalan *et al.*, 2001; Ramamoorthy and Ho, 1980) ou les problèmes d'informatique parallèle (Chaudhuri *et al.*, 1994; Dinechin, 2004; de Dinechin *et al.*, 2010; Eichenberger and Davidson, 1997; Hanen and Munier, 1994b; Ramchandani, 2006; Rajagopalan *et al.*, 2001; Ramamoorthy and Ho, 1980; Chrétienne, 2000; Munier, 1996a; Hanen and Munier, 2009; Munier, 1996b). A titre d'exemple, l'exécution des structures de contrôle de type itératif (boucles) dans une architecture à processeurs parallèles peut être modélisée par un ordonnancement cyclique. La plupart des études dans le domaine de la compilation pour les processeurs superscalaires modernes et les architectures VLIW [1] se

1. Very long Instruction Word ou Mot d'instruction très long.

focalisent sur le problème d'ordonnancement répétitif d'instructions. Ce problème est défini par un ensemble d'opérations à ordonnancer, un ensemble de dépendances entre ces opérations et l'architecture du processeur cible qui définit des contraintes de ressources complexes (Dinechin, 2004). Une opération est considérée comme une instance d'une instruction dans un programme. L'ordonnancement d'instructions de boucles internes est connu sous le nom de *pipeline* logiciel. Le *pipeline* logiciel désigne également une méthode efficace pour l'optimisation de boucles qui permet la réalisation en parallèle des opérations des différentes itérations de la boucle.
Nous nous intéressons principalement au problème d'ordonnancement périodique qui est un cas particulier de l'ordonnancement cyclique et la base du pipeline logiciel. Nous nous sommes intéressés plus particulièrement au problème d'ordonnancement modulo sous contraintes de ressources cumulatives, avec une application à l'ordonnancement d'instructions VLIW. Dans ce chapitre, nous exposons le problème général d'ordonnancement cyclique, puis nous définissons le problème d'ordonnancement modulo sous contraintes de ressources et son application au problème d'ordonnancement d'instructions VLIW. Le chapitre est structuré de la manière suivante. La section (1.2) est dédiée à la présentation du problème d'ordonnancement cyclique de base et de l'ordonnancement périodique, ainsi qu'à leurs résolutions et à la présentation des différents types de problèmes d'ordonnancement cyclique. Dans cette même section, nous présentons les problèmes à machines parallèles dans le contexte cyclique et les principales approches pour leur résolution. La section (1.3) est consacrée à la description du problème d'ordonnancement modulo. Les notions de parallélisme de niveau et de pipeline logiciel sont également présentées. Ensuite, la définition formelle du problème d'ordonnancement modulo sous contraintes de ressources est donnée. Nous clôturons cette section avec la description de l'architecture ST200 et un exemple d'application de l'ordonnancement modulo sous contraintes de ressources.

1.2 Problèmes d'ordonnancement cyclique classiques

1.2.1 Le problème central répétitif

Dans un problème d'ordonnancement cyclique, les tâches sont dites génériques et elles se répètent une infinité de fois. Autrement dit, une tâche a différentes occurrences. Une occurrence de tâche représente une réalisation de celle-ci. On note par O l'ensemble des n tâches génériques et par $< i, q >$ la $q^{ième}$ ocurrence de la tâche générique i tel que $i \in O$, $q \in \mathbb{N}$. Chaque tâche générique i a une durée p_i. Un ordonnancement σ affecte une date de début σ_i^q à chaque occurrence $< i, q >$. L'ensemble $\sigma = \{\sigma_i^q\}_{1 \leq i \leq n}^{q \geq 0}$ définit donc un ordonnancement.

Définition 1 *Le temps de cycle moyen de σ est défini par*

$$z(\sigma) = \lim_{q \to \infty} \frac{max_{i \in O}(\sigma_i^q + p_i)}{q} \tag{1.1}$$

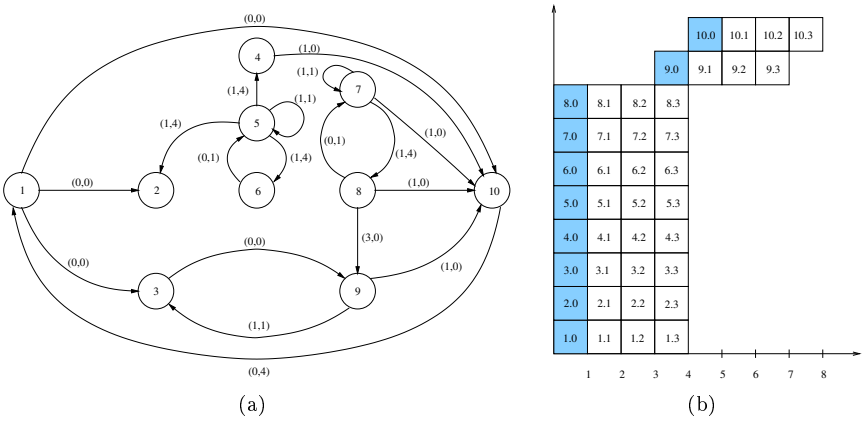
L'inverse $\xi = \frac{1}{z(\sigma)}$ est généralement référencé comme le rendement ou débit de l'ordonnancement (Hanen and Munier, 1994b).

Le problème d'ordonnancement cyclique de base consiste à trouver un ordonnancement réalisable minimisant le temps de cycle moyen $z(\sigma)$ tout en respectant des contraintes de précédence entre les tâches, explicitées un peu plus loin.

Définition 2 *Un ordonnancement est dit périodique avec une période λ si pour chaque tâche générique i nous avons (Hanen and Munier, 1994b)*

$$\forall i \in O, \forall q \geq 1, \quad \sigma_i^q = \sigma_i^0 + \lambda q \tag{1.2}$$

La période λ est égale au temps de cycle moyen $z(\sigma)$. Le vecteur $\sigma = \{\sigma_1, ..., \sigma_n\}$ avec $\sigma_i = \sigma_i^0$ pour $i = 1; \ldots, n$, est défini comme l'ordonnancement générique. Ce vecteur est répété chaque λ unité du temps. La figure 1.1(b) donne un exemple d'ordonnancement de 10 tâches où chaque tâche est exécutée chaque unité de temps ($\lambda = 1$). Nous avons représenté trois occurrences de chaque tâche. La première occurrence de chaque tâche est affichée en gris et les autres en blanc. Dans l'ordonnancement périodique, l'écart entre les dates de début de deux occurrences



(a) (b)

FIGURE 1.1 – Graphe de précédences et ordonnancement périodique de période $\lambda = 1$

successives d'une même tâche est égal au temps de cycle (ou période) λ. Dans un intervalle de longueur λ, chaque tâche est exécutée exactement une fois. Le principal indicateur d'un ordonnancement cyclique est le temps de cycle ou période λ, car la minimisation de la période λ équivaut à améliorer le rendement ou débit de l'ordonnancement.

Une contrainte de précédence représente la relation de succession entre deux tâches. Dans l'ordonnancement cyclique, les contraintes de précédence (appelées aussi contraintes de dépendance par référence aux applications informatiques) portent sur les occurrences de tâches. Une dépendance entre deux tâches (i, j), associée à une distance (ou hauteur) ω_i^j, génère pour toute occurrence q de i une contrainte de précédence entre les occurrences $< i, q >$ et $< j, q + \omega_i^j >$, ce qui définit une contrainte uniforme. L'ensemble des contraintes de précédence peut être modélisée par un graphe orienté $G = (O, E)$. Les arcs de G sont pondérés par deux fonctions $\theta : O \to N$ et $\omega : O \to N$ appelées **latence (ou longueur)** et **distance (ou hauteur)** telles que si e_i^j désigne un arc entre le nœud i et j, alors e_i^j correspond à la contrainte uniforme suivante (Hanen and Munier, 1994b) :

$$\forall q \geq 1, \ \sigma_i^q + \theta_i^j \leq \sigma_j^{q+\omega_i^j}. \tag{1.3}$$

La figure 1.1 (a) donne un exemple de contraintes de précédence avec la paire (latence, distance) indiquée sur chaque arc et l'ordonnancement périodique présenté dans la figure 1.1 (b) respecte

ces contraintes pour $\lambda = 1$. Le triplet (G, θ, ω) est référencé comme un graphe uniforme. Un cas particulier remarquable est lorsque la latence θ_i^j est égale à la durée de l'opération p_i, ce qui implique que la tâche $< j, q + \omega_i^j >$ peut débuter au plus tôt après la fin d'exécution de la tâche $< i, q >$.
Dans le contexte de l'ordonnancement d'instructions, la distance indique que le résultat de la tâche i d'une itération q quelconque est utilisé par la tâche j de l'itération $q + \omega_i^j$. La tâche i est liée à l'opération j par une contrainte de précédence intra-itération si $\omega_i^j = 0$ ou par une contrainte de précédence inter-itération si $\omega_i^j > 0$. La distance ω représente le nombre d'itérations qui séparent les instances de i et j. Par exemple, la figure 1.2 illustre les notions de latence et de distance entre deux tâches (i, j) tel qu'il existe une contrainte de précédence définie par $\theta_i^j = 2$ et $\omega_i^j = 1$, avec $\lambda = 2$.

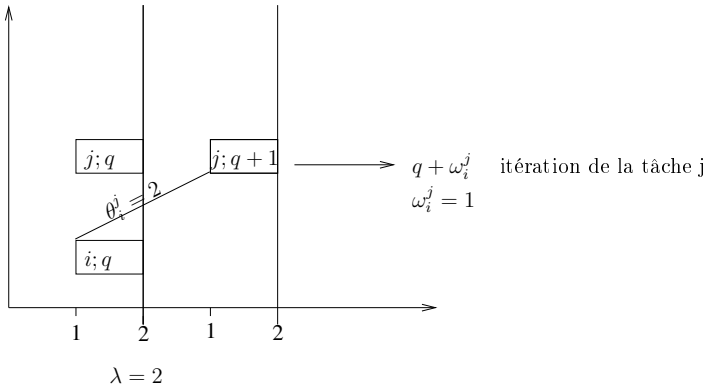


FIGURE 1.2 – Exemple de latence et distance.

Le problème d'ordonnancement cyclique de base peut être exprimé comme suit :

$$min\ z(\sigma) \tag{1.4}$$

s.c

$$\sigma_i^q + \theta_i^j \leq \sigma_j^{q+\omega_i^j} \quad \forall (i,j) \in G,\ q \in Z. \tag{1.5}$$

$$\sigma_i^q + p_i \leq \sigma_i^{q+1} \quad \forall i \in O,\ q \in Z. \tag{1.6}$$

$$\sigma_i^q \geq 0 \quad \forall i \in O \tag{1.7}$$

L'objectif est la minimisation du temps de cycle moyen $z(\sigma)$ donné par l'expression (1.4). Les contraintes (1.5) représentent l'ensemble des contraintes de précédence entre la $q^{ième}$ occurrence de la tâche générique i et la $(q + \omega_i^j)$ occurrence de la tâche générique j. Les contraintes (1.6) imposent que la $(q+1)^{ième}$ occurrence de la tâche générique i ne peut commencer avant la fin d'exécution de la $q^{ième}$ occurrence de la tâche i. Il a été prouvé dans Hanen and Munier (1994b); Ramchandani (2006) que si le problème de base de l'ordonnancement cyclique admet une solution, il existe alors un ordonnancement périodique qui minimise le temps de cycle moyen. Nous pouvons donc insérer (1.2) dans (1.5) et (1.6) et, en considérant $\sigma_i^0 = \sigma_i$, nous obtenons le problème d'ordonnancement cyclique de base suivant :

$$min\ \lambda \tag{1.8}$$

s.c

$$\sigma_i + \theta_i^j - \lambda\omega_i^j \leq \sigma_j \quad \forall (i,j) \in G. \tag{1.9}$$

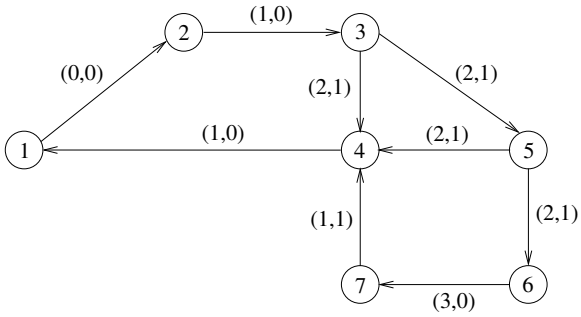
Nous notons immédiatement que le problème de décision associé (existence d'un ordonnancement de temps de cycle λ donné) est polynomial (résolution d'inégalités de potentiels). Le temps de cycle moyen minimal peut être donc obtenu en cherchant le plus petit λ vérifiant ces inégalités.

Un algorithme de résolution efficace peut être tiré de l'analyse des circuits critiques (Hanen and Munier, 1994b). Si μ représente un circuit dans le graphe G, la latence du circuit est $\theta(\mu) = \sum_{(i,j)\in\mu} \theta_i^j$ et la distance du circuit est $\omega(\mu) = \sum_{(i,j)\in\mu} \omega_i^j$. Nous définissons $h(\mu)$, *la valeur du circuit* μ, comme le rapport $\frac{\theta(\mu)}{\omega(\mu)}$. Le circuit ayant la valeur maximale de $h(\mu)$ est le circuit critique de G. Les résultats suivants ont été établis dans Hanen and Munier (1994b); Kampmeyer (2006).

- Le problème d'ordonnancement cyclique de base admet une solution si et seulement si tout circuit μ de G est de distance strictement positive.
- Pour tout circuit μ, $h(\mu)$ est une borne inférieure du temps de cycle moyen.
- Si le problème d'ordonnancement cyclique de base admet une solution, le temps de cycle moyen minimal est alors égal à la valeur du circuit critique.

Plusieurs algorithmes polynomiaux sont disponibles pour calculer le circuit critique. Un algorithme en $O(n^3 \log n)$ est proposé dans Hanen and Munier (1994b).

Dans la figure 1.3 nous donnons un autre exemple de graphe G de précédences. Chaque arc est caractérisé par deux valeurs, (θ, ω), la latence et la distance, respectivement.



$\mu_1 = (1,2,3,4,1)$ $\quad h(\mu_1) = 4$
$\mu_2 = (1,2,3,5,4,1)$ $\quad h(\mu_2) = 3$
$\mu_3 = (1,2,3,5,6,7,4,1)$ $\quad h(\mu_3) = 3,33$

FIGURE 1.3 – Circuit critique de valeur $\lambda = 4$

Dans ce graphe nous avons trois circuits $\{\mu_i\}_{1\leq i\leq 3}$. Pour chacun d'eux, nous calculons la valeur de $h(\mu_i)$. Le circuit critique est le circuit μ_1, car la valeur $h(\mu_1)$ est maximale. Nous pouvons voir que la valeur $h(\mu_1) = 4$ est égale à la valeur du temps de cycle $\lambda = 4$ dans la figure 1.4 qui représente l'ordonnancement optimal correspondant au graphe de précédences ci-dessus.

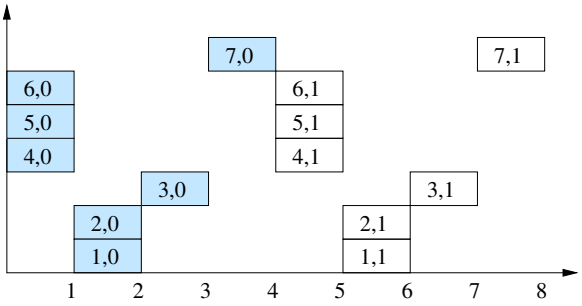


FIGURE 1.4 – Exemple d'ordonnancement avec $\lambda = 4$.

Avec une valeur de λ fixée, nous obtenons un graphe avec les valeurs des arcs égales à $\theta_i^j - \lambda\omega_i^j$. Dans la figure 1.5, nous donnons des graphes G où chaque arc est caractérisé par la valeur $\theta_i^j - \lambda\omega_i^j$. Deux cas sont présentés, le premier cas avec une valeur fixée à $\lambda = 4$ et le deuxième avec une valeur fixée à $\lambda = 3$.

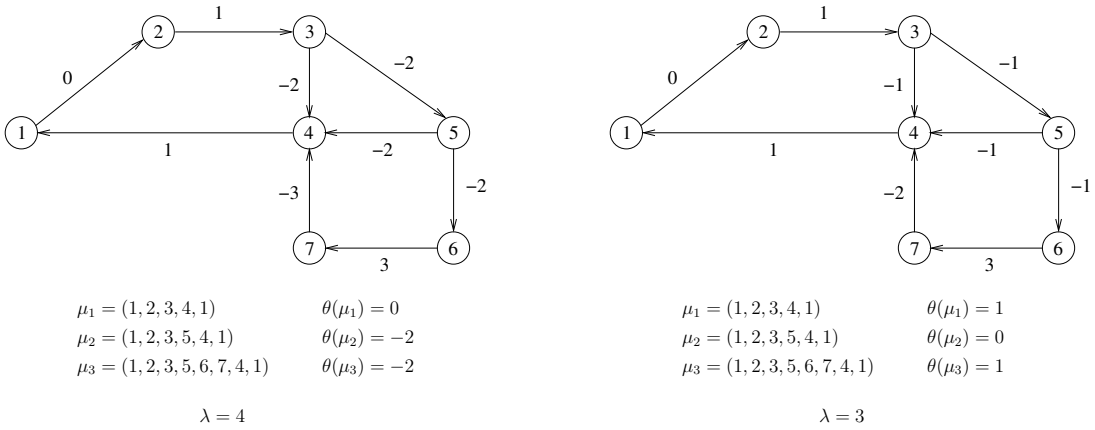


FIGURE 1.5 – Graphes avec des arcs caractérisés par la valeur $\theta_i^j - \lambda\omega_i^j$ pour λ fixée.

La valeur correspondant au circuit critique est $\lambda = 4$. Nous remarquons l'existence de circuits de longueur $\theta(\mu_i)$ positive avec $\lambda = 3$ (contrairement au graphe avec $\lambda = 4$).

1.2.2 Problèmes d'ordonnancement cyclique sous contraintes de ressources de type "machine"

Nous donnons maintenant un bref état de l'art des problèmes d'ordonnancement cyclique sous contraintes de ressources de type "machine", c'est-à-dire ne pouvant exécuter qu'une tâche à la fois. Nous renvoyons le lecteur à Levner *et al.* (2010) pour les détails sur les problèmes présentés ci-dessous.

Ordonnancement cyclique dans les cellules robotisées

Dans le domaine des systèmes de production, il est très courant que les machines soient programmées pour exécuter des tâches dans un intervalle de temps. Généralement, les machines sont groupées en cellules spécialisées dans une partie spécifique du processus de production. A l'intérieur de chaque cellule, le traitement des matériaux est réalisé par un ou plusieurs robots. Les robots peuvent être par exemple des véhicules à guidage automatique. Les principales recherches sur l'ordonnancement cyclique dans les cellules robotisées sont généralement présentées dans le cadre des problèmes de flowshop robotique.
Un problème de flowshop robotique est défini par m machines, $M_1, M_2, ..., M_m$, une station d'entrée M_0, une station de sortie M_{m+1} et un ou plusieurs robots. Les robots effectuent toutes les opérations de manutention des matériaux dans la cellule. Des exemples d'opérations sont le transport des pièces entre les machines et les stations, le chargement et le déchargement des pièces vers et depuis les machines et la station, etc. Au début toutes les pièces sont disponibles dans la station d'entrée M_0, puis elles doivent être traitées de façon séquentielle sur $M_1, M_2, ..., M_m$. Elles sont enfin délivrées à la station de sortie M_{m+1}. Un problème plus général de flowshop robotique est la spécification des mouvements des robots et des séquences pour prendre des pièces dans la station d'entrée et l'ordonnancement de ces opérations pour effectuer les séquences.
De façon générale, le problème de flowshop robotique (Crama *et al.*, 2000) consiste à déterminer l'ordre des pièces dans la station d'entrée, la séquence des activités des robots et les dates de début et de fin pour ces activités. Dans le cas où le nombre de pièces est fini, l'objectif consiste généralement à minimiser le makespan de l'ordonnancement. Cependant, d'autres modèles considèrent un nombre de pièces infini et l'objectif est la maximisation du ratio de rendement de l'ordonnancement (Crama *et al.*, 2000). Les exigences pour le traitement de la pièce j sur la machine M_i peuvent être modélisées au moyen d'une fenêtre de traitement $[L_i^j, U_i^j]$ telle que le temps utilisé pour la pièce j sur la machine M_i doit être compris entre L_i^j et U_i^j. Crama *et al.* (2000) ont travaillé sur le problème de flow shop robotisé avec des pièces identiques. Le but est de trouver un ordonnancement 1-périodique avec un temps de cycle minimum. Il a été montré que pour un cycle, l'ordonnancement optimal 1-périodique peut être calculé en temps polynomial. Dans le cas de pièces identiques et avec la considération d'un temps de déplacement des robots arbitraire, Crama *et al.* (2000) ont établi que le problème est NP-difficile quand les temps de calcul satisfont l'égalité triangulaire. Brauner (2008) donne des concepts de base et des outils pour le traitement de la production cyclique. Elle a traité le problème de K cycles de production où K pièces entrent et sortent de la cellule robotisée. Brauner and Finke (2001) ont proposé des approches par graphes et des approches par algèbres sur les résultats de Dawande *et al.* (2005). Ils ont montré que dans les cas de deux et de trois machines le taux maximum de production peut être atteint en répétant un cycle particulier qui produit une seule pièce, alors que pour quatre machines ceci n'est pas vrai. Gultekin *et al.* (2006) ont travaillé sur le problème de cellules robotisées à deux machines et pièces identiques, en considérant des contraintes de disponibilité des outillages. Ils ont supposé qu'un ensemble d'opérations ne peut être exécuté que sur la première machine, pendant qu'un autre ensemble d'opérations doit être exécuté sur la deuxième machine. L'ensemble des opérations restantes doit être exécuté sur d'autres machines. Le problème considéré est de trouver l'allocation des opérations restantes aux machines, ainsi que le cycle optimal des mouvements des robots. Il a été montré que la solution optimale peut être un cycle d'une ou de deux unités pour les mouvements des robots. Kats and Levner (1997) considèrent le problème

d'ordonnancement cyclique de jobs identiques dans un flowshop rentrant. Le problème est de trouver le cycle minimum des mouvements des robots, tel qu'exactement un nouveau job entre dans le système et exactement un job complet quitte le système pendant le cycle. Un algorithme polynomial a été présenté. Che and Chu (2009) ont considéré le problème d'ordonnancement cyclique multi-niveaux d'une cellule robotisée sans attente. Le problème a été caractérisé par un ordonnancement cyclique $r-$niveau ($r \geq 1$) dans lequel r pièces identiques, avec un temps de traitement constant, entrent et sortent de la cellule dans chaque cycle. Un algorithme pour trouver le nombre minimum des robots pour tous les temps de cycle réalisables $r-$niveau a été donné. Un résultat intéressant de ce travail est que la minimisation du nombre de robots pour l'ordonnancement $r-$ niveau peut être transformé en un problème d'affectation si le temps entre r pièces est fixé par rapport au début de cycle. Middendorf and Timkovsky (2002) ont travaillé sur le problème d'ordonnancement d'un atelier cyclique. Dans ce problème, l'environnement est constitué de machines où tous les jobs passent par les machines en utilisant les même routes comme dans un flowshop. Mais dans ce cas, il est possible d'utiliser les machines plusieurs fois dans la même route. Dans ce travail, il est montré que les problèmes de flowshop réentrant, de flowshop robotisé et de flowshop de boucles réentrantes sont des cas particuliers de problèmes d'atelier cyclique. Ils présentent également des problèmes résolubles en temps polynomial, en temps pseudo-polynomial et des problèmes NP-difficiles, ainsi que des garanties de performance sur certains types de problèmes.

Le Hoist Scheduling Problem (HSP)

Le hoist scheduling problem ou problème de robot de galvanoplastie est rencontré dans le domaine des industries électro-métallurgiques (Dupas, 2004). Ce problème concerne l'utilisation d'un ou plusieurs robots pour le transport de pièces sur le même rail, un des objectifs est donc d'éviter les collisions. Une séquence d'immersion dans un ensemble de réservoirs est définie pour chaque pièce. Chaque immersion a besoin d'un temps de traitement et doit être exécutée dans un intervalle fixé. L'objectif consiste à minimiser le temps de cycle du processus. Il a été prouvé que le problème est NP-difficile dans le cas d'un robot unique et d'un seul job ou type de pièce (Lei and Wang, 1989; Che and Chu, 2005).

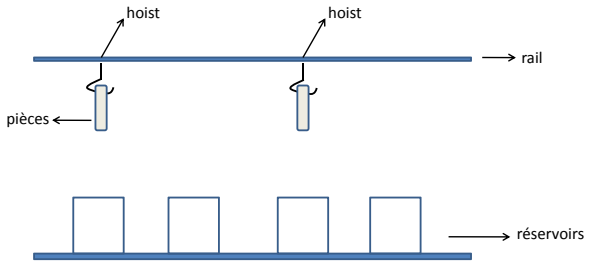


FIGURE 1.6 – Exemple d'une séquence d'immersion avec 2 hoist ou robots.

Chen *et al.* (1995) ont proposé un algorithme de branch-and-bound pour trouver un ordonnancement cyclique pour les mouvements des robots. Les solutions non réalisables sont éliminées par l'algorithme. Ceci permet de réduire le temps d'exécution. La recherche des bornes inférieures et la détection de solutions non réalisables sont basées sur la résolution de problèmes linéaires de complexité polynomiale. Phillipsa and Ungerb (1976) ont étudié le cas 1-cycle. Dans ce problème un job est introduit et extrait des réservoirs dans chaque cycle. Ils ont utilisé un modèle de programmation linéaire mixte pour un problème de circuits électroniques qui doivent être traités dans un ensemble de réservoirs. Lei and Wang (1989) ont montré que le problème de hoist scheduling cyclique est NP-complet.

Systèmes flexibles de production manufacturière

L'ordonnancement cyclique est utilisé dans la production de petites et moyennes séries. Il permet d'ordonnancer des opérations dans un cycle et de répéter ce cycle jusqu'à ce que la demande requise soit complète. Ce problème est défini pour la production de pièces de différents types qui doivent être produites dans un cycle de temps. Les ressources sont partagées entre plusieurs opérations. Chaque pièce est affectée à une séquence de production répétitive. La production des pièces peut être une séquence de montage ou de démontage de différents composants. Une ressource est affectée à un certain nombre de tâches requises dans une période de temps et la préemption n'est pas autorisée. Ce problème est un problème complexe composé lui-même de plusieurs problèmes NP-difficiles (Dupas, 2004). Korbaa and Gentina (2000) montrent que si la demande est fluctuante et si le nombre de pièces dans le processus est petit, les techniques d'ordonnancement cyclique conduisent à des solutions optimales. Dans le cas de la production cyclique manufacturière réentrante, Kenji (1999) considère la maximisation du taux de production et la minimisation du temps de l'ordonnancement. Il propose une méthode cyclique dans laquelle un ensemble de jobs est traité comme un lot qui entre de façon cyclique dans le système de production. La taille du lot est déterminée en maximisant le taux de production. Une méthode pour définir la séquence des lots est proposée afin de minimiser le temps d'ordonnancement. Pour

plus de détails sur ce type de problème, nous renvoyons le lecteur à Ouajdi *et al.* (1997); Xu *et al.* (2002).

Le Job Shop Cyclique

Ce problème considère un ensemble d'opérations qui doit être traité de façon cyclique de manière à compléter un certain nombre requis de tâches pour chaque job. L'objectif est de maximiser le débit en un temps de cycle minimum. Dans le cas du job shop périodique, Hillion and Proth (1989) ont montré que si un ordonnancement cyclique de tâches est donné et si le nombre initial de jobs en attente pour chaque machine est connu, le problème devient alors un problème d'ordonnancement cyclique de base et peut être résolu en un temps polynomial.

Roundy (1992) a présenté la définition de *sélection* et il a proposé un algorithme exact pour un problème cyclique NP-difficile. Dans ce problème, l'objectif est la minimisation simultanée de la période et du nombre de travaux en cours dans un job shop cyclique avec un seul job.

Hanen and Munier (1994b) montrent que la définition de *sélection* peut être étendue à des problèmes cycliques à travers des contraintes uniformes, ce qui implique que l'ensemble de solutions présentent une structure robuste. Soit une contrainte disjonctive entre deux tâches i et j utilisant le même processeur, quel que soit l'occurrence d'exécution de ces deux tâches $< i, q >$ et $< j, l >$, la contrainte peut être écrite comme suit

$$\exists\ q_i^j \in Z,\ \sigma_j \geq \sigma_i + p_i - q_i^j \lambda$$

tel que $q_i^j + q_j^i = 1$. q_i^j indique que pour une exécution $< i, q >$, la prochaine exécution de j sur le processeur est $< j + q + q_i^j >$.
Dans ce travail, les auteurs définissent la *sélection* d'une disjonction entre i et j comme le choix d'une valeur $q_i^j \in Z$. Une *sélection* est complète si toutes les disjonctions ont été sélectionnées. Un algorithme de branch-and-bound basé sur la construction de sélections partielles est proposé dans Hanen and Munier (1994b). Pour plus de détails sur le job shop cyclique, le lecteur peut se référer à Boussemart *et al.* (2002); Brucker and Kampmeyer (2008, 2005); Roundy (1992); Seo and Lee (2002); Kampmeyer (2006).

Machines parallèles

Dans la conception de compilateurs sur des architectures parallèles, l'ordonnancement d'une boucle peut influencer l'efficacité du compilateur. Une boucle est définie par un ensemble de tâches génériques qui doivent être exécutées plusieurs fois. Le problème concerne l'ordonnancement des instructions de manière à finir le programme dans un temps minimum.

Dans un problème de machines parallèles un ensemble d'opérations $\{O_i\}_{1 \leq i \leq n}$ est exécutées sur m processeurs identiques. Pour chaque exécution d'une opération, il est nécessaire d'utiliser un des m processeurs pendant p_i unités de temps. La préemption n'est pas autorisée. Les contraintes de ressources sont liées au nombre limité de processeurs et au nombre des opérations qui sont exécutées à une période donnée. Un ensemble de contraintes de précédence est également à considerer. Un ordonnancement réalisable est un choix de dates de début $\{\sigma_i\}_{1 \leq i \leq n}$, tel que les contraintes de ressources et de précédence sont respectées (Dinechin, 2004). L'exemple suivant correspond à une machine parallèle appelé MIMD (multiprocesseurs / multicomputers). Ce type de machine dispose de 3 processeurs. Chaque processeur peut exécuter un ensemble d'instructions

similaires ou non sur des données différentes. L'ensemble d'instructions est généré par sa propre unité de contrôle et la communication de données ou de résultats entre les processeurs peut se faire à travers une mémoire commune.

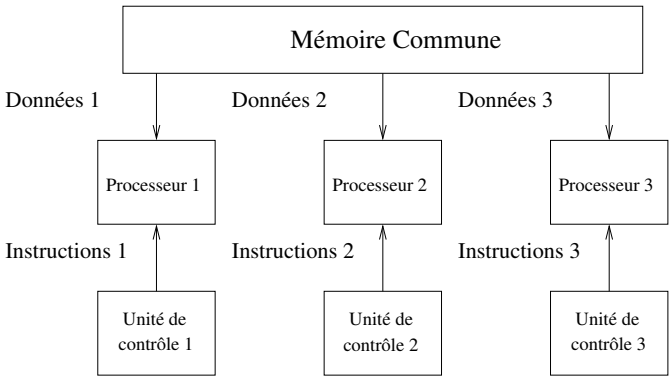


FIGURE 1.7 – Exemple de machine parallèle MIMD.

Différentes approches ont été proposées pour résoudre le problème avec machines parallèles. Hanen and Munier (1995) ont montré que la notion de *sélection* donnée pour le job shop périodique peut être étendue pour le problème de processeurs identiques parallèles. En utilisant la notion de permutation d'un ordonnancement, Hanen and Munier (1994b) proposent que si $< i, q >$ représente une occurrence de i sur le processeur k, la prochaine exécution de i sur le même processeur est alors $< i, q + \delta(s) >$ et elle est exécutée $\lambda\delta(s)$ unités du temps après. $\delta(s)$ représente le degré de la permutation s. En conséquence, le degré de s détermine la périodicité de l'ordonnancement. Hanen and Munier (1995) ont montré que les graphes composés uniquement d'un circuit de longueur m sont dominants. Hanen and Munier (1994b) présentent également un algorithme glouton qui est une extension de l'algorithme présenté dans Hanen and Munier (1995). Il a été prouvé qu'il existe toujours une solution à partir d'une énumération complète des nœuds d'un processus glouton. La complexité de cet algorithme est $O(n^3 log(n))$. Chrétienne (2000) montre un résultat très semblable à la borne de Graham pour le problème non-cyclique. Dans ce cas, l'auteur montré que le temps de cycle moyen d'un ordonnancement périodique est au moins $(2 - (\min\{H^*, m\})/m)$ fois le temps de cycle moyen minimum, où H^* est plus grand ou égal à la distance ou hauteur minimale du circuit réduit de précédence. Hanen and Munier (2009) introduisent des contraintes linéaires entre deux tâches génériques et caractérisent le comportement asymptotique d'un graphe linéaire. Elles ont développé un algorithme pour calculer les fréquences optimales et évaluer la performance d'un graphe linéaire dans différentes situations. Elles ont également montré qu'un graphe linéaire peut être associé à un graphe uniforme avec le même comportement asymptotique. Munier (1991) a montré que dans le cas des tâches à durées unitaires et avec une structure de contraintes de précédence simple, trouver un ordonnancement périodique est un problème NP-difficile dans le cas des machines parallèles. Ce problème est cependant polynomial dans le cas de deux machines. Gasperoni and Schwiegelshohn (1994) présentent un algorithme qui consiste à utiliser la solution d'un problème non-cyclique pour trouver un ordonnancement périodique réalisable pour un problème cyclique. Dans ce cas, si λ_{opt} représente le temps de cycle optimal, λ_L représente le temps de cycle donné pour l'algorithme et ρ_{max} représente le temps maximum de traitement d'une tâche, alors il a été prouvé

que $\lambda_L \leq (2 - \frac{1}{m})\lambda_{opt} + \frac{m-1}{m}(\rho_{max} - 1)$. Cette approche a été utilisée pour le problème d'ordonnancement d'un graphe uniforme sur des processeurs pipeline avec un ratio de performance approximatif de 2. Serafini and Ukovich (1989) proposent une généralisation des contraintes de précédence uniformes appelée contraintes *span*. Ces contraintes sont définies entre deux tâches génériques par $(t_i - t_j) mod \lambda \in \Delta_{ij}$, où Δ_{ij} est un intervalle cyclique et t représente l'ordonnancement des tâches génériques. Le problème *span* consiste à trouver un ordonnancement périodique de cycle λ en utilisant les contraintes *span*. Ils ont établi que si le graphe de contraintes *span* est un arbre, le problème peut être alors résolu en temps polynomial. Un algorithme de branch-and-bound basé sur cette relaxation est proposé. Les auteurs également montrent que le problème est très proche du problème du voyageur de commerce. Serafini and Ukovich (1989) ont implémenté les contraintes *span* sur m processeurs identiques et montrent que trouver le nombre minimum de processeurs exécutant un ordonnancement est un problème NP-difficile. Ce problème peut être identifié à un problème de flot dans un graphe biparti, à un problème de voyageur de commerce ou à un problème de coloration de graphes (Hanen and Munier, 1994b).

1.3 Ordonnancement modulo sous contraintes de ressources

Avant d'introduire l'ordonnancement modulo, il est important de situer ce problème par rapport au contexte de ce travail. En introduction, nous précisons que le terme ordonnancement modulo désigne un ordonnancement périodique tel que l'allocation des ressources pour une opération donnée ne change pas d'une itération sur l'autre.

1.3.1 Contexte : parallélisme d'instructions et pipeline logiciels

Les systèmes embarqués sont des systèmes d'exploitation conçus pour fonctionner sur des machines de petite taille. Les applications déployées sur de tels dispositifs sont principalement dédiées au multimédia et au traitement d'image ou du signal, comme par exemple, la lecture de fichiers vidéos, de musique ou encore des animations flash. Une caractéristique des systèmes embarqués est leur gestion avancée de l'énergie et leur capacité à fonctionner avec des ressources limitées. Aujourd'hui, l'application de parallélisme au niveau instruction permet une simplification de leur architecture.
Le parallélisme au niveau instruction *(Instruction level paralleslism (Rau and Fisher, 1993))* permet l'exécution de plusieurs opérations de façon parallèle, ce qui entraîne une accélération significative des performances. Un exemple de parallélisme au niveau instruction sur des processeurs est donné par les architectures super-scalaires et VLIW. En outre, dans les architectures VLIW une partie de la gestion du parallélisme d'instructions ("pipeline logiciel") est géré dans le compilateur.
Dans l'exemple de boucle suivant, l'opération w[i+1] dépend des résultats de l'opération z[i]. Les opérations z[i] et x[i] ne dépendent pas d'une autre opération, de sorte qu'elles peuvent être traitées simultanément.

```
int n, int b, int z[], int c[], int w[];
Pour (int i=1;i<=n;i++)
{
z[i] = c[i] + b;
```

```
x[i] = c[i]^2;
w[i+1] = z[i] + c[i];
}
```

Si nous supposons que chaque opération peut être effectuée en une seule unité de temps, ces trois instructions peuvent être alors exécutées dans un total de deux unités de temps.
La technique de "pipeline logiciel" (Lam, 1988) permet d'aller encore plus loin dans le parallélisme et dans l'imbrication des instructions d'une boucle. Selon le groupe de recherche sur l'ordonnancement (Carlier *et al.*, 1993), les pipelines sont des machines composées de processeurs parallèles spécialisés appelés unités fonctionnelles qui échangent des données au travers de registres plus ou moins partagés. Leur contrôle est assuré au moyen d'un microprogramme, donc chaque instruction gère l'activation des unités fonctionnelles ainsi que les chemins de données pendant une unité de temps.
Plus particulièrement, les techniques de pipeline logiciel sont utilisées dans le domaine de la synthèse haut niveau de circuits. Une des principales tâches de la synthèse de haut niveau est l'ordonnancement dans le temps des opérations que le circuit doit réaliser (Eddine *et al.*, 2002). Généralement, les circuits doivent réaliser des traitements itératifs qui sont équivalents à une boucle infinie. Dans les traitements itératifs, l'ordonnancement doit déterminer un ordre statique pour les exécutions répétitives des opérations génériques. Les problèmes de la synthèse haut niveau peuvent être ainsi identifiés comme des problèmes à machines parallèles. L'ordonnancement doit donc exploiter les deux niveaux de parallélisme présents dans les traitements itératifs : le parallélisme entre les opérations d'une même itération et le parallélisme entre les opérations qui appartiennent à des itérations différentes (Eddine *et al.*, 2002). Le problème de la synthèse haut niveau peut être ainsi modélisé par un problème d'ordonnancement général où l'objectif est la minimisation du nombre de ressources utilisées pour ordonnancer les opérations dans une fenêtre de temps (Sevaux *et al.*, 2011). Les opérations appartenant à différentes itérations doivent être exécutées de façon parallèle (Rossi and Sevaux, 2008). Pour plus de détails sur le problème de la synthèse haut niveau et le pipeline logiciel, nous renvoyons à Šůcha and Hanzálek (2011, 2008).
En revenant au problème d'ordonnancement d'instructions, l'objectif du pipeline logiciel est de minimiser λ de sorte que les différentes unités fonctionnelles dans l'architecture soient utilisées efficacement. Plusieurs approches ont été développées depuis l'invention du pipeline logiciel. Dans cette thèse nous allons nous concentrer sur une technique de pipeline logiciel : **le modulo scheduling** ou **ordonnancement modulo.**
La formulation générale de pipeline logiciel a été donnée par Rau and Fisher (1993). Ils ont introduit la contrainte *modulo* et des bornes inférieures pour l'intervalle d'initiation ou période λ. Une amélioration du pipeline logiciel, connue comme *modulo expansion,* a été donnée par Lam (1988). Cette amélioration consiste en une combinaison de deux techniques : une technique d'ordonnancement par constructions itératives et une technique de réduction hiérarchique. Cette hybridation permet de traiter de multiples blocs comme des opérations d'un bloc de base. Plus tard, Rau (1994) propose une nouvelle méthode d'ordonnancement modulo, appelée *modulo scheduling itératif.* Dans ce cas, un ordonnancement est recherché à partir d'une valeur de λ (généralement la plus petite valeur possible). Si aucun ordonnancement valide n'est pas trouvé, alors la valeur de λ doit s'incrémenter jusqu'à trouver un ordonnancement valide. Les principaux avantages du *modulo scheduling itératif* sont : la génération d'ordonnancements optimaux en λ pour 96% des

boucles, l'obtention de temps d'exécution totale des boucles qui sont seulement 2.8% plus long que la borne inférieure et l'ordonnancement de seulement 59% d'opérations additionnelles par rapport à l'ordonnancement acyclique (Rau, 1994).

Le Modulo scheduling ou **ordonnancement modulo** est un cas particulier d'ordonnancement cyclique. C'est un ordonnancement périodique où toutes les tâches ont le même intervalle d'initiation λ et où l'allocation des ressources est conservée pour toutes les instances des tâches. Autrement dit, un problème d'ordonnancement modulo correspond à un problème d'ordonnancement périodique où l'on considère uniquement l'ensemble des tâches génériques. Ceci simplifie de manière significative le problème à résoudre. Dans ce type de problèmes, les contraintes de précédence correspondent aux contraintes de précédence données par (1.8) qui ont été obtenues en remplaçant la date de début périodique ou (*date de début modulo*) dans les contraintes de précédence uniformes (1.3).

1.3.2 Formulation du problème d'ordonnancement modulo sous contraintes de ressources

Soit R un ensemble de m ressources. Chaque tâche générique $i \in O$ demande une quantité b_i^s de chaque ressource de type $s \in R$. Une ressource s a une disponibilité limitée B_s telle que $b_i^s \leq B_s$ pour toute tâche $i \in O$. Le problème consiste à ordonnancer dans le temps l'exécution répétitive d'un ensemble de tâches liées par des contraintes de précédence, en utilisant un nombre limité de ressources. Dans le problème d'ordonnancement cyclique, nous considérons qu'à tout instant de l'exécution des occurrences de tâches, l'usage d'une ressource ne doit pas dépasser la disponibilité de celle-ci.

Soit $\{\sigma_i\}_{1 \leq i \leq n}$ la date de début d'un ensemble de tâches génériques $\{O_i\}_{1 \leq i \leq n}$, un problème d'ordonnancement modulo sous contraintes de ressources (Resource-Constrained Modulo Scheduling désigné ci-après par son acronyme RCMSP) est défini par Dinechin (2004) avec l'ordre lexicographique entre les deux objectifs :

$$\min Lex(\lambda, \sum_{i=1}^{n} w_i \sigma_i) \tag{1.10}$$

sous les contraintes suivantes :

$$\sigma_i + \theta_i^j - \lambda \omega_i^j \leq \sigma_j \quad \forall (i,j) \in G.$$

$$\sum_{i \in A(t)} b_i^s \leq B_s, \quad \forall s \in \{1, \ldots, m\}, \forall t \in \mathbb{N}$$

le premier ensemble de contraintes correspondent aux contraintes de précédence. Le deuxième ensemble de contraintes correspondent aux contraintes de ressources "modulo". nous remarquons que si $\lambda \geq 1$, (1.2) implique que plusieurs instances de chaque tâche ne peuvent pas être ordonnanceés de façon parallèle. En conséquence, l'ensemble des instances des tâches à l'instant $t \in \mathbb{N}$ est l'ensemble $A(t) = \{i \in \{1, \ldots, n\} | \exists q \in \mathbb{N}, \sigma_i^q = t\}$. Si nous considérons un instant du temps

"générique" $\tau \in \{0, \ldots, \lambda - 1\}$ et l'ensemble $F(\tau) = \{i \in \{1, \ldots, n\} | \exists q \in \mathbb{N}, \sigma_i = \tau + q\lambda\}$ de tâches de dates de début σ_i telles $\tau = \sigma_i \mod \lambda$. En raison de la périodicité de l'ordonnancement, la contrainte de ressource peut être simplifiée et il est également possible de démontrer qu'il existe toujours $\mathcal{T} \in \mathbb{N}$ tel que pour $t \geq \mathcal{T}$, $A(t) = F(\tau)$ et pour chaque $t < \mathcal{T}$, $A(t) \subset F(\tau)$ où $t = \tau + q\lambda$. Nous pouvons ainsi écrire la contrainte de ressource comme la contrainte de ressources "modulo" suivante :

$$\sum_{i \in F(\tau)} b_i^s \leq B_s, \quad \forall s \in \{1, \ldots, m\}, \forall \tau \in \{0, \ldots, \lambda - 1\} \tag{1.11}$$

Le problème est simplifié par la recherche d'un ordonnancement commun qui initie les itérations successives qui seront répétés avec un écart de λ unités. L'objectif principal du problème d'ordonnancement modulo sous contraintes de ressources est de minimiser la période λ. Un objectif secondaire est de minimiser le makespan des itérations (plus grande date de fin des tâches génériques), ou bien une fonction pondérée des dates de fin des opérations génériques. En ordonnancement d'instructions, chaque variable intermédiaire (donnée) produite par une opération, dois être sauvegardée dans un registre tant que toutes les opérations utilisant cette donnée n'ont pas été exécutées. La période de vie d'une variable est l'intervalle défini par l'instant de sa création et l'instant de sa dernière utilisation. Les variables qui ont des périodes de vie disjointes peuvent partager le même registre. L'un des principaux critères d'optimisation est la minimisation du nombre de registres (Eddine *et al.*, 2002). Dans le cas où le nombre d'allocation de registres est une contrainte significative, l'objectif secondaire peut être utilisé pour favoriser la minimisation de la durée de vie cumulée des registres (bien que nous ne prenions pas en compte explicitement les registres dans cette étude).

Pour illustrer le problème d'ordonnancement modulo sous contraintes de ressources, nous considérons le graphe de précédences donné dans l'exemple 1.1 avec des demandes en ressources décrites dans la table suivante :

Ressources	R_1	R_2	R_3	R_4
Disponibilité	4	1	1	2
Opérations	R_1	R_2	R_3	R_4
1	1	0	0	0
2	1	1	0	0
3	1	1	0	0
4	1	1	0	0
5	1	0	0	0
6	1	1	0	0
7	1	0	0	0
8	1	0	0	0
9	1	0	1	1
10	1	0	0	0

TABLE 1.1 – Disponibilité et demande de ressources.

L'ordonnancement de la figure 1.8 respecte les contraintes de précédence et de ressources spécifiées, avec une période optimale $\lambda = 4$.

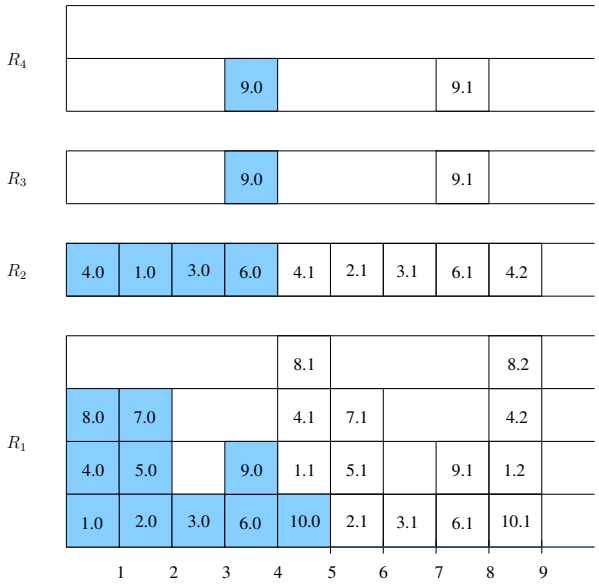


FIGURE 1.8 – Ordonnancement modulo sous contraintes de ressources optimal.

Dans cette thèse nous nous sommes intéressés au cas d'architectures VLIW. Plus particulièrement, nous travaillons avec des instances provenant du compilateur ST200 (Dinechin, 2004). Dans ce compilateur, l'allocation des registres n'est pas le problème le plus critique. Ceci est la raison de fixer la minimisation du makespan des itérations comme l'objectif secondaire du problème.

Résolution du problème d'ordonnancement modulo sous contrainte de ressource.

Dans la théorie classique d'ordonnancement modulo (Rau and Fisher, 1993), une méthode itérative est généralement utilisée pour la recherche d'un λ réalisable. Il s'agit de chercher des ordonnancements modulo réalisables pour une valeur de λ fixée. Cette valeur de λ est obtenue comme

$\lambda_{min} = \max(\lambda^{res}, \lambda^{prec})$, où nous notons $\lambda^{prec} = \max\limits_{\mu \ circuit \ de \ G} \frac{\theta(\mu)}{\omega(\mu)}$ et $\lambda^{res} = \max\limits_{1 \leq s \leq m} \frac{\sum_{i=1}^{n} b_i^s}{B_s}$.
λ^{prec} est la valeur minimale de λ tel que le graphe de précédence est réalisable (voir section 1.2.2). λ^{res} est la valeur minimale de λ qui permet que les disponibilités de ressources B_s ne soient jamais dépassées. Le problème d'ordonnancement modulo ne peut pas être résolu avec des techniques classiques d'ordonnancement. Dans le graphe de précédence de ce problème, il est possible de trouver des circuits ou des cycles avec des latences de précédences $(\theta_i^j - \lambda\omega_i^j)$ négatives. De plus, les contraintes de ressource sont de type "cumulatif" et, en même temps, ces contraintes de ressource modulo sont paramétriques par rapport à λ (Dinechin, 2004). Le problème d'ordonnancement modulo sous contraintes de ressources apparaît donc comme un problème d'ordonnancement de projet sous contraintes de ressources avec time lags maximaux et des contraintes de ressource modulo (Dinechin, 2004).
Gasperoni and Schwiegelshohn (1992) ont proposé une approche heuristique basée sur le pré-

traitement de la boucle. La disponibilité des ressources est considérée illimitée. L'objectif est d'effacer quelques nœuds du graphe de précédences de manière à obtenir un graphe de précédence acyclique et de construire ainsi une nouvelle boucle avec des contraintes de ressources limitées. Wang *et al.* (1994) présentent une nouvelle approche appelée *"Decomposed software pipelining"* ou (DSP). La technique consiste en la transformation du vecteur colonne d'instructions (σ_i) en une matrice à deux dimensions. Les lignes représentent les cycles où les opérations sont exécutées ($\sigma_i \mod \lambda$) et les colonnes indiquent l'itération dans la boucle d'origine. Le problème est alors divisé en deux sous-problèmes. En utilisant l'approche DSP, Benabid and Hanen (2009, 2011) présentent une extension des approches de Gasperoni and Schwiegelshohn (1994) et Calland *et al.* (1998) dans le cas de demandes de ressources unitaires avec contraintes de précédence. Les auteures donnent une borne du pire cas pour l'approche DSP avec un algorithme de liste pour la phase de compaction de boucle.

Des formulations de programmation linéaire en nombres entiers pour le problème d'ordonnancement modulo ont été proposées par Eichenberger and Davidson (1997) et Dinechin (2004). Dans Eichenberger and Davidson (1997), la formulation consiste à trouver les dates de débuts des tâches génériques en utilisant deux variables qui représentent les deux sous problèmes cités ci-dessus. Dinechin (2004) propose une formulation de variables entières indexées dans le temps utilisant des variables binaires indicées par le temps pour représenter directement les variables σ_i, inspiré par les formulations classiques du problème d'ordonnancement sous contraintes de ressources données par Pritsker *et al.* (1969) et Christofides *et al.* (1987). Nous reviendrons sur ces formulations dans le chapitre suivant. Govindarajan *et al.* (1994) présentent une nouvelle approche en utilisant aussi des contraintes de ressources dans le pipeline logiciel. Cette approche concerne une formulation de programmation linéaire en nombres entiers en intégrant la minimisation du nombre de tampons donnant une bonne approximation à la minimisation du nombre de registres. Une solution optimale est obtenue pour plus de 91% des instances en 30 secondes. de Dinechin (2007) propose une heuristique de recherche à grand voisinage (LNS) pour résoudre une formulation de variables entières indexées dans le temps. Dans ce cas, les résultats montrent que LNS est très efficace pour trouver une solution dans le période $\lambda - 1$ à partir de une période λ donné. de Dinechin (2004) propose une approche basée sur le déroulage de l'ordonnancement modulo et l'application de techniques d'ordonnancement acyclique pour garantir la régularité de l'ordonnancement déroulé obtenu. Lombardi *et al.* (2011) ont implémenté une méthode telle que la valeur de la période λ n'est pas fixée *apriori*. Une heuristique pour l'ordonnancement cyclique basée sur une recherche itérative d'aplanissement (Iterative flatenning) a été testée sur notre ensemble d'instances d'expérimentations. Nous invitons le lecteur à lire les documents suivants pour plus de détails sur la résolution du problème d'ordonnancement modulo sous contraintes de ressources (Aiken *et al.*, 1995; Calland *et al.*, 1996, 1998; de Dinechin, 1995, 1997, 1999, 2004).

1.3.3 Modèle d'architecture ST200

Nous présentons le cas particulier du problème d'ordonnancement modulo sous contraintes de ressources, correspondant au compilateur du processeur ST200 qui est basé sur une technologie commune entre Hewlett-Packard laboratoires et STMicroelectronics (Dinechin, 2004). Ce processeur exécute quatre opérations par cycle de temps ou période avec une opération de contrôle (goto, jump, call, return), une opération de mémoire (load, store, prefetch) et deux opérations de multiplication par cycle. Les contraintes de ressource du processeur ST200 sont

modélisées avec des ressources cumulatives, autrement dit, des ressources qui peuvent être utilisées par plusieurs tâches simultanément mais en nombre limité (comme définies dans la section précédente). Pour une description détaillée de processeur ST200, nous nous référons à Dinechin (2004). Dans le cas du processeur ST200, la table 1.1 présente six ressources. Width représente le nombre maximum d'instructions par cycle, IMX représente la valeur étendue, MUL représente l'operateur de multiplication, MEM représente l'unité de mémoire et CTL l'unité de contrôle. Un ressource artificielle ODD représente la contrainte de position des opérations. Pour l'ordonnancement des instructions, les demandes en ressources pour l'exécution des tâches sont représentées par des classes de ressources, **ALU, ALUX, MUL, MULX, MEM, MEMX, CTL.** Les classes de ressources ALU, MUL, MEM et CTL correspondent respectivement au cycle simple, aux opérations de multiplication, de mémoire et contrôle. Les classes AUX, MULX et MEMX correspondent aux opérations avec un opérande immédiat étendu (Dinechin, 2004).

Ressource	Width	IMX	MUL	MEM	CTL	ODD
Disponibilité	4	2	2	1	1	2
Classe de Ressource	Width	IMX	MUL	MEM	CTL	ODD
ALU	1					
ALUX	2	1				1
MUL	1		1			1
MULX	2	1	1			1
MEM	1			1		
MEMX	2	1		1		1
CTL	1				1	1

TABLE 1.2 – Disponibilité des ressources cumulatives et classes de ressources.

Pour illustrer le problème d'ordonnancement modulo sous contraintes de ressource sur le pipeline logiciel, nous montrons dans la figure 1.9 un exemple (de Dinechin *et al.* (2010), p. 270) d'un programme en langage C et ses opérations correspondantes sur le processeur ST200. Le processeur exécute quatre opérations par unité de temps avec une opération de contrôle au maximum (goto, jump, call, return), une opération mémoire (load, store, prefetch) et deux opérations de multiplication. Le temps de traitement de chaque opération est unitaire et les latences θ_i^j entre les opérations varient entre 0 et 3 unités du temps.

```
int                                  L?__0_8:
prod(int n, short a[], short b) {    LDH_1 g131 = 0, G127
   int s=0, i;                       MULL_2 g132 = G126, g131
   for (i=0;i<n;i++) {               ADD_3 G129 = G129, g132
      s += a[i]*b;                   ADD_4 G128 = G128, 1
   }                                 ADD_5 G127 = G127, 2
   return s;                         CMPNE_6 b135 = G118, G128
}                                    BRF_7 b135, L?__0_8
```

FIGURE 1.9 – Exemple de programme en langage C et ses opérations ST200 correspondantes.

La table 1.3 présente les contraintes de ressources et la figure 1.10 montre le graphe de précédence. Chaque nœud représente une opération générique et chaque arc représente une précédence uniforme. Deux variables fictives représentent le début et la fin de l'ordonnancement. Les paires de valeurs (θ_i^j, ω_i^j) sont montrées pour chaque arc.

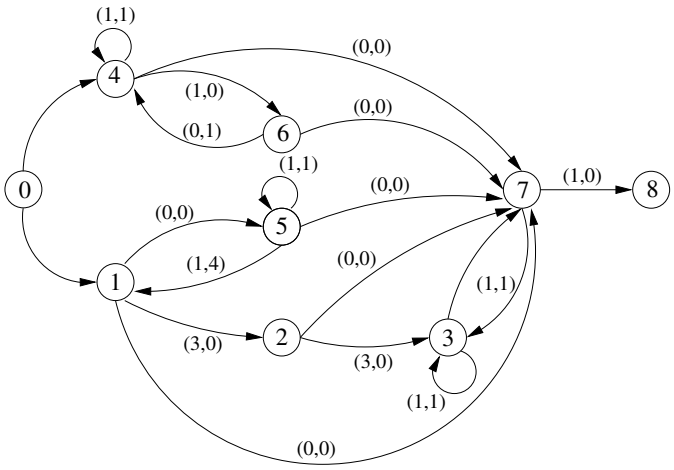


FIGURE 1.10 – Graphe de précédence uniforme.

Ressource	Issue	Memory	Control	Align
Disponibilité	4	1	1	2
ALU	1			
ALUX	2			1
MUL	1			1
MULX	2			1
MEM	1	1		
MEMX	2	1		1
CTL	1		1	1

TABLE 1.3 – Disponibilité des ressources cumulatives

La figure 1.11 montre le graphe de précédences avec les valeurs d'arcs $(\theta_i^j + \lambda\omega_i^j)$ pour une valeur $\lambda = 2$ et l'ordonnancement modulo correspondant avec $Cmax = 6$ qui permet de respecter les contraintes de précédence et de ressources. Dans le cas de graphe de précédences la transformation est fait en remplaçant les valeurs des latences et distances. Par exemple, la précedence entre le nœud 4 et 6 est caracterisée par une latence $\theta_4^6 = 1$ et une distance $\omega_4^6 = 0$. Donc, avec une valeur de $\lambda = 2$ la nouvelle valeur d'arc est 1.

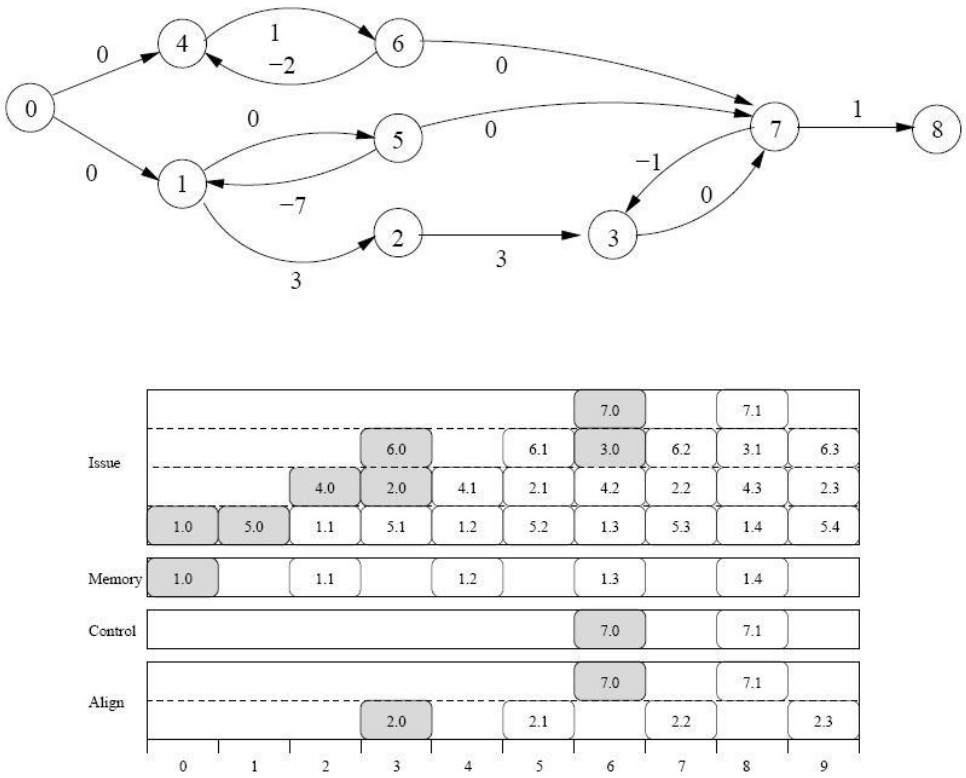


FIGURE 1.11 – Graphe de précédence uniforme et ordonnancement périodique.

1.3.4 Description des instances pour l'expérimentation

Dans cette thèse, nous utilisons deux groupes d'instances. Le premier groupe est constitué de 36 instances à durées unitaires provenant du processeur ST200 de STMicroelectronics, c'est-à-dire des instances correspondant à l'architecture présentée en la section 1.3.3. L'ensemble d'instances ST200 est caractérisé en termes de consommation des ressources principalement unitaires (voir table 1.2).

Le deuxième groupe d'instances correspond à une modification du premier ensemble d'instances. Nous avons changé la capacité des ressources en fixant pour chaque ressource une capacité égale à 10. Ensuite, nous avons généré de façon aléatoire ($U \sim [1, 10]$) la consommation de chaque tâche sur chaque ressource. Les graphes de précédence restent identiques aux graphes du premier groupe d'instances. L'objetif de ce nouveau ensemble d'instances est d'obtenir contraintes de ressources plus fortes. L'instance plus petite, *gsm-st231.10.rcms* est composée de 10 tâches génériques et 42 relations de précédence. L'instance plus grande, *gsm-st231.18.rcms* est composée de 214 tâches génériques et 1063 relations de précédence.

Le tableau 1.4 montre les caractéristiques de chaque instance du compilateur ST200, le nombre d'opérations, (#oper), le nombre de contraintes de précédences (#prec), la borne de précédence (λ^{prec}), la borne de ressource pour les instances industrielles du processeur ST200 (λ^{res} (ind))

et la borne de ressources pour le groupe d'instances modifiées (λ^{res} (mod)). Notons que les contraintes de ressource sont plus serrées dans le cas des instances modifiées.

Instance	#oper	#prec	λ_{prec}	λ^{res} (ind)	λ^{res} (mod)
adpcm-st231.1	86	405	11	21	52
adpcm-st231.2	142	722	38	35	82
gsm-st231.1	30	190	24	12	16
gsm-st231.2	101	462	12	26	59
gsm-st231.5	44	192	4	11	26
gsm-st231.6	30	130	3	7	17
gsm-st231.7	44	192	4	11	28
gsm-st231.8	14	66	1	8	9
gsm-st231.9	34	154	28	8	21
gsm-st231.10	10	42	1	4	6
gsm-st231.11	26	137	20	6	16
gsm-st231.12	15	70	1	8	10
gsm-st231.13	46	210	19	11	27
gsm-st231.14	39	176	5	10	20
gsm-st231.15	15	70	1	8	9
gsm-st231.16	65	323	4	16	38
gsm-st231.17	38	173	8	9	23
gsm-st231.18	214	1063	8	53	120
gsm-st231.19	19	86	2	8	12
gsm-st231.20	23	102	3	6	13
gsm-st231.21	33	154	18	8	20
gsm-st231.22	31	146	18	8	18
gsm-st231.25	60	273	10	16	37
gsm-st231.29	44	192	4	11	28
gsm-st231.30	30	130	3	7	16
gsm-st231.31	44	192	4	11	26
gsm-st231.32	32	138	15	8	21
gsm-st231.33	59	266	4	15	33
gsm-st231.34	10	42	2	4	7
gsm-st231.35	18	80	2	6	12
gsm-st231.36	31	143	2	10	18
gsm-st231.39	26	118	3	8	15
gsm-st231.40	21	103	2	10	12
gsm-st231.41	60	315	2	18	33
gsm-st231.42	23	102	3	6	14
gsm-st231.43	26	115	8	6	15

TABLE 1.4 – Caractéristiques des instances industrielles (processeur ST200) et des instances modifiées

1.4 Conclusion

Dans ce chapitre nous avons introduit le problème d'ordonnancement cyclique de base et le problème d'ordonnancement périodique, ainsi que les conditions nécessaires pour leur résolution. Nous avons présenté les principaux types de problèmes d'ordonnancement cyclique et nous

avons également cité quelques résultats importants qui ont permis la proposition de nouvelles approches et d'extensions plus efficaces. Nous nous sommes intéressés particulièrement au problème d'ordonnancement d'instructions avec contraintes de ressources. Dans ce contexte, nous avons présenté le problème d'ordonnancement périodique (modulo) sous contraintes de ressources comme un modèle du problème d'ordonnancement d'instruction. Nous avons également présenté deux ensembles d'instances de ces problèmes qui seront utilisées pour les expérimentations ultérieures : l'un provenant directement du compilateur pour le processeur ST200, l'autre obtenu en modifiant les contraintes de ressources du premier pour obtenir des demandes en ressources plus générales. Dans le chapitre suivant, nous abordons la recherche de bornes de la période optimale λ et du makespan et la résolution optimale du problème par des formulations de programmation linéaire en nombres entiers.

Chapitre 2

Programmation linéaire en nombres entiers pour l'ordonnancement modulo sous contraintes de ressources

L'objectif de ce chapitre est de présenter les formulations de programmation linéaire en nombres entiers (PLNE) classiques de la littérature pour le problème d'ordonnancement modulo sous contraintes de ressource (RCMSP) et de proposer des bornes inférieures et des solutions optimales pour la période λ et pour l'objectif secondaire en utilisant ces formulations. Nous présentons tout d'abord les notions générales de la PLNE et les méthodes les plus utilisées pour sa résolution. Ce chapitre donne ensuite les formulations PLNE pour le problème d'ordonnancement de projet sous contraintes de ressources (RCPSP) qui sont à la base des formulations PLNE pour le (RCMSP). Ensuite, un exposé des formulations les plus connues pour le (RCMSP) est présenté. Une étude théorique sur la qualité des relaxations est également développée. A partir de cette étude théorique, nous proposons deux théorèmes qui établissent l'équivalence entre les relaxations de ces formulations. Nous présentons enfin les résultats de la PLNE en termes de borne inférieure et de solutions optimales ou réalisables trouvées. Les travaux présentés de ce chapitre ont fait l'objet d'une communication dans une conférence nationale Ayala and Artigues (2009) et font partie d'une soumission dans une revue internationale [DAM] Ayala and Artigues (2011).

2.1 Introduction

Dans le chapitre précédent, nous avons vu que l'objectif principal du problème d'ordonnancement modulo consiste en la minimisation de l'intervalle d'initiation (ou période) λ, suivie d'un objectif secondaire, qui est généralement une fonction qui dépend de la date de début de la tâche générique σ_i, par exemple la durée totale de l'ordonnancement ou $Cmax$, ou bien la somme pondérée des dates de début des tâches. La recherche d'un intervalle minimum d'initiation (ou période) pour le problème considéré est un problème d'optimisation qui peut être résolu avec la programmation linéaire en nombres entiers (PLNE). Différentes formulations de PLNE pour

l'ordonnancement modulo sous contraintes de ressources ont été proposées. Ces formulations sont basées dans la généralisation de la formulation classique non préemptive et indexées par le temps pour le RCPSP (Resource Constrained Project Scheduling Problem)(Pritsker *et al.*, 1969). Ce chapitre présente ainsi en détail les deux formulations de PLNE les plus connues pour résoudre le problème d'ordonnancement modulo sous contraintes de ressources. La première formulation, proposée par Eichenberger and Davidson (1997), est basée sur deux ensembles de variables : une variable binaire y_i^τ, qui spécifie la date de début de l'opération i dans la période générique (telle que $\sigma_i = \sum_{\tau=0}^{\lambda-1} \tau y_i^\tau$) et la variable entière k_i qui spécifie le numéro de la période d'exécution. La deuxième formulation proposée par Dinechin (2004), comporte un seul type de variable binaire x_i^t tel que $y_i^\tau = \sum_{k=0}^{K-1} x_i^{\tau+k\lambda}$ et $k_i = \sum_{k=1}^{K-1} \sum_{t=k\lambda}^{K\lambda-1} x_i^t$ où K représente le nombre de cycles de l'ordonnancement. Les valeurs de K et λ étant fixées, nous pouvons alors représenter l'horizon du temps T par $T = K\lambda$. Dans les deux cas, en suivant l'approche de Christofides *et al.* (1987), des contraintes de précédence structurées et non structurées sont présentées.
L'objectif de ce chapitre est de proposer des bornes inférieures pour la période λ et pour l'objectif secondaire ainsi que des solutions entières à partir des formulations de PLNE pour l'ordonnancement modulo sous contraintes de ressources citées ci-dessus. Le chapitre est structuré de la manière suivante. La section 2.2 est dédiée à la présentation de la PLNE ainsi qu'aux principales méthodes pour sa résolution. La section 2.3 est consacrée à la description du problème d'ordonnancement de projet sous contraintes de ressources qui est le point de départ pour les formulations de base de l'ordonnancement modulo sous contraintes de ressources. Dans la section 2.4, nous exposons les deux formulations PLNE les plus connues, puis nous démontrons leur équivalence. Nous présentons également la définition de bornes inférieures pour la période λ et pour l'objectif secondaire. La section 2.5 donne enfin des résultats expérimentaux sur les instances décrites dans le chapitre 1 section 1.3.4.

2.2 Programmation linéaire en nombres entiers

Un problème de programmation linéaire (PL) est un problème d'optimisation où la fonction objectif et les contraintes sont toutes linéaires. Un problème de programmation linéaire en nombres entiers (PLNE) est un PL dans lequel les variables doivent prendre des valeurs entières. Un programme linéaire en nombres entiers s'écrit de la façon suivante :

$$(IP) : z\ min = cx \tag{2.1}$$

sous :

$$Ax \leq b \tag{2.2}$$

$$x \in \mathbb{N}^n \tag{2.3}$$

avec $c \in \mathbb{Z}^n, b \in \mathbb{Z}^m$ et $A \in \mathbb{Z}^{mn}$.

Les éléments de l'ensemble $S = \{x \in \mathbb{N} \mid Ax \leq b\}$ sont les solutions réalisables du problème, c'est-à-dire, les valeurs de x qui satisfont toutes les contraintes du modèle. $conv(S)$ est l'enveloppe convexe de l'ensemble S. Les éléments de l'ensemble S qui minimisent la fonction objectif (2.1) sont les solutions optimales.

2.2.1 Relaxations et bornes

Un PLNE est caractérisé par le fait que les variables doivent prendre leurs valeurs dans un domaine discret. Il existe également des programmes linéaires où seulement une partie des variables sont entières et l'autre partie peut prendre des valeurs réelles, dans ce cas, on parle d'un programme linéaire mixte. Considérons le problème (IP) de la forme

$$z = min\{cx : Ax \leq b, x \in \mathbb{N}^n\} \tag{2.4}$$

Définition 3 *(Dominikus, 2007) Nous appelons $z_{rl} = min\{cx : Ax \leq b, x \in \mathbb{R}^n\}$ la relaxation PL de IP.*

Si $z_{rl} = z$, nous dirons que la relaxation LP est exacte. Soit $C = \{x \in \mathbb{R}^n : Ax \leq b\}$ et $P = C \cap \mathbb{Z}^n$. Nous appelons $C_z = conv(P) = conv(C \cap \mathbb{Z}^n)$ le polyèdre entier associé au polyèdre C. Alors le problème (IP) est équivalent au problème

$$z = min\{cx : Ax \leq b, x \in C_z\}, \tag{2.5}$$

qui est aussi un programme linéaire continu.

Définition 4 *(Dominikus, 2007) Soit $C^{'}$ un polyèdre convexe fermé tel que $P \subset C^{'}$ (et donc $C_z \subset C^{'}$). Alors le programme linéaire $z^{'}_{rl} = min\{cx : x \in C^{'}\}$ est appelé une relaxation de z, et $z^{'}_{rl} \leq z$ est appelée une borne inférieure ou borne duale. La borne inférieure est appelée exacte si $z^{'}_{rl} = z$. Si elle existe, la solution optimale de $z^{'}_{rl}$ est notée $x^{'}$.*

Définition 5 *(Dominikus, 2007) Etant donné un point $x \in P = C \cap \mathbb{Z}^n$, alors $z \leq cx$; On appelle la valeur cx une borne supérieure ou borne primale.*

Pour certains problèmes, la borne inférieure obtenue z_{rl} arrondie à la valeur entière supérieure peut être toujours inférieure à la valeur optimale. En plus, cette solution optimale fractionnaire peut être éloignée de la solution optimale entière. Par exemple, pour le programme linéaire entier suivant,

$$\begin{gathered} max\ z = 8x_1 + 10x_2 \\ s.c \\ 4x_1 + 6x_2 \leq 24 \\ 8x_1 + 3x_2 \leq 24 \\ x_1, x_2 \geq 0, x_1, x_2 \in \mathbb{Z} \end{gathered}$$

la solution, réprésentée graphiquement, est :

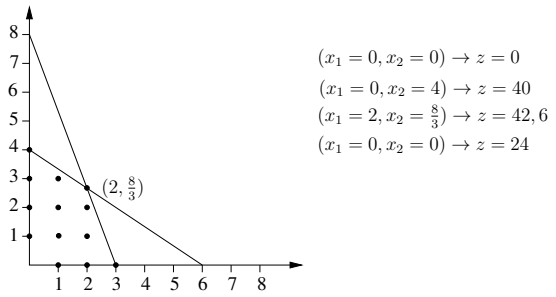


FIGURE 2.1 – Exemple de programmation linéaire entière et de relaxation.

La valeur maximale de la fonction objectif correspond au point $(x_1 = 2, x_2 = \frac{8}{3})$. Si nous arrondissons la solution, nous avons alors la solution $(x_1 = 2, x_2 = 3)$. Le point $(x_1 = 2, x_2 = 3)$ ne fait pas partie de l'ensemble des solutions réalisables.
La faiblesse de la relaxation continue est due au fait que la taille du polyèdre C est grande par rapport à $conv(S)$. Une façon d'améliorer la relaxation d'un programme linéaire entier est d'ajouter des inégalités valides qui coupent C. Pour plus de détails sur l'amélioration de la relaxation d'un programme linéaire entier, nous citons Gomory (1958); Chvátal (2006).

2.2.2 Résolution des programmes linéaires en nombres entiers

Lorsque la contrainte d'intégralité (2.3) sur les variables x est relâchée, l'ensemble S des solutions réalisables représente un polyèdre. Les contraintes sont représentées graphiquement par des demi-plans et leur intersection est un ensemble convexe ($conv(S)$). Les solutions, si elles existent, appartiennent à cet ensemble. L'idée est de chercher dans cet ensemble les valeurs des variables de décision qui minimisent ou maximisent la fonction objectif. La méthode du simplexe (Dantzig, 1951) part d'une solution réalisable à un sommet du polyèdre et cherche par itérations successives d'autres solutions correspondant à d'autres sommets qui peuvent améliorer la fonction objectif.
Différentes approches ont été étudiées pour la recherche de solutions optimales entières du PLNE. Actuellement les plus efficaces sont basées sur des algorithmes de recherche arborescente par séparation et évaluation (branch-and-bound), des méthodes de coupes et des relaxations.

Algorithmes de recherche arborescente par séparation et évaluation (branch-and-bound).

L'algorithme de recherche arborescente (Algorithme 1) énumère implicitement l'ensemble S des solutions du problème avec l'exploration progressive des différentes parties de l'espace de recherche. Le but est d'identifier et de supprimer les plus larges sous-espaces ne contenant pas de solution optimale (Demassey, 2003).

Un nœud est élagué si la relaxation continue n'a pas de solution ou si la valeur optimale de la relaxation continue est supérieure ou égale à z (la meilleure solution entière connue). La mise en place de l'algorithme nécessite spécifier deux aspect importants, le premier aspect définit la règle de sélection qui va permettre de choisir quel nœud non élagué doit être exploré. Le deuxième

aspect est la règle de branchement qui définit sur quelle variable brancher. Dans le cas général des variables entières, on choisit une variable de valeur fractionnaire dans la solution optimale de la relaxation continue et on branche sur l'arrondi supérieur et inférieur de cette valeur. Plus de détails sur l'ordre de sélection et les schémas de branchement sont disponibles dans Demassey (2003).

Algorithme 1: Algorithme de branch-and-bound (version de base)

1 Initialiser avec le calcul d'une solution admissible de valeur z ou poser $z = +\infty$
2 Résoudre la relaxation continue (z_{rl}) et mettre à jour éventuellement z
3 Appliquer un test d'élagage.
4 **si** *Il reste des nœuds non élagués* **alors**
5 | choisir un nœud non élagué et brancher sur une des variables.
6 | Pour chacun des 2 nouveaux nœuds :
7 | Résoudre la relaxation continue (z_{rl}) et mettre à jour z.
8 | Revenir à l'étape 3
9 **fin**
10 La solution courante z est optimale.

Un autre aspect important de l'algorithme de branch-and-bound est le problème de l'efficacité. Dans ce cas, c'est le nombre de nœuds explorés qui détermine le temps de calcul. A chaque nœud, la résolution d'un programme linéaire continue est effectuée.

Méthodes de coupes

L'objectif de cette méthode est de trouver le plus petit polyèdre contenant toutes les solutions admissibles entières du PLNE. L'usage le plus efficace de la méthode de coupes est la combinaison avec les méthodes de recherche arborescente. L'algorithme branch-and-cut (Algorithme 2) combine l'algorithme de branch-and-bound et la méthode des coupes polyédrales. Dans cet algorithme, à chaque itération de la recherche arborescente, l'évaluation par défaut d'un nœud est calculée par relaxation augmentée de coupes Demassey (2003). Dans un problème de maximisation entier, nous avons l'algorithme de branch-and-cut décrit par l'algorithme 2.

La programmation linéaire entière appartient à la classe de problèmes NP-complets. Aucun algorithme polynomial n'est donc connu pour sa résolution. Les algorithmes exponentiels comme le branch-and-bound sont donc susceptibles de rencontrer des difficultés si la taille des problèmes est importante. Notons que le solveur utilisé pour résoudre les PLNE étudiées dans cette thèse utilise l'algorithme du branch-and-cut.

Algorithme 2: Algorithme de branch-and-cut

1 Initialiser avec le calcul d'une solution admissible initiale $x^{'}, z^{'}$.
2 Résoudre la relaxation continue x_{rl}, z_{rl}.
3 Appliquer des algorithmes de coupes générales, par exemple Gomory ou un algorithme spécifique à le problème (amélioration de la borne supérieure z_{rl}).
4 Utiliser des heuristiques sur le problème primal (amélioration la borne inférieure z).
5 Revenir à l'étape 3 et 4 et répéter jusqu'à l'atteinte d'une condition particulière.
6 Démarrage du branch-and-bound.
7 Pour un nombre donné de nœuds, utiliser des algorithmes de coupes et des heuristiques primales.
8 Le processus se termine lorsque $z = z_{rl}$, et on a trouvé la meilleure solution réalisable x^*.

2.3 Problème d'ordonnancement de projet sous contraintes de ressources

Comme nous avons déjà mentionné, les formulations de PLNE rencontrées dans la littérature pour le RCMSP sont issues des formulations PLNE classique pour l'ordonnancement de projet sous contraintes de ressources (RCPSP). Nous présentons donc dans cette section le RCPSP et ses formulations les plus classiques.
Un projet est un ensemble d'activités ou tâches ayant des caractéristiques propres qui sont exécutées par un ensemble de ressources. Un problème d'ordonnancement de projet sous contraintes de ressources (RCPSP) consiste à trouver la date de début des activités (ou tâches) d'un projet de telle façon que les contraintes de précédence et les contraintes de ressources soient satisfaites à chaque instant t de l'ordonnancement. Le temps total d'achèvement est habituellement minimisé (Christofides *et al.*, 1987). Le problème d'ordonnancement de projet à contraintes de ressources s'exprime comme :

$$min\ Cmax \tag{2.6}$$

sous

$$Cmax \geq \sigma_i + p_i\ \forall i = 1, ..., n. \tag{2.7}$$

$$\sigma_j - \sigma_i \geq p_i,\ \forall (i,j) \in E. \tag{2.8}$$

$$\sum_{s(t)} b_i^s \leq B_s, s \in m. \tag{2.9}$$

où σ_i est la date de début de la tâche i, $\forall i = 1...n$. E représente l'ensemble de paires de tâches avec contraintes de précédence. p_i est la durée de la tâche i. b_i^s correspond à la quantité de ressource de type s nécessaire pour compléter la tâche i. $s(t)$ est l'ensemble d'activités en exécution à l'instant t. B_s représente la disponibilité de la ressource de type s (Christofides *et al.*, 1987).

De nombreux modèles de programmation linéaire en nombres entiers sont proposés dans la littérature pour la résolution du RCPSP. Nous présentons d'abord deux formulations de PLNE, qui sont la base des formulations PLNE d'ordonnancement modulo sous contraintes de ressources,

données par Eichenberger and Davidson (1997) et Dinechin (2004) qui vont être presentée dans la section (2.4).

Formulation de Pritsker.

Cette formulation a été proposée par Pritsker *et al.* (1969). La formulation contient un seul type de variables binaires de décision x_i^t indicées par le temps. $x_i^t = 1$ si l'activité i débute à l'instant t, sinon $x_i^t = 0$. La formulation est la suivante :

$$min \sum_{t=0}^{T} t x_{n+1}^t \tag{2.10}$$

s.c

$$\sum_{t=0}^{T} t x_j^t \geq \sum_{t=0}^{T} t x_i^t + p_i, \quad \forall (i,j) \in E. \tag{2.11}$$

$$\sum_{i=1}^{n} b_i^s \sum_{\tau=t-p_i+1}^{t} x_i^\tau \leq B_s, \quad \forall t \in T, \ \forall s \in m. \tag{2.12}$$

$$\sum_{t=0}^{T} x_i^t = 1, \quad \forall i = 1, ..., n. \tag{2.13}$$

$$x_i^t \in \{0,1\} \ \forall i = 1, ..., n, \forall t \in T \tag{2.14}$$

La fonction objectif consiste à minimiser le $Cmax$ car $\sum_{t=0}^{T} t x_{n+1}^t$ représente la minimisation de la date de début de la tâche d'achèvement de l'ordonnancement en T. La contrainte (2.11) représente la contrainte de précédence et indique que la date de début de l'activité ou tâche j doit être supérieure ou égale à la date de début de l'activité ou tâche i. La contrainte (2.12) assure que pour chaque ressource la somme des consommations des activités en cours d'exécution à chaque période de temps ne dépasse pas la disponibilité de la ressource. La contrainte (2.13) est appelée contrainte de non-préemption des activités, c'est-à-dire que chaque activité doit être exécutée une seule fois sur toute la durée du projet et qu'il est interdit d'interrompre une activité en cours d'exécution.
L'objectif de cette formulation est de diviser l'horizon du temps en intervalles constants dans lesquels il est possible de représenter tous les temps d'exécution des opérations.

Formulation de Christofides.

Cette formulation est semblable à la formulation de Pristker mais possède une plus forte relaxation linéaire. Elle contient des contraintes de précédence, de ressources et de non-préemption. La formulation est la suivante,

$$min \sum_{t=0}^{T} t x_{n+1}^t \tag{2.15}$$

S.C

$$\sum_{\tau=t}^{T} x_i^\tau + \sum_{\tau=0}^{t+p_i-1} x_j^\tau \leq 1 \quad \forall t \in T, \forall (i,j) \in E. \tag{2.16}$$

$$\sum_{i=1}^{n} b_i^s \sum_{\tau=t-p_i+1}^{t} x_i^\tau \leq B_s, \quad \forall t \in T, \ \forall s \in m. \tag{2.17}$$

$$\sum_{t=0}^{T} x_i^t = 1, \quad \forall i = 1, ..., n. \tag{2.18}$$

$$x_i^t \in \{0,1\}, \quad \forall i = 1, ..., n, \forall t \in T \tag{2.19}$$

Dans ce cas, la contrainte de précédence est établie pour chaque paire d'activités de l'ensemble E et pour chaque instant de l'horizon d'ordonnancement, contrairement à la formulation de Prisker où la contrainte de précédence ne prend pas en compte l'horizon de l'ordonnancement, mais uniquement les paires d'activités (i,j).

2.4 Programmation linéaire en nombres entiers pour l'ordonnancement modulo sous contraintes de ressources

Les formulations de programmation linéaire entière pour l'ordonnancement modulo sous contraintes de ressources apparaissent comme le résultat d'extensions des formulations de Pristker et Christofides et sont basées sur la résolution itérative de programmes linéaires avec une valeur de λ fixée. La résolution commence avec une méthode qui calcule un ordonnancement modulo réalisable à partir de $\lambda_{min} = \max(\lambda^{res}, \lambda^{prec})$ (voir chapitre 1, section 1.3.2). Une fois que λ_{min} est valide, un ordonnancement réalisable est trouvé, puis la minimisation de l'objectif secondaire est réalisée.

2.4.1 Formulation décomposée

Cette formulation proposée par Eichenberger and Davidson (1997), utilise la décomposition $\sigma_i = k_i\lambda + \tau_i$ où $k_i \in \mathbb{N}$ est le numéro du cycle dans lequel chaque opération est placée. τ_i est la date de début dans l'intervalle $[0, \lambda - 1]$. Cette formulation est basée sur les variables binaires y_i^τ tel que, $\tau_i = \sum_{\tau=0}^{\lambda-1} \tau y_i^\tau$, $i = 1, ..., n$. La formulation décomposée **{decomp}** s'exprime par :

$$min \sum_{i=1}^{n} w_i (\sum_{\tau=0}^{\lambda-1} \tau y_i^\tau + k_i\lambda) \tag{2.20}$$

$$\sum_{\tau=0}^{\lambda-1} y_i^\tau = 1, \forall i \in [1,n] \tag{2.21}$$

$$\sum_{\tau=0}^{\lambda-1} \tau y_i^\tau + k_i\lambda + \theta_i^j - \lambda\omega_i^j \leq \sum_{\tau=0}^{\lambda-1} \tau y_j^\tau + k_j\lambda, \forall (i,j) \in E \tag{2.22}$$

$$\sum_{i=1}^{n} y_i^\tau b_i^s \leq B_s, \forall s \in m, \forall \tau \in [0, \lambda - 1] \tag{2.23}$$

$$y_i^\tau \in \{1, 0\} \ \forall i \in [1, n], \ \forall \tau \in [0, \lambda - 1] \tag{2.24}$$

Cette formulation utilise $n \times \lambda$ variables binaires ainsi que n variables entières. Dans la figure (2.2) nous avons un exemple d'ordonnancement donné pour la formulation **{decomp}**. Par exemple, la date de début de la tâche générique 5 est donnée par : $\sigma_5 = \tau_5 + k_5\lambda$ avec $\sigma_5 = 2 + 0(4) = 2$.

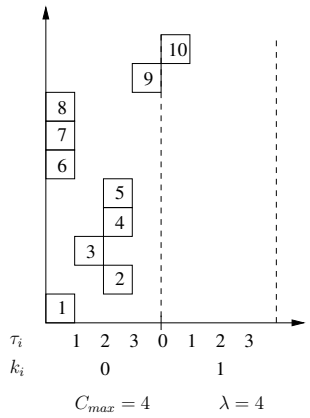


FIGURE 2.2 – Exemple d'ordonnancement avec la formulation décomposée.

La fonction objectif (2.20) et les contraintes de précédence (2.22) sont obtenues directement à partir de la la fonction objectif (1.10) et des contraintes de précédence (1.8) en remplaçant σ_i par $\sum_{\tau=0}^{\lambda-1} \tau y_i^\tau + k_i\lambda$. Si la fonction objectif est la minimisation de somme pondérée de dates de début et si nous considérons une tâche fictive $n+1$ qui représente la fin de l'ordonnancement dans le graphe G, de durée nulle ($p_{n+1} = 0$), on obtient la minimisation du makespan en posant $w_i = 0$, $\forall i \in [1, n]$ et $w_{n+1} = 1$. La contrainte (2.21), indique que chaque activité doit débuter exactement une fois dans l'intervalle d'initiation ou période λ. La contrainte (2.23) représente l'ensemble de contraintes de ressources. Elle exprime que la somme des consommations de ressources de type s pour chaque tâche générique i exécutée à l'instant $\tau = \sigma_i \mod \lambda$ ne peut pas dépasser la disponibilité de la ressource.

Eichenberger and Davidson (1997) proposent une nouvelle expression des contraintes de précédence à partir des résultats de Chaudhuri *et al.* (1994),

$$\sum_{x=\tau}^{\lambda-1} y_x^i + \sum_{x=0}^{(\tau+\theta_i^j-1) mod \lambda} y_x^j + k_i - k_j \leq \omega_i^j - \lfloor \frac{\tau + \theta_i^j - 1}{\lambda} \rfloor + 1, \forall \tau \in [0, \lambda - 1], \forall (i, j) \in E \tag{2.25}$$

Si la contrainte (2.22) est remplacée par la contrainte de précédence (2.25), une nouvelle formulation appelée *formulation structurée* **{decomp+}** est obtenue. Le mot *structurée* vient du fait que les coefficients dans l'expression (2.25) sont unitaires. Eichenberger and Davidson (1997) ont montré qu'avec la formulation **{decomp+}**, une formulation plus efficace d'ordonnancement

modulo est obtenue. Ceci conduit à une meilleure relaxation de la programmation linéaire car (2.25) et (2.21) $\Rightarrow$ (2.22). Pour la preuve, voir Eichenberger and Davidson (1997).

2.4.2 Formulation directe

Cette formulation proposée par Dinechin (2004), est basée sur les variables binaires x_i^t tel que $\sigma_i = \sum_{t=0}^{T-1} tx_i^t$, où T représente l'horizon de temps. Si nous exprimons $T = K\lambda$ alors $\sigma_i = \sum_{t=0}^{K\lambda} tx_i^t$. La formulation directe **{direct}** s'exprime par :

$$min \sum_{i=1}^{n} w_i(\sum_{t=0}^{K\lambda} tx_i^t) \tag{2.26}$$

$$\sum_{t=0}^{K\lambda} x_i^t = 1 \tag{2.27}$$

$$\sum_{t=0}^{K\lambda} tx_i^t + \theta_i^j - \lambda\omega_i^j \leq \sum_{t=0}^{K\lambda} tx_j^t, \forall (i,j) \in E \tag{2.28}$$

$$\sum_{i=1}^{n} \sum_{k=0}^{\lfloor \frac{K\lambda}{\lambda} \rfloor} x_i^{\tau+k\lambda} b_i^s \leq B_s, \forall \tau \in [0, \lambda-1], s \in [1, m) \tag{2.29}$$

$$x_i^t \in \{0,1\} \tag{2.30}$$

Cette formulation utilise $n \times T$ variables binaires. Dans la figure suivante nous avons un exemple d'ordonnancement en utilisant cette formulation.

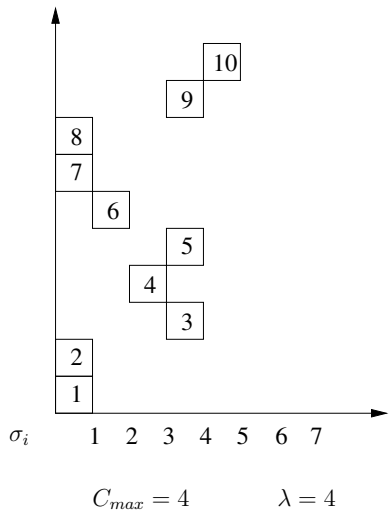


FIGURE 2.3 – Exemple d'ordonnancement avec la formulation directe.

Dans la figure 2.3, nous observons que la date de début σ_i est placée dans l'horizon T.

Comme dans la formulation **{decomp}**, la fonction objectif (2.26) et les contraintes de précédence (2.28) sont obtenues directement de la fonction objectif (1.10) et des contraintes de précédence (1.8) en remplaçant σ_i par $\sigma_i = \sum_{t=0}^{T-1} t x_i^t$. Nous pouvons considérer aussi une tâche fictive $n+1$, pour signaler la fin de l'ordonnancement dans le graphe E, (de durée nulle ($p_{n+1} = 0$)), et minimiser le makespan comme pour la formulation **{decomp}**. Même si cette formulation **{direct}** utilise aussi des variables binaires, la différence avec la formulation **{decomp}** est qu'elle ne présente pas de variables entières non binaires. De plus, bien que les contraintes d'affectation à une période, de précédence et de ressources restent identiques, la façon d'exprimer σ_i change. Dans ce cas, l'ordonnancement est fait sur l'horizon $T = K\lambda$, alors que dans la formulation **{decomp}** l'ordonnancement est fait dans la période λ. L'explication des contraintes (2.27) et (2.29) est analogue à celle donnée dans la formulation **{decomp}**.

Dupont de Dinechin propose une nouvelle contrainte de précédence donnée par l'expression suivante :

$$\sum_{h=t}^{K\lambda} x_i^h + \sum_{h=0}^{t+\theta_i^j-\lambda\omega_i^j-1} x_j^h \leq 1, \forall t \in [0, \ldots, K\lambda - 1], \forall (i,j) \in E \tag{2.31}$$

Si la contrainte (2.28) est remplacée par la contrainte de précédence (2.31), une nouvelle formulation appelée *formulation désagrégée* **{direct+}** est obtenue. Dinechin (2004) montre qu'avec la formulation **{direct+}**, une formulation plus efficace d'ordonnancement modulo est obtenue ce qui conduit à une meilleure relaxation de la programmation linéaire car (2.31) et (2.27) $\Rightarrow$ (2.28). Pour la preuve, nous renvoyons le lecteur à Dinechin (2004). La contrainte (2.31) corresponde à la contrainte (2.16) de la formulation de Christofides *et al.* (1987).

2.4.3 Comparaison des relaxations de programmation linéaire

Bien que les résultats expérimentaux présentés dans des études précédentes (de Dinechin *et al.*, 2010; de Dinechin, 2007) permettent de comparer les avantages des formulations décomposée, décomposée structurée, directe et directe désagrégée en termes de solutions entières qui peuvent être obtenues avec un logiciel commercial ou avec une heuristique basée sur la PLNE (de Dinechin, 2007; de Dinechin *et al.*, 2010), aucune étude n'a été réalisée jusqu'à présent pour comparer la qualité des relaxations des formulations PLNE.
Nous établissons deux résultats qui prouvent l'égalité entre les relaxations des formulations décomposée et directe, ainsi que l'égalité entre les relaxations des formulations décomposée structurée et directe désagrégée. Le théorème 1 montre que les relaxations des formulations **{direct}** et **{decomp}** sont équivalentes. Le théorème 2 montre que les relaxations des formulations renforcées **{direct+}** et **{decomp+}** sont aussi équivalentes.

Soit $T = K\lambda$, avec $T, K, \lambda \in \mathbb{N}^*$. Nous considérons les contraintes suivantes qui expriment les variables y et k de la formulation **{decomp}** comme une fonction linéaire des variables x de la formulation **{direct}**. Pour ceci nous utilisons l'expression $\sigma_i = \tau_i + k_i\lambda$.

$$y_i^\tau = \sum_{k=0}^{K-1} x_i^{\tau+k\lambda} \quad \forall i \in \{1, \ldots, n\}, \forall \tau \in \{0, \ldots, \lambda - 1\} \tag{2.32}$$

$$k_i = \sum_{k=1}^{K-1} \sum_{t=k\lambda}^{K\lambda-1} x_i^t \quad \forall i \in \{1, \ldots, n\} \tag{2.33}$$

Considérons les formulations PLNE **{E-direct}** et **{E-direct+}**, obtenues par l'insertion des contraintes (2.32) et (2.33) dans les formulations **{direct}** et **{direct+}** respectivement.

Compte tenu de la contrainte suivante pour les variables x_i^t

$$0 \leq x_i^t \leq 1 \quad \forall i \in \{1, \ldots, n\}, t \in \{0, \ldots, T\} \tag{2.34}$$

nous définissons aussi les formulations PLNE **{E-decomp}** and **{E-decomp+}**, obtenues par l'insertion des contraintes (2.32), (2.33) and (2.34) dans les formulations **{decomp}** and **{decomp+}**, respectivement.

Avec ces transformations, nous obtenons 4 formulations définies dans l'espace (x, y, k). Si (form) désigne une formulation PLNE, $\tilde{z}^*$(form) désigne la valeur optimale de l'objectif de sa relaxation linéaire.

Lemme 1 *En considérant les formulations **{direct}**, **{direct+}**, **{decomp}**, **{decomp+}** et leurs versions étendues, nous avons les équivalences suivantes :*

$$\tilde{z}^*(\textit{direct}) = \tilde{z}^*(\textit{E-direct}),$$

$$\tilde{z}^*(\textit{direct+}) = \tilde{z}^*(\textit{E-direct+}),$$

$$\tilde{z}^*(\textit{decomp}) = \tilde{z}^*(\textit{E-decomp}),$$

$$\tilde{z}^*(\textit{decomp+}) = \tilde{z}^*(\textit{E-decomp+}).$$

Preuve Dans le cas de la formulation **{direct}**, nous montrons que l'insertion de variables y_i et k_i ne change pas la valeur optimale de la relaxation et que ces variables ne sont soumises à aucune contrainte. Dans le cas de la formulation **{decomp}**, nous montrons que pour toute solution réalisable fractionnaire (y, k) qui respecte les contraintes de convexité sur la variable y, nous pouvons toujours trouver une solution fractionnaire x tel que $0 \leq x \leq 1$.

Le fait d'ajouter les variables y, k et les contraintes (2.32) et (2.33) aux formulations **{direct}** et **{direct+}** ne modifie pas l'espace de solution relaxé des formulations en termes des variables x car il n'y a pas d'autres contraintes sur les variables k et y. En conséquence, nous avons $\tilde{z}^*$(direct) $= \tilde{z}^*$(E-direct) et $\tilde{z}^*$(direct+) $= \tilde{z}^*$(E-direct+).

Pour les formulations **{decomp}** et **{decomp+}**, les formulation étendues **{E-decomp}** et **{E-decomp+}** sont obtenues avec la considération des variables x, $0 \leq x_i^t \leq 1$ (2.34) et des contraintes (2.32) et (2.33). Nous devons montrer que cette transformation ne modifie dans aucun cas la solution optimale relaxée. Pour cela, nous remarquons que, pour tout y et k tel que y vérifie (2.21), le système avec les inégalités (2.32), (2.33) a toujours une solution avec $0 \leq x_i^t \leq 1$.

Les variables x_i^τ apparaissent uniquement dans l'ensemble de contraintes (2.32), qui définissent les variables y_i^τ, nous pouvons donc récrire les contraintes (2.32) comme :

$$x_i^\tau = \sum_{k=1}^{K-1} x_i^{\tau+k\lambda} - y_i^\tau \quad \forall i \in \{1, \ldots, n\} \forall \tau \in \{0, \lambda - 1\}$$

Nous réécrivons $x_i^\tau \geq 0$ à partir de cette dernière expression

$$\sum_{k=1}^{K-1} x_i^{\tau+k\lambda} \geq y_i^\tau, \quad \forall i \in \{1,\ldots,n\} \forall \tau \in \{0, \lambda-1\} \tag{2.35}$$

Trouver une solution x réalisable consiste à trouver $0 \leq x_i^t \leq 1$, $t \geq \lambda$ qui vérifie (2.35) et (2.33). Cette solution réalisable peut être obtenue avec la procédure suivante.

Si k_i est un nombre entier, nous considérons $x_i^t = 0$ pour $t \in \{0,\ldots,k_i\lambda-1\} \cup \{(k_i+1)\lambda,\ldots,T-1\}$ et $x_i^{\tau+k_i\lambda} = y_i^\tau$ pour $\tau \in \{0,\ldots,\lambda-1\}$. Ceci vérifie les contraintes (2.35) et (2.33) puisque

$$k_i \sum_{\tau=0}^{\lambda-1} x_i^{\tau+k_i\lambda} = k_i \sum_{\tau=0}^{\lambda-1} y_i^\tau = k_i$$

car y vérifie (2.21).

Pour illustrer comment x est obtenu, nous supposons par exemple que $\lambda = 3$ et $K = 4$ ($T = 12$). Soit $y_{i0} = 0.2$, $y_{i1} = 0.3$, $y_{i2} = 0.5$ et $k_i = 2$. Un x non négatif qui vérifie (2.32) et (2.33) est obtenu simplement avec $x_i^t = 0$ pour $t = 0,1,2,3,4,5,9,10,11$ et $x_{i6} = 0.2$, $x_{i7} = 0.3$, $x_{i8} = 0.5$.

Si k_i n'est pas un nombre entier (nous avons $k_i < K-1$), la procédure décrite ci-dessus pour trouver un $\tilde{x}$ réalisable lorsque $\tilde{k}_i = \lceil k_i \rceil$ permet d'obtenir l'expression $\sum_{\tau=0}^{\lambda-1} \tilde{x}_i^{\tau+\lceil k_i \rceil\lambda} = \lceil k_i \rceil$.

De plus l'expression $y_i^\tau = \tilde{x}_i^\tau + \tilde{x}_i^{\tau+\lceil k_i \rceil\lambda}$ avec $\tilde{x}_i^\tau = 0$ pour toute $i \in \{0,\ldots,n\}, \tau \in \{0,\ldots,\lambda-1\}$ est obtenue à partir des contraintes (2.32). nous proposons de diminuer chaque $\tilde{x}_i^{\tau+\lceil \tilde{k}_i \rceil\lambda}$ par $\frac{(\lceil k_i \rceil - k_i)}{\lceil k_i \rceil} \tilde{y}_i^\tau$ et augmenter chaque terme $\tilde{x}_i^\tau$ par le même quantité de manière à atteindre x. On note que les égalités des contraintes (2.32) sont satisfaites. Avec cette opération nous obtenons $\lceil k_i \rceil \sum_{\tau=0}^{\lambda-1} x_i^{\tau+\lceil k_i \rceil\lambda} = k_i$ car l'expression $\lceil k_i \rceil \sum_{\tau=0}^{\lambda-1} \tilde{x}_i^{\tau+\lceil k_i \rceil\lambda}$ est reduite de $\sum_{\tau=0}^{\lambda-1} (\lceil k_i \rceil - k_i) y_i^\tau = (\lceil k_i \rceil - k_i)$.

Pour mieux comprendre cette transformation, considérons le même exemple avec $k_i = 2.4$. Nous obtenons $\tilde{x} \in [0,1]$ qui vérifie (2.32) et (2.33) pour $\tilde{k}_i = 3$ en fixant $\tilde{x}_{it} = 0$ pour $t = 0,1,2,3,4,5,6,7,8$ et $\tilde{x}_{i9} = 0.2$, $\tilde{x}_{i10} = 0.3$, $\tilde{x}_{i11} = 0.5$. Maintenant nous diminuons $\tilde{k}_i$ jusqu'à la valeur 2.4 en fixant $x_{i0} = \frac{(3-2.4)}{3} 0.2 = 0.04$, $x_{i1} = \frac{(3-2.4)}{3} 0.3 = 0.06$, $x_{i2} = \frac{(3-2.4)}{3} 0.5 = 0.1$, $x_{i9} = 0.2 - 0.04 = 0.16$, $x_{i10} = 0.3 - 0.06 = 0.24$ et $x_{i11} = 0.5 - 0.1 = 0.4$.

En conséquence, pour toute solution réalisable (y,k) qui vérifie les contraintes de convexité sur y, une solution x avec $0 \leq x \leq 1$ peut être trouvée. Donc $\tilde{z}^*(\text{decomp}) = \tilde{z}^*(\text{E-decomp})$ et $\tilde{z}^*(\text{decomp+}) = \tilde{z}^*(\text{E-decomp+})$. ∎

Théorème 1 *Soit $\tilde{z}^*(decomp)$ la valeur optimale pour la relaxation de la formulation* ***{decomp}*** *et $\tilde{z}^*(direct)$ la valeur optimale pour la relaxation de la formulation* ***{direct}****. Nous avons*

$$\tilde{z}^*(direct) = \tilde{z}^*(decomp)$$

Preuve Le lemme 1 nous permet de montrer cette équivalence à partir de la comparaison de la relaxation linéaire de la formulation directe étendue **{E-direct}** et la formulation décomposée étendue **{E-decomp}**. Nous montrons que toutes les contraintes de ces deux formulations sont équivalentes.

Selon le Lemme 1, nous avons $\tilde{z}^*(\text{decomp}) = \tilde{z}^*(\text{E-decomp})$ et $\tilde{z}^*(\text{direct}) = \tilde{z}^*(\text{E-direct})$. Les versions étendues des formulations peuvent être comparées pour obtenir le résultat souhaité. A

partir de cette idée, nous montrons que chaque solution réalisable de **{E-decomp}** est également une solution de **{E-direct}** et vice-versa. Ceci est démontré ci dessous.

- Equivalence des contraintes d'affectation (2.21) et (2.27) :
  Avec l'insertion de (2.32) dans la partie gauche de (2.21) nous obtenons :

$$\sum_{\tau=0}^{\lambda-1} \tilde{y}_i^\tau = \sum_{\tau=0}^{\lambda-1} \sum_{k=0}^{K-1} \tilde{x}_i^{\tau+k\lambda} = \sum_{t=0}^{T-1} \tilde{x}_i^t \tag{2.36}$$

  qui correspond précisément à le terme gauche de la contrainte d'affectation (2.27) de la formulation directe. Pour toute solution y qui satisfait (2.21) nous obtenons alors une solution x qui satisfait (2.27), et vice-versa.
- Equivalence des contraintes de précédence (2.22) et (2.28) :
  Avec l'insertion de (2.32) et (2.33) dans $\sum_{\tau=0}^{\lambda-1} \tau y_i^\tau + k_i\lambda$ nous obtenons

$$\begin{aligned}
\sum_{\tau=0}^{\lambda-1} \tau\tilde{y}_i^\tau + \tilde{k}_i\lambda &= \sum_{\tau=0}^{\lambda-1} \tau \sum_{k=0}^{K-1} \tilde{x}_i^{\tau+k\lambda} + \sum_{k=1}^{K-1} \sum_{t=k\lambda}^{K\lambda-1} \lambda\tilde{x}_i^t \\
&= \sum_{k=0}^{K-1} \sum_{\tau=0}^{\lambda-1} \tau\tilde{x}_i^{\tau+k\lambda} + \sum_{k=1}^{K-1} k \sum_{t=k\lambda}^{(k+1)\lambda-1} \lambda\tilde{x}_i^t \\
&= \sum_{k=0}^{K-1} \sum_{\tau=0}^{\lambda-1} \tau\tilde{x}_i^{\tau+k\lambda} + \sum_{k=0}^{K-1} \sum_{\tau=0}^{\lambda-1} k\lambda\tilde{x}_i^{\tau+k\lambda} \\
&= \sum_{t=0}^{T-1} t x_i^t
\end{aligned} \tag{2.37}$$

  A partir de cette équivalence, toute solution x qui vérifie les contraintes de précédence (2.28) correspond à une solution y qui satisfait les contraintes de précédence (2.22). La réciproque est également satisfaite.
- Equivalence des contraintes de ressources (2.23) et (2.29) :
  En ajoutant (2.32) dans le terme de gauche de (2.23) nous obtenons :

$$\sum_{i=1}^{n} \tilde{y}_i^\tau b_i^s = \sum_{i=1}^{n} \sum_{k=0}^{K-1} \tilde{x}_i^{\tau+k\lambda} b_i^s \tag{2.38}$$

  qui correspond précisément au terme de gauche des contraintes de ressources de la formulation directe (2.29).
- Bornes sur les variables y et k :
  Les contraintes (2.32) et (2.27) assurent que $0 \leq \tilde{y}_i^\tau \leq 1$ pour toute $i \in \{1, \ldots, n\}$, $\tau \in \{0, \ldots, \lambda - 1\}$. De la même manière, les contraintes (2.33) et (2.27) impliquent que $0 \leq \tilde{k}_i \leq K - 1$, pour tout $i \in \{1, \ldots, n\}$
- Equivalence de la fonction objectif :
  A partir de l'egalité (2.37) on peut facilement déduire que la valeur de la fonction objectif de la relaxation de la formulation **{E-direct}** et de la relaxation de la formulation **{E-decomp}** sont équivalentes.

Les équivalences démontrées ci-dessus prouvent que la relaxation des formulations **{E-decomp}**

et **{E-direct}** sont égales. Ce résultat peut être étendu aux relaxations des formulations **{decomp}** et **{direct}** à partir du lemme 1 ∎

Finalement, nous démontrons l'équivalence de la relaxation des formulations **{direct+}** et **{decomp+}**.

Théorème 2 *Soit $\tilde{z}^*(decomp+)$ la valeur optimale pour la relaxation de la formulation décomposée renforcée et $\tilde{z}^*(direct+)$ la valeur optimale pour la relaxation de la formulation directe renforcée, alors*

$$\tilde{z}^*(direct+) = \tilde{z}^*(decomp+)$$

Preuve Nous nous basons aussi sur le Lemme 1 pour démontrer cette équivalence, en comparant la relaxation de la formulation renforcée directe **{direct+}** et la formulation renforcée décomposée étendue **{E-decomp+}**. La seule différence avec la démonstration antérieure réside dans les contraintes de précédence. Nous montrerons que la solution optimale des deux relaxations sans la considération des contraintes de ressources est toujours $0-1$ dans les deux cas.

Comme pour le théoreme 1, nous utilisons le lemme 1 qui énonce que $\tilde{z}(\text{direct}+) = \tilde{z}(\text{E-direct}+)$ et $\tilde{z}(\text{decomp}+) = \tilde{z}(\text{E-decomp}+)$. En conséquence, il suffit de comparer la formulation **{direct+}** et la formulation **{E-decomp+}**.

A partir des égalités (2.36) et (2.38) et avec le remplacement des variables y et k par leurs expressions en fonction de la variable x, nous obtenons deux formulations dont la différence réside dans les contraintes de précédence.

Si on ignore les contraintes de ressources, on peut établit que toute solution de base réalisable liée à la matrice des contraintes des formulations est $0-1$.

D'abord, on considère la formulation renforcée décomposée **{decomp+}**. La contrainte de précédence structurée est réécrite en remplaçant les variables y_i et k_i par leurs expressions en fonction de la variable x_i^t (2.32 et 2.33). Nous considérons l'expression $\sum_{s=\tau}^{\lambda-1} y_i^s + k_i$. Nous avons donc

$$\begin{aligned}
\sum_{s=\tau}^{\lambda-1} y_i^s + k_i &= \sum_{k=0}^{K-1}\sum_{s=\tau}^{\lambda-1} x_i^{s+k\lambda} + \sum_{k=1}^{K-1}\sum_{s=k\lambda}^{K\lambda-1} x_i^s \\
&= \sum_{k=0}^{K-1}\sum_{s=\tau+k\lambda}^{(k+1)\lambda-1} x_i^s + \sum_{k=0}^{K-1}\sum_{s=(k+1)\lambda}^{K\lambda-1} x_i^s \\
&= \sum_{k=0}^{K-1}\sum_{s=\tau+k\lambda}^{K\lambda-1} x_i^s
\end{aligned}$$

Si on utilise, dans la contrainte de précédence structurée (2.25), les transformations (2.32) et (2.33) et le résultat $\sum_{s=\tau}^{\lambda-1} y_i^s + k_i = \sum_{k=0}^{K-1}\sum_{s=\tau+k\lambda}^{K\lambda-1} x_i^s$, cette contrainte peut alors être réécrite comme :

$$\sum_{k=0}^{K-1}\sum_{s=\tau+k\lambda}^{K\lambda-1} x_i^s + \sum_{s=0}^{(\tau+\theta_i^j-1)mod\lambda}\sum_{k=0}^{K-1} x_j^{s+k\lambda} - \sum_{k=1}^{K-1}\sum_{t=k\lambda}^{K\lambda-1} x_j^t \leq \omega_i^j - \lfloor\frac{\tau+\theta_i^j-1}{\lambda}\rfloor + 1$$

$$\sum_{k=0}^{K-1}\sum_{s=\tau+k\lambda}^{K\lambda-1} x_i^s + \sum_{s=0}^{(\tau+\theta_i^j-1)mod\lambda}\sum_{k=0}^{K-1} x_j^{s+k\lambda} - \sum_{k=0}^{K-1}\sum_{t=k\lambda}^{K\lambda-1} x_j^t + 1 \leq \omega_i^j - \lfloor\frac{\tau+\theta_i^j-1}{\lambda}\rfloor + 1$$

$$\sum_{k=0}^{K-1}\sum_{s=\tau+k\lambda}^{K\lambda-1} x_i^s + \sum_{s=0}^{(\tau+\theta_i^j-1)mod\lambda}\sum_{k=0}^{K-1} x_j^{s+k\lambda} + \sum_{k=0}^{K-1}\sum_{t=0}^{k\lambda-1} x_j^t - K + 1 \leq \omega_i^j - \lfloor\frac{\tau+\theta_i^j-1}{\lambda}\rfloor + 1$$

$$\sum_{k=0}^{K-1}\sum_{s=\tau+k\lambda}^{K\lambda-1} x_i^s + \sum_{k=0}^{K-1}(\sum_{s=0}^{(\tau+\theta_i^j-1)mod\lambda} x_j^{s+k\lambda} + \sum_{t=0}^{k\lambda-1} x_j^t) - K + 1 \leq \omega_i^j - \lfloor\frac{\tau+\theta_i^j-1}{\lambda}\rfloor + 1$$

$$\sum_{k=0}^{K-1}\sum_{s=\tau+k\lambda}^{K\lambda-1} x_i^s + \sum_{k=0}^{K-1}(\sum_{s=0}^{k\lambda+(\tau+\theta_i^j-1)mod\lambda} x_j^s) - K + 1 \leq \omega_i^j - \lfloor\frac{\tau+\theta_i^j-1}{\lambda}\rfloor + 1$$

$$-\sum_{k=0}^{K-1}\sum_{s=0}^{s=\tau+k\lambda-1} x_i^s + \sum_{k=0}^{K-1}\sum_{s=0}^{k\lambda+(\tau+\theta_i^j-1)mod\lambda} x_j^s \leq \omega_i^j - \lfloor\frac{\tau+\theta_i^j-1}{\lambda}\rfloor$$

Maintenant, les variables x_i^s sont remplacés par les variables $z_i^{\tau,k,p}$ que nous définissons par

$$z_i^{\tau,k,p} = \sum_{q=k}^{p}\sum_{s=0}^{\tau+q\lambda} x_i^s$$

La contrainte de précédence structurée peut être alors écrite comme :

$$-z_i^{\tau-1,0,K-1} + z_j^{(\tau+\theta_i^j-1)mod\lambda,0,K-1} \leq \omega_i^j - \lfloor\frac{\tau+\theta_i^j-1}{\lambda}\rfloor \quad \forall(i,j)\in E, \forall\tau\in\{0,\ldots,\lambda-1\} \quad (2.39)$$

La contrainte (2.21) devient

$$z_i^{\lambda-1,0,K-1} - z_i^{-1,0,K-1} = 1.\forall i \in \{1,...,n\} \quad (2.40)$$

En ajoutant les contraintes sur $x_i^{\tau+k\lambda}$, nous avons

$$0 \leq z_i^{\tau,k,k} - z_i^{\tau-1,k,k} \leq 1.\forall i\in\{1,...,n\}, \forall\tau\in\{0,\ldots,\lambda\}, \forall k\in\{0,\ldots,K-1\} \quad (2.41)$$

La matrice des contraintes Z correspondant aux contraintes (2.39), (2.40) et (2.41) a une solution avec des coordonnées entières $z_i^{\tau,k,p}$ si et seulement si Z est une matrice totalement unimodulaire, c'est-à-dire, si les déterminants des sous-matrices de Z sont égaux à 0, 1 ou -1. Une condition suffisante mais pas nécessaire pour qu'une matrice A avec $a_{i,j} \in \{0,-1,1\}$ soit totalement unimodulaire (Papadimitriou and Steiglitz, 1982) consiste à vérifier que chaque colonne de A contient au plus 2 éléments non nuls et que les lignes de A peuvent être partitionnées

en 2 ensembles disjoints I_1 et I_2 avec les propriétés suivantes :

1. Si 2 éléments d'une colonne de A ont le même signe, la ligne de l'un est alors dans I_1 et l'autre ligne dans I_2.
2. Si 2 éléments d'une colonne de A ont des signes opposés, alors les lignes des 2 éléments sont dans I_1 ou dans I_2.

En conséquence la matrice des contraintes Z est totalement unimodulaire.

Nous considérons maintenant la formulation **{direct+}** sans les contraintes de ressources. Pour démontrer l'unimodularité nous proposons une autre transformation non-singulière des variables x_i^t en les variables h_i^t définies par :

$$h_i^t = \sum_{s=0}^{T-1} x_i^s$$

Les contraintes d'affectation, les contraintes de précédence et les contraintes sur les variables x_i^t peuvent être écrites comme :

$$h_i^t = 1, \forall i \in \{1, \ldots, n\}$$

$$h_i^t - h_j^{t+\theta_i^j - \lambda\omega_i^j - 1} \geq 0, \forall (i,j) \in E, \forall t \in \{0, \ldots, T-1\}.$$

$$0 \leq h_i^t - h_i^{t-1} \leq 1 \quad \forall i \in \{1, \ldots, n\}, \forall t \in \{0, \ldots, T-1\}$$

et

$$h_i^t \geq 0$$

L'unimodularité totale de la matrice H est alors démontrée.

Les formulations directe et décomposée ont donc des solutions $0-1$ par rapport aux variables x lorsqu'on considère uniquement les contraintes de précédence. Par ailleurs, les deux formulations ont les mêmes contraintes de ressource. En conséquence leurs relaxations sont équivalentes. ∎

2.4.4 Calcul de bornes inférieures et résolution exacte

Dans cette section, nous montrons comment utiliser les formulations pour trouver des bornes inférieures et des solutions optimales ou réalisables pour la période λ et pour l'objectif secondaire (minimisation du makespan ou minimisation de la somme pondérée des dates de début des tâches).

Borne inférieure pour la période λ.

La recherche d'une borne inférieure pour la période λ commence avec une méthode qui résout le programme linéaire à partir d'une valeur de $\lambda = \lambda_{min}$ fixée, tel que, $\lambda_{min} = \max(\lambda^{res}, \lambda^{prec})$ (voir chapitre 1, section 1.3.2) et des contraintes d'intégralité relâchées. Si avec cette valeur de λ, il n'existe pas une solution pour le programme linéaire, nous incrémentons la valeur de λ ($\lambda = \lambda + 1$).

Une borne inférieure λ_{RLP} pour la période λ est atteinte lorsque le programme linéaire admet une solution. Dans notre cas, le programme linéaire correspond aux formulations présentées dans les sections 2.4.1 et section 2.4.2.

Borne inférieure et solution optimale pour le makespan ($Cmax$).

Le makespan est défini comme la différence entre la date de début et la date de fin de l'ordonnancement. Le makespan doit toujours être supérieur ou égal à la période λ. Pour trouver une borne inférieure pour le makespan, nous proposons de résoudre la relaxation des contraintes d'intégralité des formulations présentées dans les sections 2.4.1 et 2.4.2 avec une certaine valeur de λ. La résolution de ces relaxations peut donner une valeur pour le makespan plus petite que la valeur de λ, nous proposons alors une borne inférieure pour le makespan définie par $Cmax_{min} = \max(\lambda, Cmax)$, où $Cmax$ correspond à la valeur obtenue avec la résolution de la relaxation. Par exemple, la matrice suivante montre la solution de la relaxation de la formulation de Dinechin (2004) pour une instance particulière. Chaque ligne correspond à une tâche (on a ici 14 tâches). Chaque colonne représente l'instant où chaque tache est exécutée. Les éléments x_i^t de la matrice sont donc les valeurs de la solution pour la relaxation.

$$X_i^t = \begin{pmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 00 \\
0 & 0,7167 & 0,14 & 0 & 0 & 0 & 0 & 0,1433 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0,36 & 0 & 0 & 0 & 0,2 & 0,44 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0,5 & 0 & 0 & 0 & 0,5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0,2 & 0 & 0 & 0 & 0 & 0,8 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0,33 & 0 & 0 & 0,67 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0,0833 & 0 & 0 & 0 & 0,75 & ,0,167 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0,1667 & 0 & 0,5 & 0,333 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0,5 & 0,25 & 0 & 0,25 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0,7143 & 0,2857 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0,6429 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0,3571 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0,6429 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0,3571 \\
0,5714 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0,4285
\end{pmatrix}$$

Dans cet exemple, la valeur du $Cmax = 6$ alors que la valeur de la période est $\lambda = 8$.

Solutions optimales

. Pour la recherche de solutions optimales la méthode consiste de résoudre pour chaque valeur de λ le PLNE au lieu de sa relaxation. Une fois que λ est réalisable, la minimisation du $Cmax$ est effectuée.

2.5 Résultats expérimentaux

Les expérimentations sur les formulations PLNE **{decomp(+)}** et **{direct(+)}** ont été exécutées avec Cplex V. 6.0 sur un PC équipé du processeur Intel(R) Core(TM)2 Duo CPU E4400 @ 2.00GHz 1.99 GHz RAM. Nous avons implémentée en OPL Les Formulations PLNE décomposée, décomposée structurée, directe et directe désagregée.

L'objectif est d'évaluer la performance des formulations **{decomp(+)}** et **{direct(+)}**. Nous comparons tout d'abord ces formulations en termes de solutions optimales ou réalisables trouvées.

Les tableaux 2.1, 2.2, 2.3 et 2.4 montrent les temps de résolution pour les instances avec les formulations **{decomp(+)}** et **{direct(+)}**. Les instances ont été groupées en fonction de leur taille (nombre d'opérations à ordonnancer) et du temps de calcul nécessaire pour leur résolution. En ce qui concerne le temps de résolution, il semble que la formulation **{decomp(+)}** est plus rapide que la formulation **{direct(+)}**. En effet, la formulation **{direct(+)}** a un plus grande nombre de variables ($n \times T$) que la formulation **{decomp(+)}** ($(n \times \lambda) + n$ variables entières). Nous observons que généralement plus taille des instances est importante et plus le temps de calcul nécessaire pour leur résolution est également important comme on pouvait s'y attendre.

♯operat	♯instances	*ms*	*sec*	*min*	$> 1h$	No [1]
0 - 30	16	14	2	-	-	-
31 - 60	15	7	3	4	1	-
61 - 100	2	-	-	-	2	-
> 100	3	-	-	-	2	1
Total	36	21	5	4	5	1

TABLE 2.1 – Formulation **{decomp+}** instances industrielles

♯operat	♯instances	*ms*	*sec*	*min*	$> 1h$	No [1]
0 - 30	16	14	2	-	-	-
31 - 60	15	7	2	5	1	-
61 - 100	2	-	-	-	2	-
> 100	3	-	-	-	2	1
Total	36	21	4	5	5	1

TABLE 2.2 – Formulation **{direct+}** instances industrielles

♯operat	♯instances	*ms*	*sec*	*min*	$> 1h$	No [1]
0 - 30	16	3	9	4	-	-
31 - 60	15	1	2	9	2 [2]	1
61 - 100	2	-	-	-	-	2
> 100	3	-	-	-	-	3
Total	36	4	11	13	2	6

TABLE 2.3 – Formulation **{decomp+}** instances modifiées

♯operat	♯instances	ms	sec	min	$> 1h$	No [1]
0 - 30	16	3	8	5	-	-
31 - 60	15	-	3	9	-	3
61 - 100	2	-	-	-	-	2
> 100	3	-	-	-	-	3
Total	36	3	11	14	0	8

TABLE 2.4 – Formulation **{direct+}** instances modifiées

Les tableaux 2.5 et 2.6 montrent en détail les solutions optimales ou réalisables obtenues pour chaque instance. Les tables 2.7, 2.8 affichent les résultats sur les bornes inférieures de la période et du makespan sur ces mêmes instances.

Nous remarquons que la borne inferieure λ_{RLP} obtenue avec la relaxation des formulations est toujours égale à la borne triviale λ_{min} pour l'ensemble des instances industrielles et modifiées. Cela peut être dû au fait que la borne triviale λ_{min} est toujours réalisable pour ces instances ou, au contraire, que les relaxations ne sont pas suffisamment efficaces.

La table 2.5 montre les résultats de la résolution exacte pour les instances industrielles. Dans ce cas, toutes les instances ont été résolues jusqu'à l'optimalité sauf l'instance (*gsm-st231.18*) (qui n'a pas pu être résolue même après trois semaines de calcul). Nous remarquons que le temps moyens d'exécution pour la formulation **{decomp(+)}** est 6% inférieure que le temps moyens d'exécution pour la formulation **{direct(+)}**. Bien que pour quelques instances, le temps de calcul est vraiment petit (millièmes de secondes), d'autres instances demandent cependant un temps plus conséquent. Par exemple les instances (*adpcm-st231.1, gsm-st231.*) et notamment l'instance (*adpcm-st231.2*) qui nécessite 161 heures pour être résolue. 21 instances ont trouvée la solution optimale à la racine, ce qui correspondre au temps d'exécution de 0.05 seg.

L'étude des solutions optimales sur la table 2.5 montre que pour l'ensemble d'instances industrielles la solution optimale pour la période est (quasiment) toujours égale à la borne théorique triviale λ_{min}. En conséquence, l'ensemble d'instances industrielles ne peuvent pas être utilisé pour évaluer la performance des relaxations des formulations **{direct(+)}** et **{decomp(+)}** en ce qui concerne la période. Ceci n'est pas le cas pour le makespan, les relaxations des formulations **{direct(+)}** et **{decomp(+)}** permettant d'améliorer la borne.

La table 2.6 montre les résultats de la résolution exacte pour les instances modifiées. Dans ce groupe d'instances, la consommation des ressources de type non-unitaire augmente la difficulté. Par exemple, nous observons que 6 instances n'ont pas été résolues après trois semaines de calcul. Pour deux des instances (*gsm-st231.25*) et (*gsm-st231.25-st231.33*) des solutions réalisables approchées mais non-optimales avec des écarts respectifs de $1,75\%$ et 8% ont été trouvées. Nous remarquons que le temps moyens d'exécution pour la formulation **{decomp(+)}** est 28% inférieure que le temps moyens d'exécution pour la formulation **{direct(+)}**. La diffi-

1. Non-resolues.
2. Solution réalisable qui n'est pas optimale.

culté de l'ensemble d'instances modifiées par rapport à l'ensemble d'instances industrielles peut s'expliquer par le fait de que les périodes optimales sont ici plus grandes que la borne triviale $\lambda_{min} = \max(\lambda^{prec}, \lambda^{res})$.

La table2.7 montre les résultats des relaxations des formulations PLNE décomposée structurée et directe désagréegée pour les instances insdustrielles. Dans ce cas, nous ne pouvons pas améliorer la borne obtenue à partir de la valeur $\lambda_{min} = \max(\lambda^{prec}, \lambda^{res})$ car déjà cette borne est égale à la valeur optimale de la période λ. Nos observons que les résultats de la borne pour le $Cmax$ restent plus intéressantes.

La table2.8 montre les résultats des formulations PLNE décomposée structurée et directe désagréegée pour les instances modifiées. Dans ce cas, la valeur de la borne pour la période λ obtenue à partir des relaxations des formulations PLNE décomposée structurée et directe désagréegée sont toujours égaux à les résultats obtenus à partir de $\lambda_{min} = \max(\lambda^{prec}, \lambda^{res})$. Nous pouvons donc conclure que les relaxations des formulations PLNE pour le problème d'ordonnancement modulo sous contraintes de ressources sont malheureusement non efficaces pour le calcul d'une borne pour la période λ. Ces relaxations restent cependant utiles pour le calcul d'une borne pour le makespan.

Instances	λ_{min}	PLNE(decomp+)			PLNE(direct+)			écart (CPU_s%)
		λ	$Cmax$	CPU_s	λ	$Cmax$	CPU_s	
adpcm-st231.1	21	21	30	14400	21	30	16235	11.30%
adpcm-st231.2	38	40	42	582362	40	42	601000	3.10%
gsm-st231.1	24	24	33	0.05	24	33	0.05	0%
gsm-st231.2	26	26	32	79362	26	32	83991	5.51%
gsm-st231.5	11	11	17	0.05	11	17	0.05	0%
gsm-st231.6	7	7	13	17	7	13	20	15%
gsm-st231.7	11	11	17	0.05	11	17	0.05	0%
gsm-st231.8	8	8	8	0.05	8	8	0.05	0%
gsm-st231.9	28	28	28	0.05	28	28	0.05	0%
gsm-st231.10	4	4	4	0.05	4	4	0.05	0%
gsm-st231.11	20	20	21	0.05	20	21	0.05	0%
gsm-st231.12	8	8	8	0.05	8	8	0.05	0%
gsm-st231.13	19	19	25	1856	19	25	2023	8.25%
gsm-st231.14	10	10	13	301.25	10	13	478	36.97%
gsm-st231.15	8	8	8	0.05	8	8	0.05	0%
gsm-st231.16	16	16	20	7520	16	20	8156	7.80%
gsm-st231.17	9	9	16	0.05	9	16	0.05	%
gsm-st231.18	53	-	-	-	-	-	-	-
gsm-st231.19	8	8	9	0.05	8	9	0.05	0%
gsm-st231.20	6	6	10	0.05	6	10	0.05	0%
gsm-st231.21	18	18	22	0.05	18	22	0.05	0%
gsm-st231.22	18	18	22	0.05	18	22	0.05	0%
gsm-st231.25	16	16	25	3652	16	25	4001	8.72%
gsm-st231.29	11	11	17	12.6	11	17	15	16%
gsm-st231.30	7	7	13	12	7	13	15	20%
gsm-st231.31	11	11	17	47	11	17	73	35.61%
gsm-st231.32	15	15	15	0.05	15	15	0.05	0%
gsm-st231.33	15	15	21	2365	15	21	2503	5.51%
gsm-st231.34	4	4	5	0.05	4	5	0.05	0%
gsm-st231.35	6	6	11	0.05	6	11	0.05	0%
gsm-st231.36	10	10	15	27	10	15	42	35.71%
gsm-st231.39	8	8	16	0.05	8	16	0.05	%0%
gsm-st231.40	10	10	10	0.05	10	10	0.05	0%
gsm-st231.41	18	18	24	2356	18	24	2562	8.04%
gsm-st231.42	6	6	10	0.05	6	10	0.05	0%
gsm-st231.43	8	8	14	0.05	8	14	0.05	0%
Valeur moyenne	13.22	13.28	18	19836	13.28	18	20603	6.22%

TABLE 2.5 – Solutions optimales ou réalisables pour les instances industrielles

Instances	λ_{min}	PLNE(decomp+)			PLNE(direct+)			écart (CPU_s%)
		λ	$Cmax$	CPU_s	λ	$Cmax$	CPU_s	
adpcm-st231.1	52	-	-	-	-	-	-	-
adpcm-st231.2	82	-	-	-	-	-	-	-
gsm-st231.1	24	25	42	250	25	42	375	33.33%
gsm-st231.2	59	-	-	-	-	-	-	-
gsm-st231.5	26	36	46	280	36	46	299.03	6.36%
gsm-st231.6	17	27	27	152	27	27	265	43%
gsm-st231.7	28	41	45	92	41	45	115	20%
gsm-st231.8	9	12	12	0.27	12	12	0.31	12.90%
gsm-st231.9	28	32	35	56	32	35	60	6.66%
gsm-st231.10	6	8	8	0.10	8	8	0.11	9.09%
gsm-st231.11	20	24	24	0.37	24	24	0.39	5.12%
gsm-st231.12	10	13	13	12.65	13	13	19	33.42%
gsm-st231.13	27	43	48	985.03	43	48	1236	20.35%
gsm-st231.14	20	33	45	220	33	45	252	13%
gsm-st231.15	9	12	12	12.36	12	12	13	5%
gsm-st231.16	38	-	-	-	-	-	-	-
gsm-st231.17	23	33	33	90	33	33	105	14%
gsm-st231.18	120	-	-	-	-	-	-	-
gsm-st231.19	12	15	15	38.23	15	15	43	11.09%
gsm-st231.20	13	20	27	123	20	27	137	10.21%
gsm-st231.21	20	30	30	42.03	30	30	59	29%
gsm-st231.22	18	29	29	80.36	29	29	112	28%
gsm-st231.25	37	56 (Gap=1.75%)	56	604800	-	-	-	-
gsm-st231.29	28	42	42	210	42	42	513	59%
gsm-st231.30	16	25	25	58	25	25	67	13.43%
gsm-st231.31	26	39	39	142	39	39	169	16%
gsm-st231.32	21	30	30	0.25	30	30	1.01	75%
gsm-st231.33	33	52(Gap=8%)	50	604800	-	-	-	-
gsm-st231.34	7	7	7	5.05	7	7	8	30%
gsm-st231.35	12	14	16	52	14	16	53	1.88%
gsm-st231.36	18	24	28	230	24	24	403	42%
gsm-st231.39	15	21	25	95	21	25	168	44%
gsm-st231.40	12	17	21	15	17	21	29	48%
gsm-st231.41	33	-	-	-	-	-	-	-
gsm-st231.42	14	18	26	12	18	26	17	29%
gsm-st231.43	15	20	25	15	20	25	23	34%
Valeur moyenne	19	27	30	116	27	30	162	28%

TABLE 2.6 – Solutions optimales ou réalisables pour les instances modifiées

Instances	λ_{min}	λ	$C_{\max}$	CPU (s) (decomp+)	CPU (s) (direct+)
adpcm-st231.1	21	21	29.5	498	536
adpcm-st231.2	38	38	42	5326.37	6012
gsm-st231.1	24	24	33	0.02	0.02
gsm-st231.2	26	26	31.5	586	614
gsm-st231.5	11	11	15.2	0.02	0.02
gsm-st231.6	7	7	11.2	0.02	0.02
gsm-st231.7	11	11	15.2	0.02	0.02
gsm-st231.8	8	8	8	0.02	0.02
gsm-st231.9	28	28	28	0.02	0.02
gsm-st231.10	4	4	4	0.00	0.00
gsm-st231.11	20	20	21	0.02	0.02
gsm-st231.12	8	8	8	0.00	0.00
gsm-st231.13	19	19	25	0.02	0.02
gsm-st231.14	10	10	12.63	0.02	0.02
gsm-st231.15	8	8	8	0.00	0.00
gsm-st231.16	16	16	16.97	3625.12	3812.03
gsm-st231.17	9	9	15.25	0.02	0.02
gsm-st231.18	53	53	53	7256	8002.03
gsm-st231.19	8	8	8	0.00	0.00
gsm-st231.20	6	6	10	0.00	0.00
gsm-st231.21	18	18	22	15	15
gsm-st231.22	18	18	21	17	20
gsm-st231.25	16	16	24.52	789.26	814
gsm-st231.29	11	11	15.2	7.52	7.84
gsm-st231.30	7	7	11.2	0.02	0.02
gsm-st231.31	11	11	15.2	0.02	0.02
gsm-st231.32	15	15	15	4.25	4.45
gsm-st231.33	15	15	17.23	42	42
gsm-st231.34	4	4	5	0.00	0.00
gsm-st231.35	6	6	8.2	0.00	0.00
gsm-st231.36	10	10	10	0.02	0.02
gsm-st231.39	8	8	12.5	0.02	0.02
gsm-st231.40	10	10	10	0.00	0.00
gsm-st231.41	18	18	18	12	0.02
gsm-st231.42	6	6	10	0.02	0.02
gsm-st231.43	8	8	10.4	0.00	0.00

TABLE 2.7 – Résultats des bornes inférieures obtenues à partir de la relaxation des formulations **{direct+}** et **{decomp+}** pour les instances industrielles

Instances	λ_{min}	λ	$Cmax$	CPU (s) (decomp+)	CPU (s) (direct+)	$(\lambda_{opt} - \lambda)\%$
adpcm-st231.1	52	52	52	1800	1995.03	-
adpcm-st231.2	82	82	82	7214	7327	-
gsm-st231.1	24	24	32	600	626	4%
gsm-st231.2	59	59	59	7200	7298.04	-
gsm-st231.5	26	26	26	600	600	28%
gsm-st231.6	17	17	17	600	600	37%
gsm-st231.7	28	28	28	22	23	25%
gsm-st231.8	9	9	9	0.02	0.02	13%
gsm-st231.9	28	28	28	25	30	25%
gsm-st231.10	6	6	6	0.0001	0.0001	17%
gsm-st231.11	20	20	21	0.15	0.152	23%
gsm-st231.12	10	10	10	0.0001	0.0001	37%
gsm-st231.13	27	27	27	48	52	39%
gsm-st231.14	20	20	20	18	21	25%
gsm-st231.15	9	9	9	0.001	0.002	30%
gsm-st231.16	38	38	38	420	452	-
gsm-st231.17	24	24	24	720	795	20%
gsm-st231.18	120	120	120	600	603	-
gsm-st231.19	12	12	12	0.002	0.002	35%
gsm-st231.20	13	13	13	0.002	0.002	33%
gsm-st231.21	20	20	22	24	24	38%
gsm-st231.22	18	18	21	8	8	34%
gsm-st231.25	37	37	37	300	317	33%
gsm-st231.29	28	28	28	60	62	33%
gsm-st231.30	16	16	16	0.25	0.253	36%
gsm-st231.31	26	26	26	58	60	33%
gsm-st231.32	21	21	21	3.02	3	30%
gsm-st231.33	33	33	33	52	51	37%
gsm-st231.34	6	6	6	0.001	0.001	14%
gsm-st231.35	11	11	11	0.002	0.002	21%
gsm-st231.36	18	18	18	8	8	25%
gsm-st231.39	15	15	15	0.06	0.08	29%
gsm-st231.40	12	12	12	0.0001	0.0001	30%
gsm-st231.41	34	34	34	47	49	-
gsm-st231.42	14	14	14	4.03	6	22%
gsm-st231.43	15	15	15	0.026	0.02	25%
valeur moyenne	19	19	19.23%	105	110	27%

TABLE 2.8 – Résultats des bornes inférieures obtenues à partir de la relaxation des formulations **{direct+}** et **{decomp+}** pour les instances modifiées

2.6 Conclusion

Dans ce chapitre, les modèles de PLNE pour le problème d'ordonnancement modulo sous contraintes de ressources ont été exploitées. Nous avons réalisé une étude théorique et expérimentale des formulations PLNE les plus connues. Nous avons prouvé l'équivalence des relaxations des formulations PLNE et nous avons également vérifié que la formulation (decomp(+)) est plus rapide que la formulation **{direct(+)}** pour obtenir des solutions entières. Néanmoins d'autres études expérimentales ont montré que la formulation directe s'adaptait bien à des techniques de grand voisinage qui ont obtenus de très bons résultats approchés, ce qui montre donc leur intérêt indépendamment de la qualité des relaxations et du nombre de variables (de Dinechin, 2007).

Le temps de calcul pour l'ensemble des instances industrielles est plus court que pour l'ensemble des instances modifiées. Cela est certainement dû à la différence des consommations des ressources pour chaque groupe d'instances.

En ce qui concerne le calcul des bornes pour la période λ, nous avons constaté que les bornes inférieures λ_{RPL} obtenues à partir de la relaxation d'intégralité sont égales aux bornes triviales λ_{min}. Ceci est vérifié pour les deux groupes d'instances.
L'objectif du chapitre suivant est donc de proposer nouvelles bornes qui améliorent les résultats présentés dans ce chapitre.

Chapitre 3

Formulations renforcées pour le RCMSP basées sur la décomposition de Dantzig-Wolfe

Ce chapitre vise à proposer nouvelles formulations PLNE pour le RCMSP. Le but est d'améliorer les bornes obtenues dans le chapitre précédent pour la période λ. Nous proposons d'utiliser la décomposition de Dantzig-Wolfe pour générer des nouvelles formulations, qui présentent cependant un très grand nombre de variables. Pour faire face à ce problème, nous proposons un schéma de génération de colonnes de manière à résoudre la relaxation de ces formulations. Ces nouvelles formulations sont basées sur l'idée d'ensembles de tâches réalisables qui peuvent être exécutées simultanément, sans violer les contraintes de précédence ou de ressources. Dans ce contexte, l'application de la décomposition de Dantzig-Wolfe sur la formulation à variables indicées pour le temps de Pritsker et al. (1969) spécifie la formulation de Mingozzi et al. (1997) pour l'ordonnancement de projet sous contraintes de ressources (RCPSP). Nous proposons dans ce chapitre d'appliquer cette décomposition aux formulations du RCMSP présentées dans le chapitre précédent, de résoudre les relaxations PL des formulations ainsi obtenues au moyen de la génération de colonnes afin d'obtenir de nouvelles bornes pour la période et le makespan. Des résultats expérimentaux sont donnés. Ce chapitre a fait l'objet d'une communication dans une conférence nationale [ROADEF 2010], une conférence internationale [PMS 2010] et fait partie d'une soumission dans une revue internationale [DAM].

3.1 Introduction

La décomposition de Dantzig and Wolfe (1960) proposée en 1960 est une approche classique pour résoudre un programme linéaire de grande taille. La décomposition de Dantzig-Wolfe consiste à exprimer un polyèdre convexe au travers de ses points extrêmes. Le but est d'exprimer l'ensemble des contraintes par deux sous-ensembles. Une fois la décomposition est apliquée le problème est divisé en deux problèmes ; le problème maître et le sous-problème. En effect, il s'agit de résoudre le problème maître qui est malheureusement un programme linéaire avec un très grand nombre de variables. Dans ce chapitre, nous présentons notre contribution sur l'application d'un tel schéma de décomposition sur les formulations d'ordonnancement modulo

présentées dans le chapitre antérieur. Des nouvelles formulations avec un nombre exponentiel de variables de décision sont obtenues. Ces variables sont liées à l'exécution des sous-ensembles des tâches qui peuvent être exécutés dans la période λ sans violer les contraintes de précédence ou de ressource. Ensuite, nous proposons un schéma de génération de colonnes pour résoudre la relaxation des nouvelles approches et obtenir de nouvelles bornes pour la période λ et pour le Cmax. La première section de ce chapitre traitera donc les notions générales de la décomposition de Dantzig-Wolfe et la génération de colonnes. Ensuite, nous présentons les nouvelles formulations pour l'ordonnancement modulo sous contraintes de ressources obtenues à partir de la décomposition de Dantzig-Wolfe. Un schéma de génération de colonnes pour résoudre la relaxation des formulations est également proposé. Enfin, nous présentons les résultats expérimentaux sur l'ensemble d'instances industrielles et modifiées décrites dans le premier chapitre. Nous comparons les bornes obtenues avec les formulations présentées dans le chapitre antérieur avec les bornes obtenues à partir de ces nouvelles formulations basées sur la décomposition de Dantzig-Wolfe.

3.2 Décomposition de Dantzig-Wolfe et génération de colonnes

La génération de colonnes est une méthode utilisée pour résoudre efficacement les programmes linéaires de grande taille. Elle consiste en la décomposition du programme linéaire original en un problème maître et un sous problème. Le sous-problème est généré à partir du problème dual associé au problème linéaire original issu de la décomposition de Dantzig-Wolfe. Le problème maître contient un premier sous ensemble de colonnes et le sous problème est résolu à partir de l'information du problème dual. Le but est de résoudre le sous-problème pour identifier si le problème maître peut être amélioré en ajoutant de nouvelles colonnes. La génération de colonnes alterne la résolution du problème maître et le sous-problème jusqu'à l'obtention d'un ensemble de colonnes nécessaire pour trouver une solution optimale. La génération de colonnes est basée sur la décomposition de Dantzig-Wolfe. Pour expliquer cette décomposition, considérons le problème suivant :

$$
\begin{array}{llllll}
(P) & min \;\; z = & c^1x^1 + & c^2x^2 & +...+ & c^rx^r \\
s.c & A^1x^1 + & A^2x^2 & +...+ & A^rx^r & = b^0 \\
 & B^1x^1 & & & & = b^1 \\
 & & B^2x^2 & & & = b^2 \\
 & & & \ddots & & \\
 & & & & B^rx^r & = b^r \\
 & & & & x^1, x^2, ..., x^r & \geq 0
\end{array}
$$

Le programme linéaire (P) est composé d'une série de m_0 contraintes $\sum_{j=1}^{r} A^jx^j = b^0$ et m_j contraintes qui impliquent le vecteur de variables x^j. Le domaine des variables est : $c^j, x^j \in \mathbb{R}^{n_j}$, $A^j \in \mathbb{R}^{m_0 \times n_j}$, $b^0 \in \mathbb{R}^{m_0}$, $b^j \in \mathbb{R}^{m_j}$, $B^j \in \mathbb{R}^{m_j \times n_j}$.

Si le polyèdre P^j associé à l'ensemble de contraintes des variables x^j défini un nombre fini de

points extrêmes pour tout j

$$P^j = \{x \in \mathbb{R}^{n_j} \mid B^j x^j = b^j, x^j \geq 0\}$$

alors, tout point $x^j \in P^j$ peut être définit comme : $x^j = \sum_{k=1}^{N_j} \phi_k^j x_k^j$, tel que $\sum_{k=1}^{N_j} \phi_k^j = 1$ et $\phi_k^j \geq 0$, où N_j représente le nombre de sommets dans le polyèdre j. A partir de l'expression de x^j en fonction des points extrêmes, il est possible de réécrire le programme (P) sous la forme d'un nouveau programme linéaire, appelé programme maître et noté (PM), comme suit :

$$\begin{array}{llllll}
(PM) & min \;\; z = & \sum_{k=1}^{N_1} c^1 x_k^1 \phi_k^1 & +...+ & \sum_{k=1}^{N_r} c^r x_k^r \phi_k^r & \\
 & s.c & \sum_{k=1}^{N_1} A^1 x_k^1 \phi_k^1 & +...+ & \sum_{k=1}^{N_r} A^r x_k^r \phi_k^r & = b^0 \\
 & & \sum_{k=1}^{N_1} \phi_k^1 & & & = 1 \\
 & & & \ddots & & \\
 & & & & \sum_{k=1}^{N_r} \phi_k^r & = 1 \\
 & & & & \phi_k^j & \geq 0 \;\; \forall\, k, j
\end{array}$$

Cette reformulation présente moins de contraintes que la formulation initiale car les contraintes liées au vecteur de variables x^j sont remplacées pour une seule contrainte $\sum_{k=1}^{N_j} \phi_k^j = 1$. Elle introduit cependant un nombre exponentiel de variables car le vecteur x^j est remplacée par un scalaire de tous les points positifs de P^j. Dans ce nouveau modèle, les variables de décision sont les $\phi_k^j,\ k = 1, ..., N_j$.

Si dans le programme original (P), la contrainte $x^1, x^2, ..., x^r \geq 0$ est remplacée par

$$x^1, x^2, ..., x^r \in \{0, 1\}^{n_j}$$

nous obtenons alors un programme linéaire $(P)^{'}$ ainsi qu'un polyèdre $P^{j^{'}} = \{x^j \in \{0,1\}^{n_j} \mid B^j x^j = b^j\}$ qui représente l'ensemble de solutions $0-1$ de $(P)^{'}$.

Le programme maître $(PM)^{'}$ lié au programme $(P)^{'}$ peut être donc formulé de la façon suivante :

$$\begin{array}{llllll}
(PM)^{'} & min \;\; z = & \sum_{k=1}^{N_1} c^1 x_k^1 \phi_k^1 & +...+ & \sum_{k=1}^{N_r} c^r x_k^r \phi_k^r & \\
 & s.c & \sum_{k=1}^{N_1} A^1 x_k^1 \phi_k^1 & +...+ & \sum_{k=1}^{N_r} A^r x_k^r \phi_k^r & = b^0 \\
 & & \sum_{k=1}^{N_1} \phi_k^1 & & & = 1 \\
 & & & \ddots & & \\
 & & & & \sum_{k=1}^{N_r} \phi_k^r & = 1 \\
 & & & & \phi_k^j \in \{0,1\}^{n_j} & \forall\, k, j
\end{array}$$

Comme dans le cas des variables x^j continues, cette décomposition spécifie un programme linéaire avec moins de contraintes mais beaucoup plus de variables. Dans les programmes linéaires de grande taille, il est très difficile de représenter toutes les variables de manière explicite. En fait, dans la solution optimale, la plupart des variables sont hors base et donc sont nulles, c'est-à-dire, que seul un sous-ensemble de variables doit être pris en compte pour résoudre le problème. Pour résoudre la relaxation continue de ce problème, nous supposons qu'il existe une solution de base réalisable pour le problème maître (PM). Il reste à determiner si la solution est optimale ou s'il existe au contraire un sommet qui doit être intégré dans la solution de base. Dans ce cas, on ajoute une colonne. Nous commençons par résoudre un programme restreint à un sous-ensemble de variables $\phi^j_{k'}$ noté (RPM). Si nous supposons que $\phi^* \in \mathbb{R}^j_{k'}$ est une solution optimale du problème (RPM), il existe alors une solution (ζ, ν) réalisable du problème dual associé, tel que $\zeta \in \mathbb{R}^{m_0}$ est associée aux contraintes qui restent et $\nu \in \mathbb{R}^r$ est associée à la contrainte de convexité de chacun des r polyèdres P^j. La solution optimale ϕ^* de (RPM) peut être écrite comme une combinaison linéaire de la solution réalisable du problème dual, $b_0\zeta + \nu = c\phi^*$, alors $b_0\zeta + \nu = c\phi^* \geq z(PM)$. Pour évaluer si (ζ, ν) est une solution réalisable du dual de (PM), nous définissons $\bar{c}^j_k$ le coût réduit associé à la variable ϕ^j_k. L'expression du coût est :

$$\bar{c}^j_k = (c^j - \zeta A^j)x^j_k - \nu_j$$

La solution du problème (RPM) est optimale si $\bar{c}^j_k \geq 0 \ \ \forall\, k, j$. En effet, nous pouvons chercher de façon équivalente la valeur minimale de $\bar{c}^j_k$ et vérifier qu'elle est positive. Nous obtenons alors le sous-problème suivant :

$$\begin{aligned}\min_{k,j} \bar{c}^j_k &= \min_{1\leq j\leq r}\{\min_{1\leq k\leq N_j}[(c^j - (\zeta A^j)x^k_j - \nu_j]\} \\ &= \min_{1\leq j\leq r}\{\min_{x^j\in P^j}[(c^j - (\zeta A^j)x_j - \nu_j]\}\end{aligned}$$

trouver $\min \bar{c}^j_k$ est alors équivalent à résoudre r sous-problèmes de type :

$$\begin{array}{ll} (SP^j) & \min f_j = \left(c^j - \zeta A^j\right) x^j - \nu_j \\ s.c & B^j x^j = b^j \\ & x^j \geq 0 \end{array}$$

Si la valeur optimale de (SP^j) est positive ou nulle pour toute j, la solution du problème (RPM) est alors optimale. En conséquence, la solution du problème original (P) est également optimale. Dans le cas où, il existe une valeur de (SP^q) négative pour un certain q, une variable doit être intégrée dans la base de (RPM) (une colonne est donc ajoutée). La méthode de génération de colonnes est cependant très dépendante du mécanisme utilisé pour générer des colonnes. En effet, le sous-problème à résoudre est souvent NP-difficile.

3.3 Formulations renforcées pour le RCMSP

3.3.1 Formulations renforcées pour le RCMSP basées sur la décomposition de Dantzig-Wolfe

Nous nous basons sur l'approche de Vanderbeck (2000) pour proposer des nouvelles formulations pour le RCPSP basées sur la décomposition de Dantzig and Wolfe (1960).

Nous considérons le polyèdre entier borné $\mathcal{P}(\tau)$, qui définit l'ensemble de solutions réalisables respectant les contraintes de ressources pour la formulation décomposée à l'instant $\tau \in \{0, ..., \lambda\}$.

$$\mathcal{P}(\tau) = \left\{ y^\tau \in \{0,1\}^n \mid \sum_{i=1}^{n} y_i^\tau b_i^s \leq B_s, \forall s \in \{1, \ldots, m\} \right\}$$

Il existe une correspondance bijective entre l'ensemble des points réalisables non-nuls de $\mathcal{P}(\tau)$ et l'ensemble des ensembles R de tâches $\{i_1, \ldots, i_q\}$ tel que $\sum_{p=1}^{q} b_{i_p}^s \leq B_s, \forall s \in \{1, \ldots, m\}$. Chaque ensemble de R est appelé un ensemble réalisable. L'ensemble vide correspond à $y = 0$. On définit par R^* l'ensemble qui définit tous les ensembles réalisables et l'ensemble vide. Ils existe au plus 2^n éléments dans R^*.

Les éléments du polyèdre $\mathcal{P}(\tau)$ borné sont notés par P_0, P_1, ..., $P_{|R|-1}$, chacun de ces points correspond à un ensemble réalisable et P_0 correspond à l'ensemble vide. Le polyèdre $\mathcal{P}(\tau)$ peut être représenté par l'énumération de tous ses éléments comme suit :

$$\mathcal{P}(\tau) = \left\{ y^\tau \in \mathbb{R}^n \mid y^\tau = \sum_{l \in R^*} z_l^\tau P_l \text{ et } \sum_{l \in R^*} z_l^\tau = 1 \text{ pour tout } z_l^\tau \in \{0,1\}, l \in R^* \right\}$$

Nous définissons la matrice binaire a tel que $a_i^l = 1$ si la tâche i appartient à l'ensemble réalisable P_l (a_i^l représente le i-ème élément de P_l). Si on remplace y_i^τ par $\sum_{l \in R} a_i^l z_l^\tau$ et on utilise R à la place de R^* (nous ne considérons pas l'ensemble réalisable vide), nous obtenons une nouvelle formulation décomposée pour le RCMSP (M-decomp).

$$min \sum_{i=1}^{n} w_i (k_i * \lambda + \sum_{\tau=0}^{\lambda-1} \tau \sum_{l \in R} a_i^l z_l^\tau)$$

$$\sum_{l \in R} \sum_{\tau=0}^{\lambda-1} a_i^l z_l^\tau = 1 \quad i \in \{1, ..., n\} \tag{3.1}$$

$$\sum_{l \in R} z_l^\tau \leq 1, \tau \in \{0, \ldots, \lambda - 1\} \tag{3.2}$$

$$\sum_{\tau=0}^{\lambda-1} \tau \left(\sum_{l \in R} a_j^l z_l^\tau - \sum_{l \in R} a_i^l z_l^\tau \right) + (k_j - k_j)\lambda \geq \theta_i^j - \omega_i^j \lambda . (i,j) \in E \tag{3.3}$$

$$z_i^\tau \in \{0,1\} \quad \forall i \in \{1, \ldots, n\}, \forall \tau \in \{0, \ldots, \lambda - 1\}$$

$$k_i \in \{0, \ldots, K-1\}, \quad \forall i \in \{1, \ldots, n\}$$

Dans cette formulation, la variable binaire z_l^τ est égale à 1 si et seulement si le sous-ensemble l est exécuté à l'instant τ.

La relaxation de la formulation (M-decomp) est plus forte que celle de la formulation (decomp) car les contraintes de ressources ont été remplacées par l'enveloppe convexe de l'ensemble entier $\mathcal{P}(\tau)$, c'est-à-dire :

$$conv\,(\mathcal{P}(\tau)) = \left\{ y^\tau \in \mathbb{R}^n \mid y^\tau = \sum_{l \in R} z_l P_l,\ \sum_{l \in R} z_l = 1\ \forall\ 0 \leq z_l \leq 1, l \in R \right\}$$

A partir des contraintes de précédence structurées (2.25), il est possible de remplacer les contraintes (3.3) par les contraintes suivantes :

$$\sum_{x=\tau}^{\lambda-1} \sum_{l \in R} a_i^l z_l^x + \sum_{x=0}^{(\tau+\theta_i^j-1) \mod \lambda} \sum_{l \in R} a_j^l z_l^x + k_i - k_j \leq \omega_i^j - \lfloor \frac{\tau + \theta_i^j - 1}{\lambda} \rfloor + 1,$$
$$\forall \tau \in \{0, \ldots, \lambda - 1\}, \forall (i,j) \in E \tag{3.4}$$

Une nouvelle formulation renforcé appelée (M-decomp+) est donc obtenue. Cette formulation est également plus forte que la formulation (decomp+) pour la même raison que dans le cas précédent.

Il est possible d'obtenir des nouvelles formulations à partir de la décomposition de Dantzig-Wolfe sur les formulations directes. Cependant, nous conservons les variables x pour les contraintes de précédence et nous utilisons la transformation (2.32) pour établir un lien entre les variables z et x. Nous obtenons la formulation directe (M-direct) suivante lorsque les contraintes de précédence simples sont considerées :

$$min \sum_{i=1}^{n} w_i \sum_{t=0}^{T} t x_i^t$$

$$\sum_{t=0}^{T} x_i^t = 1, i = 1, ..., n$$

$$\sum_{l \in R} \left(\sum_{\tau=0}^{\lambda-1} a_i^l z_l^\tau \right) = 1,\ \forall i = 1, ...n \tag{3.5}$$

$$\sum_{l \in R} z_l^\tau \leq 1,\ \forall \tau \in [0, \lambda - 1] \tag{3.6}$$

$$\sum_{t=0}^{T} t x_i^t + \theta_i^j - \lambda \omega_i^j \leq \sum_{t=0}^{T} t x_j^t,\ \forall (i,j) \in E \tag{3.7}$$

$$\sum_{l \in R} a_i^l z_l^\tau = \sum_{k=0}^{K-1} x_i^{\tau + k\lambda} \quad \forall i \in \{1, \ldots, n\}, \forall \tau \in \{0, \ldots, \lambda - 1\} \tag{3.8}$$

$$x_i^t \in \{0, 1\},\ \forall i \in \{1, \ldots, n\},\ \forall t \in \{0, \ldots, T - 1\}$$

$$z_i^\tau \in \{0,1\}, i \in \{1,\ldots,n\}, \tau \in \{0,\ldots,\lambda-1\}$$

Lorsque les contraintes de précédence simples sont remplacées pour les contraintes de précédence désagrégées (2.31), nous obtenons la formulation (M-direct+). Comme dans le cas des formulations décomposées, les relaxations des formulations (M-direct) et (M-direct+) sont respectivement plus fortes que celles des formulations (direct) et (direct+). Notons que l'application de la décomposition de Dantzig-Wolfe sur la formulation avec les variables indexées par le temps de Pritsker *et al.* (1969) définit la formulation de Mingozzi *et al.* (1997) pour l'ordonnancement de projet sous contraintes de ressources (RCPSP). Cette formulation a été cependant proposée sans faire aucune référénce à la décomposition de Dantzig-Wolfe.

3.3.2 Réduction du nombre d'ensembles réalisables

Nous avons déjà remarqué dans la section précédente qu'il existe au plus 2^n ensembles réalisables de tâches. Toutefois, ce nombre maximal n'est jamais atteint lorsque les contraintes de ressource sont actives. En effet, tous les sous-ensembles de tâches avec demandes cumulatives qui dépassent les disponibilités des ressources peuvent être écartés *a priori*.

Dans la formulation proposée par Mingozzi *et al.* (1997) pour le problème RCPSP non-périodique, la réduction de nombre d'ensembles réalisables est obtenu à partir de l'hypothèse que deux tâches liées par des contraintes de précédence ne peuvent pas faire partie du même ensemble réalisable. Dans le contexte du RCMSP, cette affirmation n'est pas suffisante et nous devons considérer des conditions plus complexes. Ces conditions sont décrites ci-dessous.

D'abord, nous définisons les conditions pour avoir $\sigma_i mod\lambda = \sigma_j mod\lambda$, c'est-à-dire, $\tau_i = \tau_j$. Alors, s'il existe une contrainte de précédence (i,j), nous avons :

$$\sigma_j \geq \sigma_i + \theta_i^j - \lambda\omega_i^j$$

$$\tau_j + k_j\lambda \geq \tau_i + k_i\lambda + \theta_i^j - \lambda\omega_i^j$$

$$\tau_j \geq \tau_i + k_i\lambda - k_j\lambda + \theta_i^j - \lambda\omega_i^j$$

$$\tau_j - \tau_i \geq (k_i - k_j)\lambda + \theta_i^j - \lambda\omega_i^j$$

Pour avoir $\tau_i = \tau_j$, nous devons trouver k_i et k_j tel que

$$k_j \geq k_i + \left\lceil \frac{\theta_i^j}{\lambda} \right\rceil - \omega_i^j \tag{3.9}$$

avec k_i et k_j entiers.
Dans le cas où il existe aussi une contrainte de précédence (j,i), alors

$$\tau_i - \tau_j \geq (k_j - k_i)\lambda + \theta_j^i - \lambda\omega_j^i$$

Si $\tau_i = \tau_j$, nous avons

$$k_i \geq k_j + \left\lceil \frac{\theta_j^i}{\lambda} \right\rceil - \omega_j^i \tag{3.10}$$

Dans ce cas trouver des valeurs k_i et k_j tel que (3.9) et (3.10) soient vérifiées avec k_i, k_j entiers est possible si et seulement si

$$\left\lceil \frac{\theta_i^j}{\lambda} \right\rceil - \omega_i^j + \left\lceil \frac{\theta_j^i}{\lambda} \right\rceil - \omega_j^i \leq 0 \tag{3.11}$$

Si nous supposons que le problème a une contrainte de précédence précédence réalisable aussi, nous avons

$$\theta_i^j - \lambda\omega_i^j + \theta_j^i - \lambda\omega_j^i \leq 0 \tag{3.12}$$

Nous démontrons que généralement (3.12) $\not\Rightarrow$ (3.11) avec un contre-exemple. Supposons que $\theta_i^j = 1, \omega_i^j = 0, \theta_j^i = 1, \omega_j^i = 1$ et $\lambda = 2$. (3.12) est satisfait car $\theta_i^j - \lambda\omega_i^j + \theta_j^i - \lambda\omega_j^i = 1-0+1-2 = 0$ et une solution réalisable est donnée par $\sigma_i = 0$ et $\sigma_j = 1$. Toutefois, cette solution ne satisfait pas (3.9) et (3.10) car $\left\lceil \frac{\theta_i^j}{\lambda} \right\rceil - \omega_i^j + \left\lceil \frac{\theta_j^i}{\lambda} \right\rceil - \omega_j^i = \left\lceil \frac{1}{2} \right\rceil - 0 + \left\lceil \frac{1}{2} \right\rceil - 1 > 0$. Ainsi, une condition nécessaire pour avoir $\tau_i = \tau_j$ est donnée par (3.11). Si cette condition n'est pas satisfaite, i et j ne peuvent pas appartenir au même ensemble admissible. Cette règle peut être appliquée pour réduire *a priori* le nombre d'ensembles de tâches réalisables à considérer dans l'ensemble R. En fait, ce principe peut être étendu pour tout circuit dans le graphe de précédences.

Théorème 3 *Soit* (i_1, i_2), (i_2, i_3), $\ldots$, $(i_q, i_{q+1} = i_1)$ *un circuit dans le graphe de précédences, les sous-ensembles des tâches* $\{i_1, \ldots, i_q\}$ *ne peuvent faire partie du même ensemble réalisable si* $\sum_{s=1}^{q} \left\lceil \frac{\theta_{i_s}^{i_s+1}}{\lambda} \right\rceil - \omega_{i_s}^{i_s+1} > 0$

Preuve La preuve est triviale à partir de la généralisation directe du cas avec deux tâches. ∎

3.4 Génération de colonnes pour le RCMSP

Les formulations (M-direct), (M-direct+), (M-decomp) et (M-decomp+) présentent un grand nombre de variables ($\lambda\times \mid R_l \mid$). Nous proposons un schéma de génération de colonnes pour résoudre les relaxations continues de ces formulations.

Dans la formulation (M-decomp), ρ_i dénote la variable duale associée aux contraintes (3.1), β_τ dénote la variable duale liée aux contraintes (3.2) et δ_{ij} la variable duale qui correspond aux contraintes (3.3).

Les contraintes duales associée à la variable z_l^τ du problème maître donné par la formulation (M-decomp) sont formulées de la façon suivante :

$$\sum_{i=1}^{n} a_i^l \sigma_i - \beta_\tau + \sum_{(i,j)\in E} \delta_{ij}\tau(a_j^l - a_i^l) \leq \sum_{i=1}^{n} w_i \tau a_i^l, \forall l \in R, \tau \in \{0, \ldots, \lambda - 1\}$$

ce qui donne

$$\sum_{i=1}^{n} w_1(i,\tau) a_i^l \leq \beta_\tau, \forall l \in R, \tau \in \{0, \ldots, \lambda - 1\}$$

où $w_1(i,\tau) = \rho_i + \left(\tau \sum_{(j,i)\in E} \delta_{ji} - \sum_{(i,j)\in E} \delta_{ij} + w_i\right)$.

La formulation (M-decomp+) diffère de la formulation (M-decomp) par les contraintes de précédence désagrégées (3.4). Soit δ_{ij}^{τ} la variable duale des contraintes (3.4). Nous obtenons les contraintes duales suivantes correspondant à la variable z_l^{τ} dans le problème maître donné par la formulation (M-decomp+) :

$$\sum_{i=1}^{n} w_2(i,\tau)a_i^l \leq \beta_\tau, \forall l \in R, \tau \in \{0,\ldots,\lambda-1\}$$

où $w_2(i,\tau) = \rho_i + \sum_{(j,i)\in E}((\tau+\theta_j^i-1) \mod \lambda+1)\delta_{ji}^{\tau} + \sum_{(i,j)\in E}(\lambda-\tau)\delta_{ij}^{\tau} + w_i\tau$.

La formulation (M-direct) diffère de la formulation (M-decomp) par la contrainte de précédence et par le fait que la fonction objectif ne prenne pas en compte les variables z_l^{τ}. Cependant, les contraintes (3.8) lient les variables x and z. Soit γ_i^{τ} les variables duales correspondantes aux contraintes (3.8). Nous obtenons les contraintes duales suivantes pour les variables z_l^{τ} associées au problème maître donné par la formulation (M-direct) :

$$\sum_{i=1}^{n} w_3(i,\tau)a_i^l \leq \beta_\tau, \forall l \in R, \tau \in \{0,\ldots,\lambda-1\}$$

où $w_3(i,\tau) = \rho_i + \gamma_i^{\tau}$.

Comme les contraintes de précédence désagrégées ne prennent pas en compte les variables z, les contraintes duales sont alors identiques pour la formulation (M-direct+).

A partir d'une formulation donnée, il est possible de trouver une contrainte duale violée, c'est-à-dire, un vecteur a_i tel que $\sum_{i=1}^{n} w_q(i,\tau)a_i \leq \beta_\tau$ (pour $q = 1,2,3$). La contrainte duale violée peut être determinée avec la résolution des sous-problèmes $(SP_q(\tau))$ suivants :

$$max \sum_{i=1}^{n} w_q(i,\tau)a_i \tag{3.13}$$

$$\sum_{i=1}^{n} a_i b_i^s \leq B_s, \forall s \in \{1,\ldots,m\} \tag{3.14}$$

$$a_i \in \{0,1\}, \quad \forall i \in \{1,\ldots,n\} \tag{3.15}$$

Le sous-problème correspond à un problème de sac-à-dos multidimensionnel pour chaque $\tau = 0,...,\lambda-1$. Nous avons donc λ sous-problèmes à résoudre.

En premier lieu, le problème maître est résolu, puis les λ sous-problèmes sont également résolus.

Dès que la solution d'un sous problème dépasse β_τ pour une valeur de τ donné, la variable a_i correspondante génère un nouvel ensemble réalisable, et une nouvelle variable z_l^{τ} est ajoutée dans le problème maître. Le nouveau problème maître est alors résolu. Ce processus itératif est repeté jusqu'à ce qu'aucune contrainte duale violée ne soit trouvée pour tout $\tau \in \{0,\ldots,\lambda-1\}$.

Les règles présentées dans la section 3.3.2 peuvent être utilisées pour réduire l'espace de solution du sous-problème en considérant les contraintes :

$$\sum_{i\in U} a_i \leq |U|-1, \quad \forall U \in \mathcal{U}$$

où $\mathcal{U}$ est l'ensemble d'ensembles des tâches $U = \{i_1, \ldots, i_q\}$ qui vérifient $\sum_{s=1}^{q} \left\lceil \frac{\theta_{i_s}^{i_s+1}}{\lambda} \right\rceil - \omega_{i_s}^{i_s+1} > 0$ comme dans le théorème 3. Ces contraintes peuvent être ajoutées sous la forme d'inégalités valides au sous problème (3.13-3.15).

3.4.1 Ensemble initial des colonnes

Une solution réalisable du problème maître restreint est nécessaire pour assurer que l'information duale transférée au sous-problème est correcte. Une telle solution réalisable peut toujours être trouvée en utilisant une méthode à deux-phases, comme la méthode à deux-phases de l'algorithme du simplexe, pour trouver une solution de base réalisable (Barnhart *et al.*, 1998). Pour déterminer un premier problème maître restreint réalisable, nous ajoutons λ variables artificielles C_τ, $\tau \in [0, \lambda-1]$ et nous autorisons la violation des contraintes de convexité (3.2). Ces contraintes sont remplacées par

$$\sum_{l \in R} z_l^\tau - C_\tau \leq 1, \tau \in [0, \lambda - 1] \tag{3.16}$$

Ensuite, l'objectif consiste à résoudre le problème d'optimisation qui consiste à rendre ces variables artificielles nulles pour pouvoir les extraire du problème. Pour cela, nous minimisons la somme des variables artificielles ($\min \sum_{\tau=0}^{\lambda-1} C_\tau$).

L'ensemble de colonnes est initialisé avec des ensembles réalisables singletons $\{i\}$. Le processus itératif commence avec l'objectif modifié jusqu'à ce que $\sum\limits_{\tau=0}^{\lambda-1} C_\tau = 0$. Ensuite, nous passons à la phase 2, où on considère l'objectif original et les contraintes de convexité. Nous remarquons que la phase 1 de la génération de colonnes peut s'achever avec une valeur d'objectif positive, ce qui indique que le problème est non-réalisable pour cette valeur de λ.

3.4.2 Calcul des bornes inférieures

La plus petite valeur de λ réalisable obtenue avec la solution des relaxations des nouvelles formulations proposées, en utilisant le schéma de génération de colonnes pour le RCMSP présenté ci-dessus, est une borne inférieure pour la période λ. Comme nous avons montré dans le chapitre précédent, la recherche d'une borne inferieure pour la période λ est un processus itératif qui commence avec une valeur de $\lambda_{min} = \max(\lambda^{res}, \lambda^{prec})$ (voir chapitre 1, section 1.3.2). Si avec cette valeur de λ il n'existe pas une solution admissible, la valeur de λ est incrementée ($\lambda = \lambda+1$). La borne pour le makespan est définie comme dans le chapitre antérieur, $Cmax_{min} = \max(\lambda, Cmax)$, où Cmax correspond à la valeur obtenue avec la résolution des relaxations en utilisant le schéma de génération de colonnes, uniquement si le programme linéaire admet une solution réalisable pour une valeur de λ donnée.

3.5 Résultats expérimentaux

Dans cette section nous évaluons la performance de la relaxation des nouvelles formulations proposées. Nous présentons également une comparaison des bornes obtenues dans le chapitre antérieur à partir des formulations PLNE et les bornes obtenues dans ce chapitre en utilisant le schéma de génération de colonnes proposé. Les expérimentations ont été exécutées sur Cplex

11.2 avec un Intel(R) Core(TM)2 Duo CPU E4400 @ 2.00GHz 1.99 GHz RAM. Le problème maître et le sous-problème sont résolus avec CPLEX. Nous avons utilisé les deux groupes des instances décrites dans le premier chapitre et déjà utilisées dans le chapitre 2.

Pour l'ensemble des instances industrielles, toutes les instances ont été résolues. Cependant, les résultats expérimentaux ont montré que la borne pour la période λ obtenue avec le schéma de génération de colonnes proposé trouve la même valeur que la borne triviale λ_{min}. Ce résultat est normal puisque cette borne est optimale pour la quasi-totalité des instances industrielles.

En ce qui concerne le makespan, les résultats expérimentaux montrent que la relaxation de ces nouvelles formulations n'améliore pas les résultats obtenus par la relaxation des formulations classiques de PLNE présentées au chapitre précédent. Ce résultat peut s'expliquer par la structure de l'architecture utilisée. Puisque la demande en ressources est unitaire, le sous problème devient facile et, dans ce cas, la génération de colonnes ne peut améliorer la formulation initiale. Malheureusement, nous ne pouvons donc pas utiliser l'ensemble des instances industrielles pour évaluer les performances des nouvelles formulations proposées.

Pour l'ensemble des instances modifiées, les résultats expérimentaux sont beaucoup plus intéressants. Nous rappelons que dans ce cas, la demande de ressources de type non-unitaire augmente la difficulté des problèmes. Le schéma de génération de colonnes proposé permet d'ameliorer la qualité des bornes pour la période λ et pour le makespan. Nous avons réalisé les tests pour les quatre formulations proposées (M-decomp), (M-decomp+),(M-direct) et (M-direct+). Par rapport à la borne pour la période λ, nous avons obtenu des résultats identiques avec les quatre formulations. En ce qui concerne la borne pour le Cmax, les formulations (M-decomp) et (M-direct) ont trouvé des bornes identiques, ainsi que les formulations (M-decomp+) et (M-direct+). Cependant, les formulations renforcées ont donné toujours une meilleure relaxation. Nous montrons les résultats obtenus à partir de la formulation (M-decomp+) car cette formulation a montré les temps d'exécution les plus courts et nous sommes intéressés par les meilleurs résultats. La table 3.1 présente un récapitulatif des résultats obtenus pour la période λ.

Nombre d'instances résolues	34/36
Test avec $\lambda = \lambda^{opt}$	23/28
Ecart maximal $\lambda^{opt} - \lambda$	1
Ecart moyen $\lambda^{opt} - \lambda$	0.18

TABLE 3.1 – Récapitulatif des bornes obtenues pour la période λ

Bien que le nombre total d'instances soit égal à 36, nous avons uniquement considéré les instances résolues optimalement pour les formulations PLNE, c'est-à-dire, 28 instances.

En ce qui concerne la qualité des bornes obtenues pour la période λ, la borne de 82% des instances (23 instances) correspond à la valeur optimale de la période λ. Les instances dont la borne est inférieure à la valeur optimal (5 instances), présentent un écart maximal d'une unité par rapport à la solution optimale. Les bornes obtenues pour la période λ sont donc très efficaces avec un écart moyen de 0.18.

La table 3.2 présente un récapitulatif des résultats obtenus pour le Cmax. Dans ce cas, la borne de 54% des instances (15 instances) correspond à la valeur $Cmax^{opt}$. L'écart maximal est de 12 et l'écart moyen est de 2.58.

Nombre d'instances résolus	34/36
Test avec Cmax=Cmax^{opt}	15/28
Ecart maximal Cmax−Cmax^{opt}	12
Ecart moyen Cmax−Cmax^{opt}	2.58

TABLE 3.2 – Récapitulatif des bornes obtenues pour Cmax

Dans la table 3.3, nous présentons une comparaison entre les bornes pour la période λ obtenues avec les relaxation des formulations PLNE et les bornes obtenues à partir des nouvelles approches.

	Relaxation PLNE (decomp+)	Nouvelle approche (M-decomp+)
Nombre d'instances résolues	36/36	34/36
Test avec $\lambda = \lambda^{opt}$	0/28	15/28
Ecart maximal $\lambda^{opt} - \lambda$	16	1
Ecart moyen $\lambda^{opt} - \lambda$	7.10	0.58

TABLE 3.3 – Comparaison des bornes pour la période λ

Bien que la relaxation des formulations PLNE permette la résolution de l'intégralité des instances, nous observons que la qualité de la borne pour la période λ est moins bonne. Avec la relaxation PLNE, les bornes n'atteignent la valeur optimale pour aucune des instances. L'écart maximal est de 16 et l'écart moyen est de 7.10. En ce qui concerne le Cmax, la table 3.4 montre que les bornes obtenues avec les nouvelles formulations sont aussi beaucoup plus efficaces.

	Relaxation PLNE (decomp+)	Nouvelle approche (M-decomp+)
Nombre d'instances résolues	36/36	34/36
Test avec Cmax=Cmax^{opt}	0/28	15/28
Ecart maximal Cmax−Cmax^{opt}	25	12
Ecart moyen Cmax−Cmax^{opt}	9.57	2.58

TABLE 3.4 – Comparaison des bornes pour le Cmax

En ce qui concerne les temps de calcul, la table 3.5 montre la répartition du nombre d'instances résolues classeés en fonction de leur taille (nombre d'opérations à ordonnancer) et du temps de calcul nécessaire pour leur résolution. La colonne NR correspond au nombre d'instances non résolues.

Formulation		(decomp+)				(M-decomp+)				
		ms	*sec*	*min*	$> 1h$	*ms*	*sec*	*min*	$> 1h$	NR
♯Oper	♯Instances									
10 - 30	16	13	1	2	-	4	12	-	-	-
31 - 60	15	0	11	4	-	0	9	6	-	-
61 - 100	2	0	0	2	-	0	0	2	0	-
> 100	3	0	0	2	1	0	0	0	1	2
Total	36	13	12	10	1	4	21	8	1	2

TABLE 3.5 – Comparaison du nombre d'instances résolues classées en fonction de taille et temps de calcul.

La table 3.5 montre que la résolution de la relaxation de la formulation (decomp+) est plus rapide que la résolution de la relaxation de la formulation (M-decomp+) en utilisant le schéma de génération de colonnes proposé. On peut expliquer ces différences de temps par les caractéristiques d'implémentation du schéma de génération de colonnes. En effet, nous avons réalisé une implémentation basique qui ne prend pas en compte les règles déduites pour la réduction du nombre d'ensembles réalisables.

Les figures 3.1 et 3.2 montrent la comparaison des bornes pour la période λ et pour le Cmax respectivement. La table 3.6 montre en détail les résultats obtenus pour chaque instance. Nous présentons les bornes obtenues avec la relaxation de la formulation (decomp+), les bornes obtenues avec la nouvelle approche (M-decomp+) et les solutions optimales obtenues dans le chapitre 2 avec les formulations PLNE.

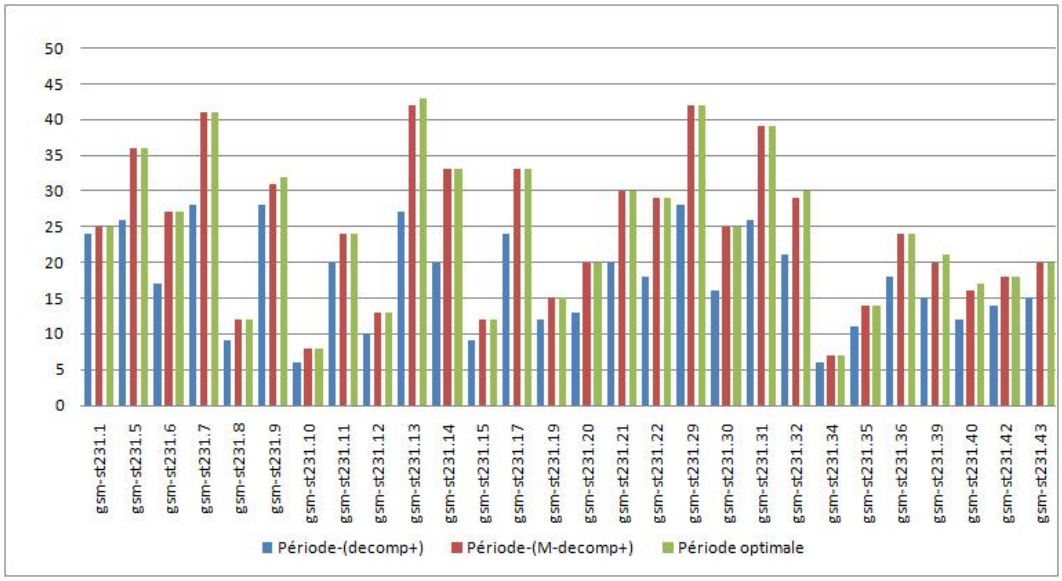


FIGURE 3.1 – Bornes pour la période λ. Instances modifiées.

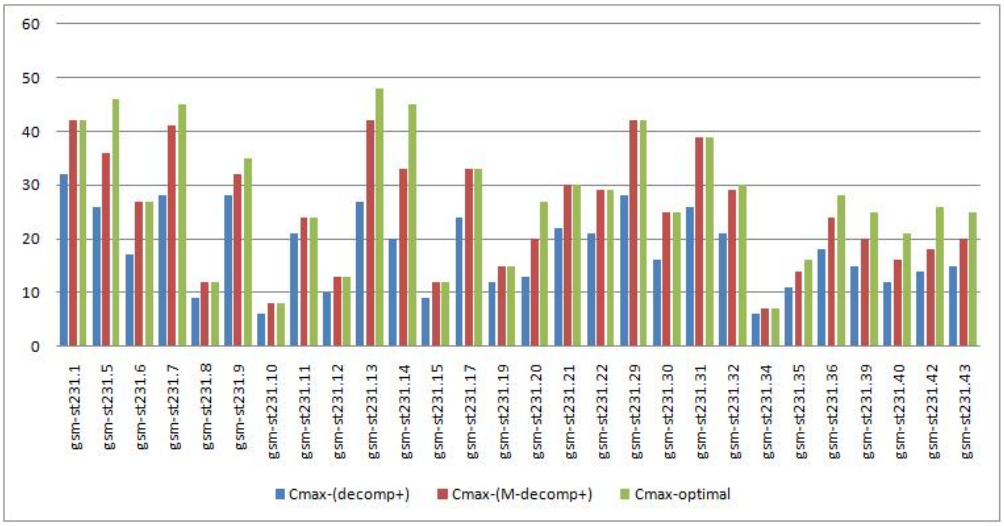


FIGURE 3.2 – Bornes pour le Cmax. Instances modifiées.

Instances	(decomp+)			(M-decomp+)			Opt	
	λ	$Cmax$	CPU_s	λ	$Cmax$	CPU_s	λ	$Cmax^*$
adpcm-st231.1	52	52	1800	79	79	1758.38	-	-
adpcm-st231.2	82	82	420	-	-	-	-	-
gsm-st231.1	24	32	600	25	42	32.15	25	42
gsm-st231.2	59	59	7200	93	61	4823	-	-
gsm-st231.5	26	26	600	36	36	385.27	36	46
gsm-st231.6	17	17	600	27	27	17.44	27	27
gsm-st231.7	28	28	22	41	41	17.63	41	45
gsm-st231.8	9	9	0.02	12	12	27.36	12	12
gsm-st231.9	28	28	25	31	32	2.57	32	35
gsm-st231.10	6	6	0.0001	8	8	1.73	8	8
gsm-st231.11	20	21	0.15	24	24	1.2	24	24
gsm-st231.12	10	10	0.0001	13	13	0.08	13	13
gsm-st231.13	27	27	48	42	42	48.29	43	48
gsm-st231.14	20	20	18	33	33	15.7	33	45
gsm-st231.15	9	9	0.001	12	12	16.1	12	12
gsm-st231.16	38	38	420	59	59	469	-	-
gsm-st231.17	24	24	720	33	33	16.01	33	33
gsm-st231.18	120	120	600	-	-	-	-	-
gsm-st231.19	12	12	0.002	15	15	0.3	15	15
gsm-st231.20	13	13	0.002	20	20	0.88	20	27
gsm-st231.21	20	22	24	30	30	10.78	30	30
gsm-st231.22	18	21	8	29	29	1.85	29	29
gsm-st231.25	37	37	300	55	55	175.52	56 (Gap=1.75%)	56
gsm-st231.29	28	28	60	42	42	28.82	42	42
gsm-st231.30	16	16	0.25	25	25	3.48	25	25
gsm-st231.31	26	26	58	39	39	28.4	39	39
gsm-st231.32	21	21	3.02	29	29	627	30	30
gsm-st231.33	33	33	52	51	51	3287.42	52(Gap=8%)	50
gsm-st231.34	6	6	0.001	7	7	1.56	7	7
gsm-st231.35	11	11	0.002	14	14	0.46	14	16
gsm-st231.36	18	18	8	24	24	333.16	24	28
gsm-st231.39	15	15	0.06	20	20	9.17	21	25
gsm-st231.40	12	12	0.0001	16	16	2.56	17	21
gsm-st231.41	34	34	47	49	49	812.74	-	-
gsm-st231.42	14	14	4.03	18	18	13.4	18	26
gsm-st231.43	15	15	0.026	20	20	25.43	20	25

TABLE 3.6 – Bornes inférieures pour la période λ et Cmax obtenues par génération de colonnes pour les instances modifiées.

3.6 Conclusion

Dans ce chapitre, nous avons proposé des nouvelles formulations basées sur la décomposition de Dantzig-Wolfe pour résoudre le problème d'ordonnancement modulo sous contraintes de ressources. Ces nouvelles formulations comportent un grand nombre de variables. Nous proposons un schéma de génération de colonnes à deux phases pour la résolution de leur relaxation. Les sous-problèmes correspondent à λ problèmes de sac-à-dos multidimensionnel. Ces nouvelles formulations s'avèrent très efficaces. Nous avons réussi à obtenir une amélioration significative des bornes pour la période λ et pour le Cmax dans le cas des instances modifiées. Dans le cas des instances industrielles, le schéma de génération de colonnes ne présente pas de différences avec les résultats obtenus par la résolution des relaxations des PLNE classiques. Ceci peut s'expliquer pour le typologie des demandes de ressources qui sont souvent des demandes unitaires, ce qui rend le sous-problème facile. En ce qui concerne le temps d'exécution, la résolution de la relaxation des nouvelles formulations, nécessite d'avantage de temps de calcul que la résolution des relaxations des PLNE classiques. Cependant, une implémentation plus sophistiquée, qui prend en compte les règles déduites pour la réduction du nombre d'ensembles réalisables, peut être envisagée de manière à améliorer les résultats.

Chapitre 4

Relaxation Lagrangienne et solutions réalisables

Dans ce chapitre nous presentons une méthode de relaxation Lagrangienne et une heuristique permettant d'obtenir des bornes supérieures pour la période. Nous proposons de résoudre la relaxation Lagrangienne obtenue à partir de la dualisation de la contrainte de ressources dans la formulation directe renforcée proposée par Dinechin (2004). Dans ce contexte, nous présentons une méthode de résolution basée sur les résultats de Möhring et al. (2000). Les auteurs ont montré pour un problème similaire que le sous-problème Lagrangien est équivalent à résoudre un problème de coupe minimale dans un graphe orienté. La relaxation Lagrangienne permet d'obtenir la même borne (période et Cmax) que la relaxation de la formulation directe mais sans nécessiter l'utilisation d'un solveur de programmation linéaire. Pour l'heuristique, nous nous basons sur une modification de la formulation décomposée proposée par Eichenberger and Davidson (1997). Nous proposons de remplacer dans la formulation PLNE les variables de décision k_i par un nouveau vecteur $k_i^{'}$. Ce vecteur $k_i^{'}$ peut être exprimé en fonction des valeurs x_i^t determinés dans le calcul de la coupe minimale lors de la résolution de la relaxation Lagrangienne. Ce chapitre a fait l'objet d'une communication dans une conférence internationale [ISCO 2010] et il fait également partie d'une publication dans Electronics Notes in Discrete Mathemathics 36-(2010) 191-198.

4.1 Introduction

La résolution de programmes linéaires en nombres entiers est souvent basée sur des méthodes de décomposition. Ces méthodes visent à décomposer le problème en sous-problèmes qui sont plus faciles à résoudre. Une décomposition entraîne la relaxation de certaines contraintes et permet donc de déterminer une borne pour le problème. Comme nous l'avons vu dans le chapitre précédent, le principe des méthodes de décomposition est de décomposer le problème général en un ou plusieurs sous-problèmes qui sont généralement coordonnés par un programme linéaire appelé le problème maître. L'objectif est toujours d'approcher l'enveloppe convexe des solutions réalisables du PLNE. La relaxation Lagrangienne s'applique lorsque dans la matrice de contraintes il est possible d'identifier un sous-ensemble de contraintes qui rendent le problème difficile. Le but de la relaxation est donc de génerer des problèmes qui sont plus faciles à résoudre Geoffrion (1974). Dans ce chapitre, nous proposons le calcul de bornes inférieures basées sur la formulation directe désagrégée proposée par Dinechin (2004) (voir chapitre 2, section 2.4.2). Nous savons

que la borne est de mauvaise qualité pour la période mais nous cherchons ici à calculer une borne inférieure pour le makespan et des solutions réalisables de bonne qualité. Par ailleurs la relaxation Lagrangienne peut avoir un intérêt pour proposer méthodes de recherche des solutions réalisables entières (en général irréalisables), notre heuristique utilise cette propriété.

Nous proposons la dualisation des contraintes de ressources qui vont être donc associées à des multiplicateurs de Lagrange. Cette dualisation génère un sous-problème Lagrangien qui, comme ont montré Möhring *et al.* (2000), peut être résolu à partir du calcul d'une coupe minimale dans un graphe orienté. Une fois le sous-problème Lagrangien résolu, nous proposons la génération d'un vecteur $k_i^{'}$ $\forall i = 1, ..., n$ à partir de la transformation 2.33 qui permet d'exprimer les variables de décision k_i de la formulation décomposée proposée par Eichenberger and Davidson (1997) (voir chapitre 2, section 2.4.1) en fonction des variables x_i^t. Dans ce cas, les valeurs de x_i^t sont determinées à partir de la coupe minimale obtenue lors de la résolution du sous-problème Lagrangien. Ensuite, nous proposons la recherche de bornes supérieures à partir de la résolution de la formulation décomposée structurée en remplaçant les variables de décision k_i par le nouveau vecteur $k_i^{'}$.

La deuxième section de ce chapitre décrit les notions générales de la relaxation Lagrangienne. La troisième section présente la relaxation Lagrangienne pour la formulation directe désagrégée ainsi que la définition du sous-problème Lagrangien. Ensuite, nous décrivons la résolution du problème dual Lagrangien par l'algorithme du sous-gradient. Dans la dernière section, nous présentons l'heuristique proposée et des résultats expérimentaux.

4.2 Relaxation Lagrangienne

La relaxation Lagrangienne est une méthode qui peut être appliquée lorsqu'il est possible de reconnaître des contraintes difficiles dans la matrice des contraintes. La relaxation de ces contraintes génère des sous-problèmes qui sont plus faciles à résoudre (Geoffrion, 1974). Les contraintes qui vont être relachées sont connues sous le nom de contraintes *compliquantes*. Ces contraintes relâchées sont pondérées par des coefficients que l'on appelle *multiplicateurs de Lagrange* et ensuite réinjectées dans la fonction objectif.

Soit $X = \{x \in \mathbb{N}^n \mid Bx \leq b\}$. Considérons le programme linéaire en nombres entiers suivant

$$\begin{gathered} P : min\ z = cx \\ s.c \\ Ax \leq a \\ Bx \leq b \\ x \in X \end{gathered}$$

Admettons que l'ensemble de contraintes $Ax \leq a$ est l'ensemble de contraintes difficiles. Soit $\delta \in \mathbb{R}_+^m$ un vecteur non-négatif de multiplicateurs Lagrangiens. Nous définissons la relaxation Lagrangienne de P associée aux contraintes difficiles et au vecteur Lagrangien δ comme le programme linéaire $(L_x(\delta))$ suivant :

$$\begin{array}{c} L_x(\delta) : min\ z = cx + \delta(Ax - a) \\ s.c \\ Bx \leq b \\ x \in X \end{array}$$

La violation des contraintes ($Ax \leq a$) est ainsi pénalisée dans la fonction objectif par l'intermédiaire des multiplicateurs de Lagrange δ.

Pour un vecteur δ donné et $L_x(\delta)$ sur $x \in X$ (ou sur $x \in conv(X)$), nous obtenons une solution optimale $x(\delta)$. Trouver la meilleure solution $x(\delta)$ possible définit le sous-problème dual Lagrangien L_δ :

$$L_\delta = \max_{\delta \in \mathbb{R}^m_+} L_x(\delta)$$

Nous remarquons que le sous-problème dual Lagrangien L_δ est un problème dans l'espace dual des multiplicateurs Lagrangiens.

4.2.1 Solution Lagrangienne réalisable

$x(\delta)$ n'est pas en générale une solution réalisable pour le problème original P. Soit z^* la valeur optimale de la fonction objectif de P, le théorème suivant montre que la valeur z^* reste toujours entre $cx(\delta) + \delta\,[Ax(\delta) - a] \leq z^* \leq cx(\delta)$.

Théorème 4 *1. Si $x(\delta)$ est une solution optimale de $L_x(\delta)$ pour un $\delta \geq 0$, alors $cx(\delta) + \delta\,[Ax(\delta) - a] \leq z^*$.*

2. Si $x(\delta)$ est en plus une solution réalisable pour P, alors $cx(\delta) + \delta\,[Ax(\delta) - a] \leq z^* \leq cx(\delta)$.

3. De plus $\delta\,[Ax(\delta) - a] = 0$, alors $x(\delta)$ est une solution optimale de P, c'est-à-dire, $cx(\delta) = z^*$.

Preuve Voir Geoffrion (1974). ∎

Nous remarquons que si le sous-problème dual Lagrangien intègre la propriété d'intégralité, c'est-à-dire, si $conv\{x \in X \mid Bx \leq b\} = \{x \mid Bx \leq b\}$, les points extrêmes de $\{x \mid Bx \leq b\}$ appartiennent alors à X et la borne trouvée par L_δ ne peut pas être de meilleure qualité que la borne $\widetilde{z}$ determinée par la relaxation continue de P.

Les corollaires suivants démontrent que la borne determinée par la relaxation Lagrangienne est au moins d'une qualité égale à la borne trouvée par la relaxation continue de P.

Corollaire 1 *si $conv\{x \in X \mid Bx \leq b\} = \{x \mid Bx \leq b\}$, alors $\widetilde{z} = L_x(\delta) = L_\delta \leq z$. Dans ce cas, la relaxation Lagrangienne est égale à la borne $\widetilde{z}$.*

Corollaire 2 *si $conv\{x \in X \mid Bx \leq b\} \subset \{x \mid Bx \leq b\}$, alors $\widetilde{z} \leq L_x(\delta) = L_\delta \leq z$. Dans ce cas, la relaxation Lagrangienne peut améliorer la borne $\widetilde{z}$.*

Dans cette thèse, nous utilisons la méthode de sous-gradient adaptée à partir des résultats de Möhring *et al.* (2000) pour la résolution du dual Lagrangien. Nous décrivons en détail la méthode de sous-gradient plus tard dans la section (4.3.2).

4.3 Relaxation Lagrangienne de la formulation directe desagrégée

Dans cette section, nous présentons la relaxation Lagrangienne de la formulation directe renforcée (direct+)(voir chapitre 2, section 2.4.2). Soit $\widetilde{\chi}(\lambda)$ le polyèdre associé à l'espace de solutions considerant la relaxation des contraintes d'intégralité des variables dans la formulation (direct+), une borne inférieure λ_{LP} pour la période λ est determinée par résolution itérative jusqu'à trouver le plus petit λ tel que $\widetilde{\chi}(\lambda)$ soit non-vide. Le principal inconvénient de cette approche est qu'il existe un nombre pseudo-polynomial de variables x_i^t et de contraintes 2.31. Pour une valeur de T grande, la résolution de ce programme linéaire peut en effet entraîner un temps de calcul important.

Considérons maintenant l'expression

$$L_{\delta,\lambda}(x) = \sum_{s=1}^{m}\sum_{\tau=0}^{\lambda-1}\delta_{\tau,s}(\sum_{i=1}^{n}\sum_{\substack{t=0\\ \text{mod } \lambda=\tau}}^{T} b_i^s x_i^t) - \sum_{s=1}^{m}\sum_{\tau=0}^{\lambda-1}\delta_{\tau,s}B_s.$$

et le programme linéaire

$$\min\{L_{\delta,\lambda}(x) \mid x \in \hat{\chi}(\lambda)\} \tag{4.1}$$

où $\delta_{\tau,s} \geq 0 \ \forall \ [0, \lambda-1]$, et $\hat{\chi}(\lambda)$ représente le polyèdre définit par les contraintes 2.27, 2.31 et 2.34 de la formulation directe renforcée. Le programme linéaire 4.1 avec les multiplicateurs $\delta_{\tau,s}$ est obtenu à partir de la relaxation des contraintes 2.29. Soit $L^*_{\delta,\lambda}$ la valeur de la fonction objectif optimale.

Proposition 1 *$\widetilde{\chi}(\lambda)$ est vide si pour un $\delta \geq 0$, $\hat{\chi}(\lambda)$ est vide ou $L^*_{\delta,\lambda} > 0$.*

Preuve De par la définition des polyèdres $\widetilde{\chi}(\lambda)$ et $\hat{\chi}(\lambda)$, il est evident que $\widetilde{\chi}(\lambda) \subseteq \hat{\chi}(\lambda)$. Si $\widetilde{\chi}(\lambda)$ est vide, alors $\hat{\chi}(\lambda)$ est également vide et si $L^*_{\delta,\lambda}$ est strictement positif pour tout $x \in \widetilde{\chi}(\lambda)$, alors

$$\sum_{s=1}^{m}\sum_{\tau=0}^{\lambda-1}\delta_{\tau,s}(\sum_{i=1}^{n-1}\sum_{\substack{t=0\\ \text{mod } \lambda=\tau}} b_i^s x_i^t) > \sum_{s=1}^{m}\sum_{\tau=0}^{\lambda-1}\delta_{\tau,s}B_s.$$

Comme $\delta \geq 0$, nous avons

$$\sum_{i=1}^{n-1}\sum_{\substack{t=0\\ \text{mod } \lambda=\tau}} b_i^s x_i^t > B_s \text{ pour certains } \tau \in [0, \lambda-1]\,,\ s \in [1, m]$$

alors $\widetilde{\chi}(\lambda)$ est vide. ∎

La proposition 1 permet de définir la borne inférieure λ_{RL} pour la période λ comme le plus petit λ tel que le problème Lagrangien dual L_δ définit par

$$\max_{\delta \geq 0} \min\{L_{\delta,\lambda}(x) \mid x \in \hat{\chi}(\lambda)\} \tag{4.2}$$

n'a pas une solution optimale positive ($L^*_\delta \leq 0$). En plus, le problème L_δ possède la propriété d'intégralité car $Co\{x \in \{0,1\} \cap \widetilde{\chi}(\lambda)\} = \hat{\chi}(\lambda)$, alors $\lambda_{L_\delta} = \lambda_{LP}$.

Malheureusement nos résultats expérimentaux du chapitre 2 ont montré que la borne de la période obtenue de cette manière n'excède pas sur les instances testées la borne triviale λ_{min}. Par contre la borne peut avoir un intérêt pour la durée totale ($Cmax$). Pour l'obtenir, il suffit d'ajouter une $n+1$ ième composante au vecteur x (la tâche fictive finale) et les contraintes de précédence correspondantes (toute tâche i précède $n+1$ avec une latence unitaire et une distance nulle). Soit $\hat{\chi}'(\lambda)$ le polyèdre correspondant. On remplace alors $L_{\delta,\lambda}(x)$ par $L'_{\delta,\lambda}(x) = Cmax + L_{\delta,\lambda}(x)$ avec $Cmax = \sum_{t=0}^{T} tx_{n+1}^t$. Le sous problème Lagrangien est alors donné par

$$\min\{L'_{\delta,\lambda}(x) \mid x \in \hat{\chi}'(\lambda)\} \tag{4.3}$$

Une borne sur le $Cmax$ optimal égale à celle obtenue par la relaxation de la formulation (direct+) est donnée par la dual Lagrangien suivant :

$$\max_{\delta \geq 0} \min\{L'_{\delta,\lambda}(x) \mid x \in \hat{\chi}'(\lambda)\} \tag{4.4}$$

4.3.1 Définition et résolution du sous-problème Lagrangien

Nous supposons qu'une valeur de λ réalisable pour la relaxation de programmation linéaire de la formulation (direct+) est connue.

Pour l'obtention d'une borne sur le $Cmax$, le sous-problème Lagrangien est défini par 4.3. Plus précisément nous avons

$$L'_{\delta,\lambda}(x) = Cmax + L_{\delta,\lambda}(x) = \sum_{t=0}^{T} tx_{n+1}^t + \sum_{s=1}^{m}\sum_{\tau=0}^{\lambda-1} \delta_{\tau,s}(\sum_{i=1}^{n} \sum_{\substack{t=0 \\ t \mod \lambda=\tau}} b_i^s x_i^t) - \sum_{s=1}^{m}\sum_{\tau=0}^{\lambda-1} \delta_{\tau,s} B_s$$

Le terme constant $\sum_{s=1}^{m}\sum_{\tau=0}^{\lambda-1} \delta_{\tau,s} B_s$ peut être exclu de la fonction objectif car il n'intervient pas dans la minimisation.

Si nous définissons un poids $f_{i,t}$ tel que

$$f_{i,t} = \begin{cases} \sum_{s=1}^{m} b_i^s \delta_{\tau,s} & \text{si } i < n+1 \text{ et } t < T \\ t & \text{si } i = n+1 \\ 0 & \text{sinon} \end{cases} \tag{4.5}$$

le sous problème Lagrangien peut être alors récrit comme :

$$\min\{\sum_{i=1}^{n+1}\sum_{t=0}^{T} f_{i,t} x_i^t \mid x \in \hat{\chi}'(\lambda)\} \tag{4.6}$$

Ce problème est un cas particulier du problème d'ordonnancement de projet avec coûts dépendant des dates de début décrit par Möhring *et al.* (2000) qui présentent un algorithme de résolution qui utilise le calcul d'une coupe de capacité minimale dans un graphe orienté. C'est-à-dire un ensemble d'arcs F associé à deux sommets, a et b qui représentent des sommets source et cible respectivement, tels que toute chemin reliant a à b dans le graphe contienne au moins un arc de F. Nous transposons les résultats de Möhring *et al.* (2000) à notre problème.

Soit le graphe orienté $G^{'} = (V^{'}, E^{'})$ défini par :

- **l'ensemble de sommets :** $V^{'} = \{v_{i,t} \mid i = 1...n+1,\ t \in \{0, ..., T+1\}\} \cup \{a, b\}$.
- **l'ensemble d'arcs** $E^{'}$ **:** trois types d'arcs sont définis pour chaque $i = 1, ..., n+1$ et pour chaque $t \in T$. Les arcs d'affectation $(v_{i,t}, v_{i,t+1})$ avec une capacité $f_{i,t} = f(v_{i,t}, v_{i,t+1})$, les arcs temporels $(v_{i,t}, v_{i,t+(\theta_i^j - \lambda\omega_i^j)})$ avec une capacité infinie et les arcs $(a, v_{i,0})$ et $(v_{i,Tmax+1,b})$ également avec une capacité infinie.

La figure 4.1 illustre la structure du graphe $G^{'}$.

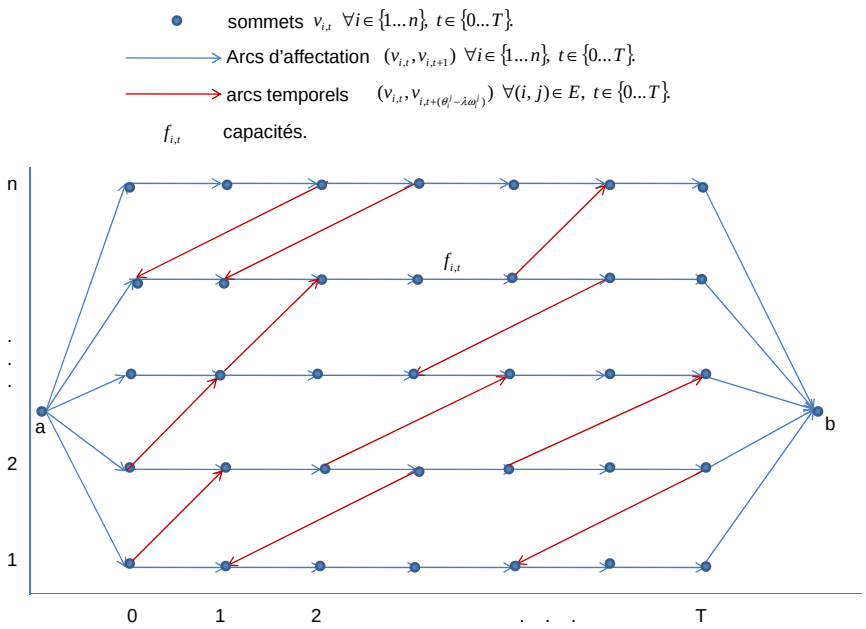


FIGURE 4.1 – Structure du graphe $G^{'}$.

Dans le graphe $G^{'}$, une coupe $a - b$ est une paire ordonnée $(X, \bar{X})$ d'ensembles disjoints telle que $X, \bar{X} \subset V^{'}$ avec $X \cup \bar{X} = G$ et $a \in X,\ b \in \bar{X}$. Un arc (u, v) est un arc sortant de la coupe, si $u \in X$ et $v \in \bar{X}$. La capacité $c(X, \bar{X})$ d'une coupe $X, \bar{X}$ est la somme des capacités des arcs sortants dans la coupe, $c(X, \bar{X}) = \sum_{(u,\bar{u}) \in (X,\bar{X})} C(u, \bar{u})$. Le théorème suivant demontre que le problème Lagrangien 4.3 peut être transformé en un problème de coupe minimale dans le graphe proposé.

Théorème 5 *(Möhring et al. (2000)) Il existe une correspondance une à une entre la coupe minimale $a - b$ et la paire $(X, \bar{X}) \in G$ avec exactement n arcs avant et une solution optimale x pour le probleme Lagrangien 4.3 obtenu par $x_{i,t} = 1$ si $(v_{i,t}, v_{i,t+1})$ est un arc sortant de la coupe $(X, \bar{X})$, et $x_{i,t} = 0$ sinon. La valeur $w(x)$ d'une solution optimale du problème Lagrangien 4.3 est égale à la capacité $c(X, \bar{X})$ de la coupe minimale $(X, \bar{X})$ de G'.*

Preuve Voir Möhring *et al.* (2000). ∎

Ainsi pour résoudre le sous-problème Lagrangien $L'_{\delta,\lambda}(x)$ on peut utiliser algorithme de flot, comme par exemple l'algorithme de Ford et Fulkerson.

4.3.2 Résolution du problème dual Lagrangien

Pour calculer un vecteur de multiplicateurs Lagrangiens optimal $\delta = \delta_{\tau,s}$, $\tau \in [0, \lambda - 1]$, $s \in [1, m]$, nous utilisons la méthode de sous-gradient en suivant l'approche de Möhring *et al.* (2000). Soit δ^k le vecteur de multiplicateurs Lagrangiens dans la *k-ième* itération, le vecteur de multiplicateurs Lagrangiens dans la *k+1-ème* itération est défini par :

$$\delta^{k+1} = \left[\delta^k + \gamma^k g^k\right]^+ \tag{4.7}$$

où γ^k est le pas de déplacement et g^k est le sous-gradient dans l'objectif $L_{\delta,\lambda}(x)$ au point δ^k. Ce sous-gradient g^k est défini par :

$$g^k_{s,\tau} = \sum_{i=i}^{n-1} b^s_i \Big(\sum_{\substack{t=0 \\ t \mod \lambda = \tau}} x^k_{i,t} \Big) - B_s \tag{4.8}$$

Nous remarquons que pour chaque itération de la méthode de sous-gradient, un ordonnancement qui respecte les contraintes de précédence, mais qui par contre ne respecte pas nécessairement les contraintes de ressources est calculé. Dans le processus du sous-gradient, l'objectif est de réduire les violations des contraintes de ressources. Pour calculer le pas de déplacement γ^k, nous suivons l'approche de Möhring *et al.* (2000).

$$\gamma^k = \gamma(L^* - L'_{\delta,\lambda}(x^k)/\|g^k\|^2 \tag{4.9}$$

où L^* est une borne supérieure, γ est le pas escalier qui est reglé en fonction de l'amélioration de la borne inférieure. Si la borne inférieure ne présente pas une augmentation significative dans trois itérations, γ est affecté par un facteur de 0.8. Ensuite, s'il n'existe pas une amélioration dans les 15 itérations suivantes, l'algorithme s'arrête. Pour améliorer la convergence, nous utilisons la technique proposée par Camerini *et al.* (1975) où le calcul du vecteur de multiplicateurs (expression 4.7) est remplacé par l'expression

$$\delta^{k+1} = [\delta^k + \gamma^k d^k]^+ \tag{4.10}$$

où $d^k = g^k$ si $k = 0$ et $d^k = g^k + \beta g^{k-1}$, sinon. $\beta \in [0, 1]$ est le pas qui dépend de l'angle entre deux sous-gradients successifs.

La borne inférieure est trouvée par la résolution successive des sous problèmes Lagrangiens $L'_{\delta^k,\lambda}(x^k)$ jusqu'à ce qu'un nombre d'itérations maximal soit atteint où que la borne n'évolue plus pendant plusieurs itérations.

4.4 Heuristique et résultats expérimentaux

Les bornes inférieures du makespan obtenues par l'algorithme de résolution du dual Lagrangien présenté dans la section précédente sur les instances industrielles et modifiées s'approchent de celles obtenues par la résolution de la relaxation de la formulation (direct+) mais sans toutefois parvenir à améliorer les temps d'exécution (voir colonnes λ_{RL}, $Cmax_{RL}$ et CPU dans les tables 4.3 et 4.4). Par contre, contrairement à la solution du programme linéaire, le vecteur x obtenu par l'algorithme du dual Lagrangien est entièr $(0-1)$. Nous proposons de profiter de cette caractéristique pour calculer une solution réalisable approchée et une borne supérieure de la période optimale.

Pour cela, nous reprenons la formulation décomposée structurée proposée par Eichenberger and Davidson (1997) (voir chapitre 2, section 2.4.1). Dans cette formulation, nous proposons de remplacer le vecteur k_i pour un vecteur $k_i^{'}$ $\forall i = 1, ..., n$ à partir de la transformation 2.33 qui permet d'exprimer les variables de décision k_i de la formulation décomposée proposée par Eichenberger and Davidson (1997) en fonction des variables x_i^t. Dans ce cas, les valeurs de x_i^t (entières) sont déterminés à partir de la meilleure coupe minimale obtenue lors de la résolution du problème Lagrangien. Une borne supérieure de la période peut être obtenue à partir de la résolution de la formulation décomposée structurée, en remplaçant les variables de décision k_i par le nouveau vecteur $k_i^{'}$. La borne supérieure pour la période λ, noté par λ^{sup}, est trouvée à partir d'un processus itératif qui commence avec la valeur de $\lambda = \lambda_{LP}$. Si, avec cette valeur de λ il n'existe pas de solution admissible, la valeur de λ est incrémentée $(\lambda = \lambda + 1)$. La plus petite période réalisable est alors une borne supérieure de la période optimale.

Une borne supérieure pour le $Cmax$ est définie comme la valeur obtenue une fois que le programme linéaire admet une solution réalisable pour une valeur de λ donnée. C'est une borne supérieure valable si $\lambda \geq \lambda^{sup}$ que nous notons $Cmax^{sup}(\lambda^{sup})$.

L'algorithme de résolution des sous-problèmes et du dual Lagrangien a été codé en C++. La borne supérieure est obtenue avec Cplex 11.2. Les expérimentations ont été exécutées sur un Intel(R) Core(TM)2 Duo CPU E4400 @ 2.00GHz 1.99 GHz RAM. Nous avons utilisé les deux groupes d'instances décrits dans le premier chapitre et déjà utilisées dans le chapitre 2.

La table 4.1 montre un récapitulatif des résultats obtenus pour les bornes supérieures de la période λ et du $Cmax$. Dans le cas des instances industrielles, le remplacement du vecteur k_i dans la formulation (décomp+) permet de trouver des bornes supérieures pour la période λ telles que λ_{sup} est égale à λ_{opt} dans 78% des cas (28 instances). L'écart moyen est de 0.55%. Par rapport à la borne supérieure pour le $Cmax$, nous comparons uniquement les instances telles que $\lambda_{sup} = \lambda_{opt}$ et nous obtenons l'optimum dans 21 cas sur 28. Rappelons que $Cmax$ est un objectif secondaire et que la minimisation du $Cmax$ est effectuée une fois que le programme linéaire admet une solution. La table 4.3 montre en détail les solutions pour chaque instance. La plupart des instances ont été résolues en quelques centièmes de secondes. Toutefois à ces temps de calcul doivent être ajoutés les temps de résolution du dual Lagrangien donnés dans la table 4.3 qui restent très élévés et pourraient certainement être améliorés par une meilleure implémentation des algorithmes de flot maximal et de sous-gradient.

Nombre d'instances résolues	36/36
Test avec $\lambda = \lambda^{opt}$	28/36
Ecart maximal $\lambda^{opt} - \lambda$	8
Ecart moyen $\lambda^{opt} - \lambda$	0.55
Test avec $Cmax_{sup} = Cmax^{opt}$	21/28
Ecart maximal $Cmax^{opt} - Cmax_{sup}$	5
Ecart moyen $Cmax^{opt} - Cmax_{sup}$	0.68

TABLE 4.1 – Récapitulatif des bornes supérieures obtenues pour la période λ et pour le $Cmax$. Instances industrielles

La table 4.2 montre un récapitulatif des résultats obtenus en termes de bornes supérieures de la période λ et du $Cmax$ pour l'ensemble d'instances modifiées. Dans ce cas, la génération du vecteur $k_i^{'}$ est très particulier. Pour 76% d'instances le vecteur $k_i^{'}$ généré est nul. Ce vecteur $k_i^{'}$ ne permet pas de trouver une solution réalisable. Dans ce cas, nous appliquons une transformation du vecteur $k_i^{'}$. Cette transformation consiste à modifier les éléments nuls du vecteur jusqu'à que le programme linéaire trouve une solution. Nous commençons par la valeur de $k_{n+1}^{'}$. Nous remplaçons la valeur $k_{n+1}^{'} = 0$ par $k_{n+1}^{'} = 1$. Ensuite, nous résolvons le programme linéaire avec ce nouveau vecteur. Si le programme linéaire ne trouve pas une solution, nous commençons à modifier les autres valeurs $k_i^{'}$ une par une, jusqu'à ce que le programme linéaire admette une solution. Nous remarquons que la transformation du vecteur $k_i^{'}$ permet de trouver pour toutes les instances une solution réalisable avec une unique modification du vecteur (en remplaçant la valeur de l'element $k_{n+1}^{'}$).

Nous obtenons avec cette méthode les résultats suivants. Pour 50% des instances (14 instances), la valeur λ_{sup} est égale à la valeur λ_{opt}. Un écart moyen de 0.65 et un écart maximal de 3 sont trouvés. En ce qui concerne la valeur $Cmax_{sup}$, nous observons bien que lorsque la valeur λ_{sup} est supérieure a la valeur λ_{opt}, la valeur de $Cmax_{sup}$ peut devenir plus petite que la valeur $Cmax_{opt}$. Lorsque $\lambda_{sup} = \lambda_{opt}$, nous obtenons le makespan optimal dans 12 cas sur 14. La table 4.4 montre en détail les solutions pour chaque instance. La plupart des instances ont été résolues en quelques centièmes de secondes sauf les instances *gsm-st231.18* et *gsm-st231.25* dont le temps de résolution est de l'ordre de la minute. Nous pouvons faire la même remarque que pour les instances industrielles en ce qui concerne les temps de calcul du dual Lagrangien donnés dans la table 4.4.

Nombre d'instances résolues	33/36
Test avec $\lambda_{sup} = \lambda^{opt}$	14/28
Ecart maximal $\lambda^{opt} - \lambda_{sup}$	3
Ecart moyen $\lambda^{opt} - \lambda_{sup}$	0.64
Test avec $Cmax_{sup} = Cmax^{opt}$	12/14
Ecart maximal $Cmax^{opt} - Cmax_{sup}$	5
Ecart moyen $Cmax^{opt} - Cmax_{sup}$	1.21

TABLE 4.2 – Récapitulatif des bornes supérieure obtenues pour la période λ et pour le $Cmax$. Instances modifiées

TABLE 4.3 – Bornes inférieures pour la période λ et pour le $Cmax$. Instances industrielles

Instance	λ_{RL}	$Cmax_{RL}$	CPU_s	λ_{LP}	λ_{opt}
adpcm-st231.1	21	29	4266	21	21
adpcm-st231.2	38	42	7285	38	40
gsm-st231.1	24	33	352	24	24
gsm-st231.2	26	31	2931	26	26
gsm-st231.5	11	15	1327	11	11
gsm-st231.6	7	11	4	7	7
gsm-st231.7	11	15	5	11	11
gsm-st231.8	8	8	5	8	8
gsm-st231.9	28	28	3578	28	28
gsm-st231.10	4	4	2	4	4
gsm-st231.11	20	21	8	20	20
gsm-st231.12	8	8	826	8	8
gsm-st231.13	19	25	407	19	19
gsm-st231.14	10	13	40	10	10
gsm-st231.15	8	8	9	8	8
gsm-st231.16	16	17	33317	16	16
gsm-st231.17	9	15	570	9	9
gsm-st231.18	53	53	142000	53	-
gsm-st231.19	8	8	13	8	8
gsm-st231.20	6	10	29	6	10
gsm-st231.21	18	22	5	18	18
gsm-st231.22	18	18	22	18	18
gsm-st231.25	16	25	86000	16	16
gsm-st231.29	11	15	421	11	11
gsm-st231.30	7	11	194	7	7
gsm-st231.31	11	15	506	11	11
gsm-st231.32	15	15	14	15	15
gsm-st231.33	15	17	2155	15	15
gsm-st231.34	4	5	3	4	4
gsm-st231.35	6	8	18	6	6
gsm-st231.36	10	10	52	10	10
gsm-st231.39	8	12	47	8	8
gsm-st231.40	10	10	21	10	10
gsm-st231.41	18	18	740	18	18
gsm-st231.42	6	10	13	6	6
gsm-st231.43	8	10	44	8	8

TABLE 4.4 – Bornes inférieures pour la période λ et pour le $Cmax$. Instances modifiées

Instance	λ_{RL}	$Cmax_{RL}$	CPU_s	λ_{LP}	λ_{opt}
adpcm-st231.1	52	52	57225	52	-
adpcm-st231.2	82	82	87342	82	-
gsm-st231.1	24	33	2526	24	25
gsm-st231.2	59	59	7252	59	-
gsm-st231.5	26	26	18506	26	36
gsm-st231.6	17	17	3438	17	27
gsm-st231.7	28	28	1897	28	41
gsm-st231.8	9	9	16	9	12
gsm-st231.9	28	29	39	28	32
gsm-st231.10	6	6	8	6	8
gsm-st231.11	20	21	544	20	24
gsm-st231.12	10	10	22	10	13
gsm-st231.13	27	27	2617	27	43
gsm-st231.14	20	20	10146	20	33
gsm-st231.15	9	9	301	9	12
gsm-st231.16	38	38	15666	38	-
gsm-st231.17	23	23	603	23	33
gsm-st231.18	120	120	205000	120	-
gsm-st231.19	12	12	33	12	15
gsm-st231.20	13	13	129	13	20
gsm-st231.21	20	23	1135	20	30
gsm-st231.22	18	22	1491	18	29
gsm-st231.25	37	37	110083	37	56(gap=1.75%)
gsm-st231.29	28	28	1396	28	42
gsm-st231.30	16	16	345	16	25
gsm-st231.31	26	26	1775	26	39
gsm-st231.32	21	21	11003	21	30
gsm-st231.33	33	33	3467	33	52(gap=8%)
gsm-st231.34	6	6	8	6	7
gsm-st231.35	11	11	19	11	14
gsm-st231.36	18	18	37	18	24
gsm-st231.39	15	15	42	15	21
gsm-st231.40	12	12	29	12	17
gsm-st231.41	34	34	1060	34	-
gsm-st231.42	14	14	20	14	18
gsm-st231.43	15	15	78	15	20

TABLE 4.5 – Bornes supérieures pour la période λ et pour le $Cmax$. Instances industrielles

Instance	λ_{sup}	$Cmax_{sup}$	CPU_s	λ_{opt}	$Cmax_{opt}$
adpcm-st231.1	21	30	17	21	30
adpcm-st231.2	40	42	33	40	42
gsm-st231.1	32	33	0,05	24	33
gsm-st231.2	26	32	46	26	32
gsm-st231.5	11	18	38	11	17
gsm-st231.6	7	15	0,05	7	13
gsm-st231.7	11	18	0,05	11	17
gsm-st231.8	8	8	0,05	8	8
gsm-st231.9	28	28	0,05	28	28
gsm-st231.10	4	4	0,05	4	4
gsm-st231.11	21	21	0,05	20	21
gsm-st231.12	8	8	0,05	8	8
gsm-st231.13	19	25	0,05	19	25
gsm-st231.14	10	13	0,05	10	13
gsm-st231.15	8	8	0,05	8	8
gsm-st231.16	16	21	25	16	20
gsm-st231.17	11	17	0,05	9	16
gsm-st231.18	53	53	72	-	-
gsm-st231.19	8	9	0,05	8	9
gsm-st231.20	6	10	0,05	6	10
gsm-st231.21	18	22	0,05	18	22
gsm-st231.22	21	22	0,05	18	22
gsm-st231.25	16	25	0,05	16	25
gsm-st231.29	11	18	0,05	11	17
gsm-st231.30	8	14	0,05	7	13
gsm-st231.31	11	18	0,05	11	17
gsm-st231.32	15	15	0,05	15	15
gsm-st231.33	15	26	39	15	21
gsm-st231.34	5	6	0,05	4	5
gsm-st231.35	6	11	0,05	6	11
gsm-st231.36	10	20	0,05	10	15
gsm-st231.39	9	17	0,05	8	16
gsm-st231.40	10	10	0,05	10	10
gsm-st231.41	21	28	0,05	18	24
gsm-st231.42	6	10	0,05	6	10
gsm-st231.43	8	15	0,05	8	14

TABLE 4.6 – Bornes supérieures pour la période λ et pour le $Cmax$. Instances modifiées

Instance	λ_{sup}	$Cmax_{sup}$	CPU_s	λ_{opt}	$Cmax_{opt}$
adpcm-st231.1	-	-	-	-	-
adpcm-st231.2	-	-	-	-	-
gsm-st231.1	28	39	38	25	42
gsm-st231.2	-	-	-	-	-
gsm-st231.5	37	37	38	36	46
gsm-st231.6	27	27	0,05	27	27
gsm-st231.7	42	42	0,05	41	45
gsm-st231.8	12	12	0,05	12	12
gsm-st231.9	34	34	0,05	32	35
gsm-st231.10	8	8	0,05	8	8
gsm-st231.11	24	24	0,05	24	24
gsm-st231.12	13	13	0,05	13	13
gsm-st231.13	44	48	0,05	43	48
gsm-st231.14	34	34	0,05	33	45
gsm-st231.15	12	17	0,05	12	12
gsm-st231.16	62	62	25	-	-
gsm-st231.17	33	33	0,05	33	33
gsm-st231.18	121	121	78	-	-
gsm-st231.19	15	15	0,05	15	15
gsm-st231.20	21	21	0,05	20	27
gsm-st231.21	30	39	0,05	30	30
gsm-st231.22	29	29	0,05	29	29
gsm-st231.25	56	56	62	56(gap=1.75%)	56
gsm-st231.29	42	42	0,05	42	42
gsm-st231.30	25	25	0,05	25	25
gsm-st231.31	39	39	0,05	39	39
gsm-st231.32	30	30	0,05	30	30
gsm-st231.33	51	51	39	52(gap=8%)	50
gsm-st231.34	8	8	0,05	7	7
gsm-st231.35	15	15	0,05	14	16
gsm-st231.36	25	25	0,05	24	28
gsm-st231.39	22	22	0,05	21	25
gsm-st231.40	18	18	0,05	17	21
gsm-st231.41	52	52	0,05	-	-
gsm-st231.42	19	19	0,05	18	26
gsm-st231.43	22	22	0,05	20	25

4.5 Conclusion

L'objectif de ce chapitre a été de proposer une alternative à la programmation linéaire pour calculer des bornes inférieures pour $Cmax$. Nous avons présenté ainsi une méthode fondée sur la relaxation Lagrangienne obtenue à partir de la dualisation de la contrainte de ressources dans la formulation directe renforcée proposée par Dinechin (2004). Nous avons utilisé les résultats de Möhring *et al.* (2000) pour transformer le problème Lagrangien en un problème de coupe minimale dans un graphe orienté. Pour la résolution du problème dual Lagrangien, nous avons utilisé la méthode de sous-gradient. Par rapport aux bornes inférieures, les résultats expérimentaux ont montré que les bornes obtenues s'approchaient des bornes de la programmation linéaire mais avec des temps de calculs trop importants.

Toutefois nous avons profité de l'intégralité des solutions obtenues pour proposer une heuristique afin de calculer des bornes supérieures de la période et du $Cmax$. Nous attribuons ainsi aux tâches les numéros de période d'exécution k_i produits par la relaxation et nous résolvons le problème réduit aux dates de début dans la période τ_i par la formulation décomposée (Eichenberger and Davidson, 1997) pour trouver la plus petite période réalisable. Nous avons trouvé ainsi des bornes supérieures λ_{sup} égales aux périodes λ_{opt} pour presque toutes les instances industrielles. Dans le cas des instances modifiées, la borne λ_{sup} est égale à λ_{opt} pour la moitié des instances.

Dans le chapitre suivant, nous améliorons cette heuristique avec une méthode différente pour la génération du vecteur k_i.

Chapitre 5

Heuristique hybride pour le RCMSP

Dans ce chapitre nous présentons une heuristique hybride pour le problème du RCMSP. Cette approche est fondée sur le principe du pipeline logiciel décomposé (DSP). En suivant le principe de l'heuristique proposée au chapitre précédent, les numéros de période des tâches sont calculés avec l'heuristique DSP alors que les dates de début dans la période sont calculées parla formulation PLNE décomposée présentée dans le deuxième chapitre. Le but est de proposer une nouvelle approche pour la recherche de solutions réalisables tout en minimisant la période λ. Les résultats obtenus avec cette approche montrent des avantages en ce qui concerne les temps d'exécution et des bonnes performances par rapport aux solutions optimales. Ce chapitre a fait l'objet d'une soumission dans une revue internationale (Ayala et al., 2011).

5.1 Introduction

Dans les chapitres précédents, nous avons montré l'usage des différentes formulations PLNE pour résoudre le problème du RCMSP. Bien que ces formulations exactes permettent d'obtenir des solutions optimales, la performance de ces formulations est fortement liée au nombre d'opérations. La détermination d'une solution exacte est, pour les instances de grande taille, très coûteuse en temps de calcul. Nous pouvons donc essayer de résoudre le problème avec des algorithmes efficaces pour la recherche d'une "bonne" solution. Ces algorithmes proposent des solutions généralement non optimales mais qui présentent des écarts acceptables par rapport aux solutions optimales en un temps de calcul raisonnable. Dans ce cas, on parle donc de méthodes heuristiques. Différentes méthodes heuristiques ont été développées pour résoudre le RCMSP. Dans la section (1.3.2) chapitre 1, nous avons présenté les différentes approches heuristiques pour résoudre le RCMSP et nous en avons proposé une dans le chapitre précédent. Nous proposons dans ce chapitre d'utiliser l'approche Pipeline logiciel décomposée (DSP) pour générer une nouvelle approche pour résoudre le RCMSP. Cette nouvelle approche est de type hybride, c'est-à-dire, qu'elle combine des méthodes heuristiques et des méthodes exactes. Les approches hybrides ont montré de bonnes performances dans la résolution du RCMSP. Un exemple d'une méthode hybride pour résoudre le RCMSP est présenté dans de Dinechin (2007). Dans ce travail, une heuristique de recherche à grand voisinage (LNS) a été présentée pour résoudre des formulations à variables entières indexées de temps. Les résultats obtenus montrent que l'algorithme (LNS) est très efficace pour trouver une solution dans le période $\lambda - 1$ à partir d'une période λ donnée. Dans le chapitre précédent, nous avons proposé une méthode hybride où le numéro de

période des tâches est calculé par la relaxation lagrangienne (et éventuellement ajusté) alors que la date de début dans la période est calculée au moyen de la formulation de PLNE décomposée présentée dans le chapitre 2.
La nouvelle approche hybride présentée dans ce chapitre est le résultat du travail réalisé en collaboration avec Abir Benabid (LIP6, Université Pierre et Marie Curie) et Claire Hanen (Université Paris Ouest Nanterre la Défense). Abir Benabid a effectué sa thèse sous la direction de Claire Hanen et le titre de son travail était *"Etude du RCPSP Cyclique"*. Cette thèse se focalise sur l'approche DSP pour traiter le problème du RCMSP. Bien que le principe du pipeline logiciel décomposé et ses éléments principaux soient présentés dans ce chapitre, nous renvoyons le lecteur à Benabid (2011) pour plus de détails. Dans ce travail, un état de l'art sur le (DSP) est présenté ainsi qu'une description en détail des différents algorithmes utilisés dans les différentes phases de résolution du problème.
Ce chapitre est structuré de la manière suivante : le principe de pipeline logiciel décomposé (DSP) est d'abord présenté. Ensuite, nous proposons notre nouvelle approche hybride pour résoudre le RCMSP est décrite. Des résultats expérimentaux pour tester l'efficacité de la nouvelle approche sont enfin présentés.

5.2 Pipeline logiciel décomposé (DSP)

Le pipeline logiciel décomposé *"Decomposed software pipelining"* ou (DSP) a été proposé par Wang *et al.* (1994). Le DSP est une technique qui consiste en la transformation du vecteur colonne d'instructions (σ_i) en une matrice à deux dimensions. Les lignes de cette matrice représentent les cycles où les opérations [1] sont exécutées ($\sigma_i \mod \lambda$, c'est-à-dire la date de début τ_i dans la période) et les colonnes indiquent l'itération dans la boucle d'origine (i.e. le numéro de la période k_i). L'approche de pipeline logiciel consiste en la décomposition du problème original en deux sous-problèmes : le premier sous problème consiste en la transformation du graphe cyclique en un graphe acyclique sans prendre en compte les contraintes de ressources. Le deuxième sous problème revient à déterminer un ordonnancement acyclique sous contraintes de ressources. De façon générale, le premier sous-problème est appelé "phase de décalage des instructions" du DSP. Le deuxième sous problème est connu sous le nom de "phase de compaction de boucle". La formulation (que nous avons appelée "décomposée") (Eichenberger and Davidson, 1997) n'est autre que la traduction en PLNE de cette décomposition, d'où l'intérêt d'une approche "hybride".

5.2.1 Phase de décalage des instructions

Le décalage des instructions est une technique d'optimisation qui dans le domaine de la synthèse de circuits est connu sous le nom de *retiming*. La définition de retiming a été proposé par Leiserson and Saxe (1991). Le retiming est basée sur le déplacement des éléments de synchronisation ou registres afin de modifier les propriétés du circuit (poids des arcs) sans changer son comportement fonctionnel (Huard, 2008). La notion de retiming a été utilisée dans le pipeline logiciel initialement par Calland *et al.* (1998) pour la recherche d'ordonnancements périodiques sous contraintes de ressources. Dans le cadre d'ordonnancements périodiques, la notion de re-

1. Une opération est considérée comme une instance d'une instruction dans un programme.

timing est liée à l'idée de déplacement des instructions entre différentes itérations. Nous allons présenter toute d'abord les notions de base correspondantes au circuit retiming.

Circuit retiming

Définir un retiming sur un graphe $G(O, E, \theta, \omega)$ consiste à assigner à chaque sommet ($1 \leq i \leq n \in O$), une valeur ρ_i qui représente le décalage ou la quantité de temps, dont l'exécution de l'opération i peut être retardée par rapport à l'exécution de l'ensemble total des opérations. Le retiming est obtenue à partir d'une transformation qui affecte les arcs du graphe $G(O, E, \theta, \omega)$. Un vecteur retiming sur un graphe $G(O, E, \theta, \omega)$ est défini par une fonction qui affecte les poids des arcs du graphe de la manière suivante :

$$\forall\ (i,j) \in E, \quad \rho : (i,j) \to \mathbb{Z},\ \omega^{\rho}_{i,j} = \rho(\omega^{j}_{i}) = \omega^{j}_{i} + \rho_j - \rho_i$$

Pour conserver une signification physique en tant que circuit au graphe, une définition de retiming légal est définie par Leiserson and Saxe (1991) comme suit :

Définition 6 *Un retiming légal est un retiming tel que*

$$\forall\ (i,j) \in E, \quad \rho : (i,j) \to \mathbb{Z},\ \omega^{\rho}_{i,j} = \rho(\omega^{j}_{i}) = \omega^{j}_{i} + \rho_j - \rho_i \geq 0$$

Dans le cadre du pipeline logiciel, cette définition de retiming légal est très importante. Un retiming ρ_i négatif indique que l'opération i a besoin d'un résultat d'une autre opération j qui sera exécutée dans une itération future, ce qui n'est pas réalisable.

La table 5.1 montre le vecteur retiming ρ_i calculé pour l'exemple donné dans la figure 1.3 du chapitre 1.

Opérations	O_1	O_2	O_3	O_4	O_5	O_6	O_7
ρ_i	1	1	1	1	0	1	2

TABLE 5.1 – Vecteur retiming légal ρ_i pour l'exemple 1.3.

Ces valeurs de retiming correspondent à des retards qui sont appliqués à l'exécution des différentes itérations des opérations. Appliquer un retiming ρ au graphe $G = (O, E, \theta, \omega)$ signifie que la $q^{ième}$ itération de l'opération i ($< i, q >$) est exécutée maintenant à l'itération $< q + \rho_i >$. L'application d'un retiming ρ à une opération i implique aussi un changement dans les contraintes de précédences. Si la $q^{ième}$ itération de la opération i précède la $< q + \omega^{j}_{i} >$ *ième* itération de la opération j alors la $< q + \rho_i >$ *ième* itération de l'opération i précède la $< q + \omega^{j}_{i} + \rho_j >$ *ième* itération de l'opération j.

Une caractéristique des circuits intégrés est la période d'horloge. D'après Huard (2008), "la période d'horloge d'un circuit définit le temps durant lequel les sorties des registres doivent être stables afin de garantir la stabilité de toutes les entrées. La période d'horloge, notée par $\phi(G)$, est calculée comme la somme maximale des délais des unités fonctionnelles présentes sur un chemin sans registres du circuit". Dans le cadre des circuits intégrés, le circuit retiming a été utilisé pour déplacer les registres dans un circuit synchrone avec le but de diminuer le temps de calcul maximal entre deux registres. Réduire cette distance maximale entre les registres, permet

également de réduire la période d'horloge sans augmenter le temps nécessaire de calcul. Ceci permet d'obtenir une augmentation du débit du circuit (Benabid, 2011). Calland *et al.* (1998) ont proposé un algorithme pour le calcul du retiming qui minimise la longueur du chemin critique du graphe G_ρ (graphe G avec des arcs ponderés par le retiming). Ils ont montré que minimiser le chemin critique est équivalent à minimiser la période d'horloge pour le décalage d'instructions.

Différentes méthodes ont été proposées pour la construction des retimings utilisés dans la phase du décalage des instructions.

Algorithme de Gasparoni et Schwiegelsohn

L'algorithme considère uniquement les contraintes de précédence. Considérons le graphe $G = (O, E, \theta, \omega)$. Soit λ un nombre entier non-négatif, G défini donc un graphe avec des arcs pondérés $G'_\lambda = (O', E', \omega')$ comme suit :

- Sommets de G'_λ : un nouveau sommet s est ajouté à O. ($O' = O \cup \{s\}$).
- Arcs de G'_λ : un arc de s vers touts les autres sommets est ajouté à l'ensemble de précédences E : ($E' = E \cup (\{s\} \times O)$).
- Pondération des arcs de G'_λ : soit e une précédence entre les opérations $(i, j) \in O$. $\omega'(e) = 0$ si $e \in E' \setminus E$ et $\omega'(e) = \theta(e) - \lambda\omega(e)$ si $e = (i, j) \in E$.

La période λ est un intervalle d'initiation ou période valide si et seulement si G'_λ ne contient pas un circuit de poids positif. Si G'_λ ne contient pas un circuit de poids positif et si σ_s^i dénote la longueur du circuit critique de s à tout nœud i dans G'_λ, un ordonnancement valide est alors définie par $\sigma_i^q = \sigma_s^i + \lambda q$. Nous savons que déterminer si λ est un intervalle d'initiation ou période valide est un problème facile à résoudre.

Dans le cas où λ est un intervalle d'initiation valide, il est alors possible de trouver un ordonnancement avec l'algorithme de Bellman-Ford. Comme dans les chapitres précédentes, l'intervalle d'initiation optimale λ_{prec} est défini comme le plus petit λ entier non-négatif tel que G'_λ ne contient pas des circuits de longueur positive. Par ailleurs, $\lambda_{prec} = 0$ si G est acyclique et $\lambda_{prec} = \max_C\{\lceil \frac{\theta(C)}{\omega(C)} \rceil\}$ où C sont les circuits de G sinon. Une recherche binaire articulée avec l'algorithme de Bellman-Ford permet le calcul de λ_{prec} dans un temps polynomial (voir chapitre 1).
Nous notons $\sigma^\infty(i, 0)$ à l'ordonnancement obtenu à partir de l'algorithme décrit ci-dessus. Un vecteur retiming ρ_i^{GS} est proposé comme :

$$\rho_i^{GS} = \lfloor \frac{\sigma^\infty(i, 0)}{\lambda_{prec}} \rfloor, \ \forall\ i = 1, ..., n$$

Minimisation du chemin critique

Soit G_ρ le graphe retiming de G, le problème consiste à déterminer un retiming qui permet de minimiser le plus long chemin de poids nul de G_ρ. Ce problème est équivalent à minimiser la période d'horloge pour les circuits intégrés (Calland *et al.*, 1998).
Soit $i \overset{P}{\rightsquigarrow} j$ un chemin P du nœud i au nœud j dans un graphe G. Soit $\omega(P) = \sum_{e \in P} \omega(e)$ la somme des précédences des poids des arcs de P. Soit $\theta(P) = \sum_{j \in P} \theta(j)$ la somme des latences des nœuds

de P. Nous définissons deux valeurs : D et $\triangle$ comme suit :

$$D(i,j) = \min\{\omega(P) : i \overset{P}{\rightsquigarrow} j\}$$

,

$$\triangle\ (i,j) = \max\{\theta(P) : i \overset{P}{\rightsquigarrow} j \text{ et } \omega(P) = D(i,j)\}$$

D et $\triangle$ sont calculés lors de la recherche des plus courts chemins pour toute paire de nœuds du graphe G, où les arcs $i \overset{e}{\rightsquigarrow} j$ sont pondérées par le couple $(\omega(e), \theta(i))$. Finalement, nous définissons

$$\phi(G) = \max\{\theta(P) : P \text{ chemin de } G, \omega(P) = 0\}$$

$\phi(G)$ représente la longueur du chemin critique avec des poids nuls dans le graphe G. $\phi(G)$ correspond à la période d'horloge. Le théorème suivant a été proposé par Leiserson and Saxe (1991). Ce théorème est la base pour définir l'algorithme 3 qui permet de trouver un retiming tel que la période d'horloge du graphe G_ρ est minimale.

Théorème 6 *Soit le graphe $G = (O, E, \theta, \omega)$, λ un nombre réel positif et ρ une fonction telle que $\phi : O \to \mathbb{Z}$. Alors ρ est un retiming légal du graphe G tel que $\phi(G_\rho) \leq \lambda$ si et seulement si :*

1. *$\rho_i - \rho_j \leq \omega(e)$ pour tout arc $i \overset{e}{\rightsquigarrow} j \in G$*
2. *$\rho_i - \rho_j \leq D(i,j) - 1$ pour toute paire de sommets $i, j \in O$ tel que $\triangle\ (i,j) \geq \lambda$*

Preuve Voir Leiserson and Saxe (1991). ∎

Algorithme 3: Minimisation du chemin critique

1 Calculer D et $\triangle$
2 Ordonnancer $\triangle$ dans un ordre décroissant.
3 Recherche binaire entre les éléments $\triangle\ (i,j)$ pour trouver la période d'horloge réalisable minimum. Pour chaque période d'horloge réalisable possible λ, appliquer l'algorithme de Bellman-Ford pour tester les conditions du théorème 6.
4 Pour la valeur minimale du période d'horloge réalisable trouvée à l'étape 3, définir la valeur du retiming légal ρ_j.

L'algorithme 3 détermine un retiming tout en minimisant la période d'horloge en $\mathcal{O}(\mid O \mid^3 log \mid O \mid)$. Une variation de l'algorithme 3 consiste à remplacer l'algorithme de Bellman-Ford par l'algorithme FEAS (Leiserson and Saxe, 1991). La complexité de cette variante est $\mathcal{O}(\mid O \mid\mid E \mid log \mid O \mid)$.

Minimisation du nombre d'arcs

Un deuxième critère d'optimisation pour l'obtention des valeurs du retiming est la minimisation du nombre d'arcs. Darte and Huard (2000) ont proposé une extension de l'algorithme de recherche de plus courts chemins à coût minimal proposé par Fulkerson. Cette extension est connu comme algorithme des arcs non conformes (Gondran and Minoux (1995), p. 178-185). Soit $G = (O, E, \theta, \omega)$, l'objectif est de trouver un retiming ρ du graphe G, de manière que G'_λ possède

le plus petit nombre d'arcs possible. L'idée est de compter le nombre d'arcs e tel que $\omega^{'} = 0$. Un coût $h_\rho(e)$ est défini pour chaque arc e comme suit :

$$h_\rho(e) = \begin{cases} 1 \text{ si } \omega^{'}(e) = 0 \\ 0 \text{ sinon} \end{cases}$$

Le coût du retiming ρ est défini pour $\sum_{e \in E} h_\rho(e)$, qui répresente le nombre d'arcs avec poids nuls dans le graphe $G^{'}_\lambda$. Le retiming ρ est optimal lorsque $\sum_{e \in E} h_\rho(e)$ est minimal. Les fonctions $f : E \to \mathbb{Z}$ sont des flots définis dans le graphe G de la manière suivante :

$$\sum_{e=(i,j) \in E} f(e) = \sum_{e=(j,i) \in E} f(e), \ \forall i \in O,$$

Pour un retiming légal ρ et un flot non-négatif f, un coût $k(e)$ est défini pour chaque arc $e \in E$:

$$k(e) = \begin{cases} h_\rho(e) & \text{si } f(e) = 0 \\ h_\rho(e) - 1 + f(e)\omega^{'}(e) & \text{sinon} \end{cases}$$

L'objectif maintenant est de minimiser le coût du retiming $h_\rho(e)$ en augmentant le coût du flot $k(e)$ à partir d'un retiming et un flot de coût nul donné. Le retiming optimale correspond alors au retiming qui minimise le nombre d'arcs de distance nulle.

Minimisation du nombre des arcs et minimisation du chemin critique

Huard (2008) ont proposée un algorithme qui combine la minimisation de la période d'horloge et la minimisation du nombre d'arcs de poids nul. La complexité de l'algorithme est $\mathcal{O}(\mid E \mid\mid O \mid^2)$. Le but de l'algorithme est de minimiser le nombre de contraintes pour la compaction de la boucle sans augmenter la longueur du chemin critique. Pour cela, des nouvelles contraintes entre les paires de sommet du graphe sont ajoutées. Ces nouvelles contraintes sont des nouveaux arcs qui représentent des contraintes sur le retiming. Ces arcs n'affectent pas le coût du retiming $h_\rho(e)$. Une modification est alors introduite dans l'indice $k(e)$ (Huard, 2008).

5.2.2 Phase de compaction de la boucle

La phase de compaction de la boucle consiste en la définition d'un ordonnancement non-cyclique σ qui prend en compte les contraintes de ressources et les contraintes de précédence du graphe $G^{'}_\lambda$. L'idée est d'ordonnancer la boucle sans chevaucher les itérations successives entre elles. La phase de compaction de la boucle est généralement réalisée à partir des techniques classiques d'ordonnancements acycliques. Les algorithmes de liste sont souvent utilisés pour trouver un ordonnancement réalisable. Une fois l'ordonnancement acyclique proposé, celui-ci est répété à chaque itération. La méthode hybride que nous proposons utilise la formulation décomposée structurée proposée par Eichenberger (voir chapitre 2, section (2.4.1)) pour la phase de compaction de la boucle. Une des avantages de la méthode est qu'elle permet une résolution optimale de la phase de compaction de la boucle. Cette phase est normalement résolue par des heuristiques ou d'autres méthodes approches. Pour plus de détails sur les méthodes de la littérature

pour résoudre cette phase de compaction de la boucle nous renvoyons le lecteur à Huard (2008); Calland *et al.* (1998); Benabid (2011).

5.3 Formulation hybride pour le RCMSP

Nous allons présenter maintenant la méthode hybride basée sur l'approche DSP.
La construction de la phase de décalage des instructions a été réalisé à partir de la définition de retiming légal. Deux types de retiming ont été considérés. Les retiming obtenus avec l'algorithme de Gasparoni et Schwiegelsohn et les retiming obtenus avec l'algorithme qui combine la minimisation de nombres d'arcs et la minimisation du chemin critique proposé par Huard (2008). La phase de compaction de la boucle est résolue à partir de la formulation décomposée structurée proposée par Eichenberger. Dans ce cas, nous avons remplacé les valeurs des variables k_i par le vecteur retiming obtenu dans la phase du décalage des instructions. Le vecteur retiming est équivalent aux variables k_i de la formulation d'Eichenberger. En effet, nous nous référons à Calland *et al.* (1998), la date de début des opérations est défini comme $\sigma_i^q = \sigma_a(i) + \lambda(q + \rho(i))$ où $\sigma_a(i)$ est l'ordonnancement du graphe acyclique de période $\max(\sigma_a(i)+\theta(i))$. L'ordonnancement modulo est caractérisé par $\sigma_i^k = \sigma_i^0$. L'ordonnancement obtenue par Calland, Darte et Robert peut donc s'écrire comme $\sigma_i = \sigma_a(i) + \rho(i)\lambda$. Nous observons que ρ_i est équivalent aux variables k_i de la formulation d'Eichenberger. En effet, dans cette formulation l'ordonnancement est défini par $\sigma_i = \tau_i + k_i\lambda$. L'algorithme 4 illustre les différentes phases de la méthode hybride proposée.

Algorithme 4: Méthode hybride

1 Calculer un retiming légal ρ pour G
2 Trouver λ un réalisable (méthode itérative) et une solution réalisable du systeme suivant :

$$min \sum_{i=1}^{n} w_i(\sum_{\tau=0}^{\lambda-1} \tau y_i^\tau + \rho_i\lambda)$$

$$\sum_{\tau=0}^{\lambda-1} y_i^\tau = 1, \forall i \in [1, n]$$

$$\sum_{x=\tau}^{\lambda-1} y_x^i + \sum_{x=0}^{(\tau+\theta_i^j-1)mod\lambda} y_x^j + \rho_i - \rho_j \leq \omega_i^j - \lfloor \frac{\tau + \theta_i^j - 1}{\lambda} \rfloor + 1, \forall \tau \in [0, \lambda), \forall (i, j) \in E$$

$$\sum_{i=1}^{n} y_i^\tau b_i^s \leq m_s, \forall s \in \{1, \ldots, m\},\ \forall \tau \in [0, \lambda - 1]$$

$$y_i^\tau \in \{0, 1\}$$

3 Définir un ordonnancement $\sigma_i\ \forall\ i = 1..n$, $\sigma i = \tau_i + \rho_i\lambda$

Cet algorithme utilise une recherche itérative de la valeur de λ à partir d'une valeur fixée $\lambda = \lambda_{min}$, où λ_{min} est la borne theorique minimale (voir chapitre 1 section (1.3.2)). Un des avantages de la méthode hybride (comme dans le chapitre précédent) est la réduction du nombre de variables et de contraintes. En effet, la méthode hybride supprime les variables k_i et la contrainte associée

à leur domaine. En ce qui concerne la fonction objectif, la méthode hybride permet de considérer d'autres critères d'optimisation, comme par exemple la minimisation du nombre de registres.

5.4 Résultats expérimentaux

Dans cette section la performance de la nouvelle méthode hybride est évaluée. Nous avons exécuté l'algorithme 4 en utilisant deux types de retiming. Un retiming généré à partir de l'algorithme de Gasparoni et Schwiegelshon, noté comme (GS), et un autre retiming généré à partir de l'algorithme qui minimise le nombre des arcs et aussi le chemin critique. Cette méthode est notée (HD). Nous présentons d'abord la performance de la méthode hybride sur les deux groupes d'instances présentés dans le chapitre 1. Une comparaison de la performance de la méthode hybride par rapport à différentes méthodes proposées dans la littérature pour la recherche des solutions réalisables est ensuite présentée. Ces méthodes correspondent à l'approche basée sur la relaxation lagrangienne du chapitre précédent, l'approche DSP implémentée par Benabid (2011) et une approche proposée par Lombardi *et al.* (2011) basé sur l'introduction de contraintes de précédences (Precedence constraint posting) pour mesurer d'écarts par rapport à la solution optimale. Dans le cas de travaux de Lombardi *et al.* (2011) la comparaison est faite sur la base de 33 instances pour les instances industrielles et sur la base de 25 instances pour les instances modifiées. Les expérimentations ont été exécutées sur Cplex 11.2 avec un Intel(R) Core(TM)2 Duo CPU E4400 @ 2.00GHz 1.99 GHz RAM.

La table 5.2 montre un récapitulatif des résultats obtenus avec la méthode hybride et les algorithmes GS et HD pour les instances industrielles.

	λ_{HD}	λ_{GS}
Nombre d'instances résolues	36/36	36/36
Test avec $\lambda = \lambda^{opt}$	34/36	32/36
Ecart maximal $\lambda^{opt} - \lambda$	1	3
Ecart moyen $\lambda^{opt} - \lambda$	0.05	0.22

TABLE 5.2 – Récapitulatif des résultats obtenus pour la période λ avec la méthode hybride. Instances industrielles

Nous observons que la méthode hybride a trouvé une solution réalisable pour l'instance *gsm-st231.18*, cette instance n'a pas pu être résolue avec la PLNE. En plus, la solution est optimale car la valeur de λ trouvé avec la méthode hybride est égale à la borne inférieure théorique. La table 5.7 montre en détail les résultats obtenus pour chaque instance ainsi que le temps de calcul. Nous remarquons que les algorithmes implémentés pour le calcul du retiming arrivent aux mêmes résultats en ce qui concerne la valeur de la période λ pour la plupart des instances industrielles. Les résultats montrent cependant une meilleure performance avec l'algorithme HD qui rapporte un écart moyen de 0.05. Nous remarquons que la méthode hybride avec l'algorithme HD est plus rapide que l'approche hybride avec l'algorithme GS. Ces différences se justifient par le fait que l'algorithme HD utilisé dans la phase de décalage des instructions, réduit le nombre de contraintes du problème et permet de calculer plus rapidement une solution optimale.

La table 5.3 montre un récapitulatif des résultats obtenus sur l'ensemble d'instances modifiées.

	λ_{HD}	λ_{GS}
Nombre d'instances résolues	34/36	32/36
Test avec $\lambda = \lambda^{opt}$	21/28	20/28
Ecart maximal $\lambda^{opt} - \lambda$	4	3
Ecart moyen $\lambda^{opt} - \lambda$	0.39	0.42

TABLE 5.3 – Récapitulatif des résultats obtenus pour la période λ avec de la méthode hybride. Instances modifiées

Dans ce cas, la méthode hybride permet la résolution d'un plus grand nombre d'instances que les formulations PLNE. La table 5.8 montre en détail les résultats obtenus pour chaque instance ainsi que le temps de calcul. Dans ce cas, nous remarquons une diminution du temps de calcul par rapport aux temps de calcul obtenus par la résolution de la PLNE. Nous présentons les tableaux 5.4 et 5.5 qui évaluent la performance de la méthode hybride par rapport à d'autres méthodes utilisées pour la recherche de solutions réalisables. Dans ces tableaux nous notons λ_{DSP} la valeur de la période λ obtenue par l'approche DSP proposée par Benabid (2011), λ_{iFLAT} la valeur de la période λ obtenue par l'approche iFLAT proposée par Lombardi *et al.* (2011) et λ_{sup} la valeur des bornes supérieures pour la période λ du chapitre précédent.

	λ_{HD}	λ_{GS}	λ_{DSP}	λ_{iFLAT}	λ_{sup}
Nombre d'instances résolues	36/36	36/36	36/36	33/33	36/36
Test avec $\lambda = \lambda^{opt}$	34/36	32/36	30/36	26/33	28/36
Ecart maximal $\lambda^{opt} - \lambda$	1	3	3	3	8
Ecart moyen $\lambda^{opt} - \lambda$	0.005	0.22	0.25	0.24	0.55

TABLE 5.4 – Comparaison de la méthode hybride avec des autres méthodes pour la recherche des solutions réalisables. Instances industrielles

	λ_{HD}	λ_{GS}	λ_{DSP}	λ_{iFLAT}	λ_{sup}
Nombre d'instances résolues	34/36	32/36	36/36	25/25	33/36
Test avec $\lambda = \lambda^{opt}$	21/28	20/28	20/28	16/25	14/28
Ecart maximal $\lambda^{opt} - \lambda$	4	3	5	3	3
Ecart moyen $\lambda^{opt} - \lambda$	0.39	0.42	0.82	0.64	0.64

TABLE 5.5 – Comparaison de la méthode hybride avec des autres méthodes pour la recherche des solutions réalisables. Instances modifiées

La table 5.9 montre en détail les résultats obtenus pour le groupe d'instances modifiées. Bien qu'avec la méthode hybride nous ayons deux instances critiques dans le groupe d'instances modifiées qui n'ont pas été résolues, la méthode hybride améliore la période approche λ_{DSP} pour deux instances *adpcm-st231.1* et *gsm-st231.1*. Ceci explique le meilleure performance (écart maximal). Un autre aspect important est la choix de la méthode pour obtenir le retiming. Nous observons que l'algorithme pour la génération du retiming a une influence importante pour la résolution du problème. Par exemple les instances *gsm-st231.9* a été résolue à l'optimum uniquement avec l'algorithme HD. L'instance *gsm-st231.17* a trouvé la solution optimale avec

l'algorithme HD, l'algorithme GS et l'algorithme par relaxation lagrangienne. Une comparaison des différents retiming pour la phase de décalage des instructions et l'utilisation d'un algorithme de liste pour résoudre la phase de compaction de la boucle est présenté par Benabid (2011). Par rapport à la méthode iFLAT proposée par Lombardi *et al.* (2011) où les résultats obtenus sont calculés en 1 seconde, la méthode hybride est également très efficace en ce qui concerne le temps de calcul (quelques millièmes de secondes pour la majorité des instances).

En ce qui concerne les temps de calcul de la méthode hybride, la table 5.6 présente la distribution de temps de calcul pour chaque ensemble d'instances. Nous remarquons que les instances industrielles sont pour la plupart résolues en millièmes de secondes. Dans le cas des instances modifiées, le temps de calcul est également dans l'ordre des millièmes de secondes pour la majorité des instances. De plus, un plus grand nombre d'instances sont résolues avec la méthode hybride plutôt qu'avec les formulations PLNE.

		Instances ST200					Instances Modifiées				
		ms	sec	min	$> 1h$	No	ms	sec	min	$> 1h$	No
♯Oper	♯Instances										
10 - 30	16	16	-	-	-	-	16	-	-	-	-
31 - 60	15	15	-	-	-	-	13	-	2	-	-
61 - 100	2	2	-	-	-	-	-	-	1	1	-
> 100	3	2	1	-	-	-	-	-	-	1	2
Total	36	35	1	0	0	0	29	0	3	2	2

TABLE 5.6 – Distribution du temps de calcul avec la méthode hybride pour les deux ensembles d'instances.

TABLE 5.7 – Résultats de la méthode hybride pour les instances industrielles

Instances	HD retiming		GS retiming			
	λ^{hyb}	CPU_s	λ^{hyb}	CPU_s	$\lambda^{hyb}/\lambda^{opt}$	$\lambda^{hyb} - \lambda^{opt}$
adpcm-st231.1	21	0.0007	21	< 0.01s	1	0
adpcm-st231.2	40	0.0012	41	15.0000	1	0
gsm-st231.1	24	0.0001	24	< 0.01s	1	0
gsm-st231.2	26	0.0001	26	< 0.01s	1	0
gsm-st231.5	11	0.0001	11	< 0.01s	1	0
gsm-st231.6	7	0.0000	7	< 0.01s	1	0
gsm-st231.7	11	0.0001	11	< 0.01s	1	0
gsm-st231.8	8	0.0000	8	< 0.01s	1	0
gsm-st231.9	28	0.0001	28	< 0.01s	1	0
gsm-st231.10	4	0.0000	4	< 0.01s	1	0
gsm-st231.11	20	0.0001	20	< 0.01s	1	0
gsm-st231.12	8	0.0001	8	< 0.01s	1	0
gsm-st231.13	19	0.0001	19	< 0.01s	1	0
gsm-st231.14	10	0.0001	10	< 0.01s	1	0
gsm-st231.15	8	0.0000	8	< 0.01s	1	0
gsm-st231.16	16	0.0001	16	< 0.01s	1	0
gsm-st231.17	9	0.0001	12	< 0.01s	1	0
gsm-st231.18	53	45.0000	53	900.0000	1	0
gsm-st231.19	8	0.0000	8	< 0.01s	1	0
gsm-st231.20	6	0.0000	6	< 0.01s	1	0
gsm-st231.21	18	0.0001	18	< 0.01s	1	0
gsm-st231.22	18	0.0001	18	< 0.01s	1	0
gsm-st231.25	16	0.0002	16	< 0.01s	1	0
gsm-st231.29	11	0.0001	11	< 0.01s	1	0
gsm-st231.30	7	0.0000	7	< 0.01s	1	0
gsm-st231.31	11	0.0000	11	< 0.01s	1	0
gsm-st231.32	15	0.0000	15	< 0.01s	1	0
gsm-st231.33	15	0.0000	15	< 0.01s	1	0
gsm-st231.34	4	0.0000	5	< 0.01s	1	0
gsm-st231.35	6	0.0000	6	< 0.01s	1	0
gsm-st231.36	10	0.0000	10	< 0.01s	1	0
gsm-st231.39	9	0.0000	8	< 0.01s	1	0
gsm-st231.40	10	0.0000	10	< 0.01s	1	0
gsm-st231.41	18	0.0000	18	< 0.01s	1	0
gsm-st231.42	6	0.0000	6	< 0.01s	1	0
gsm-st231.43	9	0.0000	11	< 0.01s	1	0

TABLE 5.8 – Résultats de la méthode hybride pour les instances modifiées

Instances	HD retiming		GS retiming			
	λ^{hyb}	CPU_s	λ^{hyb}	CPU_s	$\lambda^{hyb}/\lambda^{opt}$	$\lambda^{hyb} - \lambda^{opt}$
adpcm-st231.1	79	1 day	-	-	-	-
adpcm-st231.2	-	-	-	-	-	-
gsm-st231.1	29	0.0030	28	< 0.01s	1.12	3
gsm-st231.2	93	1 day	-	-	-	-
gsm-st231.5	36	< 0.01s	36	< 0.01s	1	0
gsm-st231.6	27	< 0.01s	27	< 0.01s	1	0
gsm-st231.7	41	< 0.01s	41	< 0.01s	1	0
gsm-st231.8	12	< 0.01s	12	< 0.01s	1	0
gsm-st231.9	32	< 0.01s	34	< 0.01s	1	0
gsm-st231.10	8	< 0.01s	8	< 0.01s	1	0
gsm-st231.11	24	< 0.01s	24	< 0.01s	1	0
gsm-st231.12	13	< 0.01s	13	< 0.01s	1	0
gsm-st231.13	43	< 0.01s	43	5 min	1	0
gsm-st231.14	34	< 0.01s	34	< 0.01s	1.03	1
gsm-st231.15	12	< 0.01s	12	< 0.01s	1	0
gsm-st231.16	59	600.0000	59	20 min	-	-
gsm-st231.17	33	< 0.01s	33	< 0.01s	1	0
gsm-st231.18	-	-	-	-	-	-
gsm-st231.19	15	< 0.01s	15	< 0.01s	1	0
gsm-st231.20	20	< 0.01s	20	< 0.01s	1	0
gsm-st231.21	30	< 0.01s	30	< 0.01s	1	0
gsm-st231.22	29	< 0.01s	29	< 0.01s	1	0
gsm-st231.25	56	1 h	56	32 min	-	-
gsm-st231.29	42	< 0.01s	42	< 0.01s	1	0
gsm-st231.30	25	< 0.01s	25	< 0.01s	1	0
gsm-st231.31	39	< 0.01s	39	< 0.01s	1	0
gsm-st231.32	30	< 0.01s	30	< 0.01s	1	0
gsm-st231.33	52	15min	52	25 min	-	-
gsm-st231.34	8	< 0.01s	8	< 0.01s	1.14	1
gsm-st231.35	16	< 0.01s	16	< 0.01s	1.14	2
gsm-st231.36	29	< 0.01s	29	< 0.01s	1.21	5
gsm-st231.39	23	< 0.01s	23	< 0.01s	1.1	2
gsm-st231.40	17	< 0.01s	17	< 0.01s	1	0
gsm-st231.41	50	< 0.01s	50	2 min	-	-
gsm-st231.42	19	< 0.01s	19	< 0.01s	1.06	1
gsm-st231.43	23	< 0.01s	23	< 0.01s	1.15	3

TABLE 5.9 – Résultats obtenus avec les différents méthodes de la littérature. Instances modifiées

Instance	λ_{HD}	λ_{GS}	λ_{DSP}	λ_{iFLAT}	λ_{sup}	λ_{opt}
adpcm-st231.1	79	-	80	-	-	-
adpcm-st231.2	-	-	139	-	-	-
gsm-st231.1	29	28	30	-	25	25
gsm-st231.2	93	-	93	-	-	-
gsm-st231.5	36	36	36	-	37	36
gsm-st231.6	27	27	27	27	27	27
gsm-st231.7	41	41	41	41	42	41
gsm-st231.8	12	12	12	-	12	12
gsm-st231.9	32	34	32	32	34	32
gsm-st231.10	8	8	8	8	8	8
gsm-st231.11	24	24	24	24	24	24
gsm-st231.12	13	13	13	13	13	13
gsm-st231.13	43	43	43	43	44	43
gsm-st231.14	34	34	34	36	34	33
gsm-st231.15	12	12	12	12	12	12
gsm-st231.16	59	59	59	-	62	-
gsm-st231.17	33	33	33	33	33	33
gsm-st231.18	-	-	194	-	121	-
gsm-st231.19	15	15	15	15	15	15
gsm-st231.20	20	20	20	20	21	20
gsm-st231.21	30	30	30	30	30	30
gsm-st231.22	29	29	29	29	29	29
gsm-st231.25	56	56	55	-	56	56(gap=1.75%)
gsm-st231.29	42	42	42	42	42	42
gsm-st231.30	25	25	25	26	25	25
gsm-st231.31	39	39	39	41	39	39
gsm-st231.32	30	30	30	30	30	30
gsm-st231.33	52	52	52	-	51	52(gap=8%)
gsm-st231.34	8	8	8	7	8	7
gsm-st231.35	16	16	16	15	15	14
gsm-st231.36	29	29	29	27	25	24
gsm-st231.39	23	23	23	22	22	21
gsm-st231.40	17	17	17	17	18	17
gsm-st231.41	50	50	50	-	52	-
gsm-st231.42	19	19	19	19	19	18
gsm-st231.43	23	23	23	22	22	20

5.5 Conclusion

Dans ce chapitre, nous avons présenté une méthode hybride pour le RCMSP basée sur le pipeline logiciel décomposé. Nous avons utilisé des algorithmes de retiming pour résoudre la phase de décalage des instructions (détermination du numéro de la période) et la formulation PLNE décomposée structurée pour résoudre la phase de compaction de la boucle (détermination de la date de début dans la période). L'avantage de cette méthode par rapport aux méthodes décrites dans la littérature est que la PLNE permet de résoudre optimalement le sous-problème de compaction de la boucle. Dans le groupe d'instances industrielles la méthode hybride s'avère très efficace en ce qui concerne la qualité des solutions et le temps de résolution. Dans le groupe d'instances modifiées, la méthode hybride permet de résoudre un plus grand nombre d'instances que la PLNE. De plus, une amélioration significative du temps de résolution est observée. Nous remarquons que la choix de l'algorithme retiming dans la résolution de la phase de décalage des instructions est un aspect important. Le risque le plus important de la méthode, est que nous avons observé que pour certaines instances du groupe des instances modifiées, la phase de compaction de la boucle reste toujours très difficile à résoudre avec la PLNE. Ceci est également vrai avec l'algorithme de la relaxation lagrangienne où trois instances n'ont pas été resolues.

Conclusion et perspectives

Dans cette thèse, le problème de l'ordonnancement cyclique a éte abordé. Nous avons traité plus précisément le problème de l'ordonnancement modulo sous contraintes de ressources et l'application de celui-ci pour l'ordonnancement d'instructions de type VLIW. L'ordonnancement modulo est un ordonnancement périodique où toutes les tâches ont le même intervalle d'initiation λ et où l'allocation des ressources est conservée pour toutes les instances des tâches. Autrement dit, un problème d'ordonnancement modulo correspond à un problème d'ordonnancement périodique où l'on considère uniquement l'ensemble de tâches génériques. Ceci simplifie en effet de manière significative le problème à résoudre.

Différentes méthodes exactes et approchées ont été proposées pour résoudre le problème d'ordonnancement modulo. Dans ce travail, nous nous sommes focalisé sur la résolution du problème à partir de formulations linéaires en nombres entiers originellement proposées pour le RCMSP. Dans ce contexte, nous nous sommes servis de la formulation décomposée proposée Eichenberger and Davidson (1997) et de la formulation directe proposée par Dinechin (2004). Pour les deux formulations, des contraintes de précédence structurées et non-structurées ont été proposées. La résolution de ces formulations consiste en un processus itératif qui commence avec une valeur de λ fixée, une fois que le programme linéaire en nombres entiers trouve une solution pour une valeur λ, il est donc possible de construire un ordonnancement périodique valide.
La résolution des formulations linéaires en nombres entiers permettent donc de trouver des solutions réalisables pour le problème. Cependant la plupart des instances ne peuvent pas être résolues en un temps raisonnable. Une des contributions de cette thèse est l'étude théorique sur la qualité des relaxations des formulations PLNE et leur utilisation pour la recherche des bornes inférieures pour la période λ et pour le $Cmax$. Nous avons également vérifié que le temps nécessaire pour la résolution de la formulation (decomp(+)) est plus court que pour la formulation (direct(+)). D'autres études expérimentales ont néanmoins montré que la formulation directe s'adaptait bien à des techniques de grand voisinage qui permettent d'obtenir de très bonnes solutions approchés, ce qui montre donc leur intérêt indépendamment de la qualité des relaxations et du nombre de variables. En ce qui concerne le calcul des bornes pour la période λ, nous avons constaté expérimentalement sur les instances testées que les bornes inférieures obtenues à partir de la relaxation des contraintes d'intégrité sont égales aux bornes théoriques $\lambda = \max(\lambda_{prec}, \lambda_{res})$. Ceci est vérifié pour les deux groupes d'instances (industrielles et modifiées). Les bornes obtenues pour le critère secondaire (makespan) sont par contre plus fortes.

Dans le but d'améliorer les bornes obtenues à partir des relaxations de la PLNE, nous avons proposé des nouvelles formulations basées sur la décomposition de Dantzig-Wolfe pour résoudre le problème d'ordonnancement modulo sous contraintes de ressources. Ces nouvelles formula-

tions sont basées sur la définition des ensembles de tâches réalisables qui peuvent être exécutées simultanément en respectant les contraintes de précédence et de ressources. En effet, la modélisation des contraintes de ressources génère un grand nombre de variables, ce qui entraîne des temps de résolution assez importants. Pour résoudre la relaxation de ces nouvelles formulations, un schéma de génération de colonnes dont les sous-problèmes ont été identifiés comme des problèmes de sac-à-dos multidimensionnel a été proposé. La méthode permet d'atteindre une amélioration significative des bornes pour la période λ et pour le $Cmax$ dans le cas des instances modifiées. Dans le cas des instances industrielles, le schéma de génération de colonnes n'améliore pas les résultats obtenus avec la résolution des relaxations des PLNE classiques.

Bien que l'objectif premier de ce travail a été toujours la recherche de bornes inférieures, Nous nous sommes également intéressés à la recherche de bornes supérieures (solutions réalisables du problème). Dans ce contexte, nous avons d'abord proposé le calcul d'une borne inférieure à partir de la relaxation Lagrangienne obtenue avec la dualisation des contraintes de ressources de la formulation directe désagrégée. Le problème lagrangien peut être transformé en un problème de graphe (problème de coupe minimale). Le problème dual obtenu a été résolu avec la méthode du sous-gradient. La principal avantage de cette méthode est que les résultats du problème lagrangienne peuvent être utilisés pour proposer une méthode pour la recherche de bornes supérieures. Les résultats expérimentaux ont montré que pour les bornes inférieures notre implémentation de la relaxation lagrangienne ne permettait pas d'accélérer les temps de calcul par rapport à la relaxation des formulations PLNE. En ce qui concerne des bornes supérieures, dans le cas des instances industrielles, les bornes trouvées sont presque toujours égales aux valeurs optimales trouvées avec la résolution des formulations PLNE. Pour les instances modifiées, la méthode arrive à trouver une solution optimale uniquement pour la moitié des instances.

Pour améliorer les résultats des bornes supérieures obtenues dans le chapitre 4, nous avons proposée dans le chapitre 5 une méthode hybride pour la recherche de bornes supérieures pour la période λ. En fait, les méthodes de calcul des bornes supérieures des chapitres 4 et 5 sont basées sur le même principe : le calcul préalable des numéros de périodes d'exécution (valeurs k_i dans la formulation décomposée), suivi du calcul des dates de début dans la période. La différence entre les méthodes est la façon de calculer les valeurs k_i (en utilisant la solution de la relaxation lagrangienne dans le premier cas).

La méthode presénté dans le chapitre 5 est basée sur le principe du pipeline logiciel décomposé. Nous avons utilisé des algorithmes de calcul de retiming pour résoudre la phase de décalage des instructions et la formulation PLNE décomposée structurée pour résoudre la phase de compaction de la boucle. L'avantage de cette méthode par rapport à d'autres méthodes décrites dans la littérature est que la PLNE permet de résoudre optimalement le sous-problème de compaction de la boucle (comme également la méthode proposée dans le chapitre précédent). Les résultats expérimentaux ont montré une très bonne performance lorsqu'on regarde le rapport entre la qualité des solutions et les temps de résolution. L'un des principaux avantages de cette méthode par rapport à la PLNE est l'amélioration significative du temps de résolution ainsi que la résolution d'un nombre plus grand d'instances modifiées.

Parmi les perspectives de cette thèse nous identifions trois possibles axes. Le premier axe qui peut être vu comme une perspective à court terme est l'amélioration de l' implémentation du code de programmation du schéma de génération de colonnes proposé. L'idée est d'inclure dans le code des règles déduites dans le troisième chapitre de manière à réduire l'ensemble de sous-ensembles réalisables. Dans ce même contexte, la méthode d'insertion de colonnes peut être également

améliorée par rapport à l'implémentation basique du code utilisée. Un deuxième piste intéressante est liée à la possibilité de générer une méthode heuristique basée sur la relaxation lagrangienne pour améliorer les résultats obtenus en ce qui concerne la recherche de solutions réalisables, principalement dans le cas où la consommation de ressources est non-unitaire. Il faudra pour cela parvenir à une implémentation beaucoup plus efficace de la relaxation lagrangienne. Finalement, une perspective à plus long terme serait l'extension des méthodes et techniques proposées pour des problèmes d'ordonnancement stochastiques en considérant la durée des opérations ou d'autres paramètres du problème comme des variables aléatoires.

Bibliographie

Alexander Aiken, Alexandru Nicolau, and Steven Novack. Resource-constrained software pipelining. *IEEE Trans. Parallel & Distributed Systems*, 6 :1248–1270, December 1995.

Maria Ayala and Christian Artigues. Programmation linéaire en nombres entiers pour l'ordonnancement modulo sous contraintes de ressources. In *Roadef*, 2009.

Maria Ayala and Christian Artigues. On integer linear programming formulations for the resource-constrained modulo scheduling problem. Technical report, LAAS-CNRS, France, 2011.

Maria Ayala, Christian Artigues, Claire Hanen, and Abir Benabid. The resource-constrained modulo scheduling problem : an experimental study. Technical report, LAAS-CNRS, France, 2011.

Cynthia Barnhart, Ellis L. Johnson, George L. Nemhauser, Martin W. P. Savelsbergh, and Pamela H. Vance. Branch-and-Price : Column generation for solving huge integer programs. *Operations Research*, 46(3) :316–329, 1998.

Abir Benabid. *Etude du RCPSP Cyclique*. PhD thesis, Thèse d'état, Université P. et M. Curie, 2011.

Abir Benabid and Claire Hanen. Decomposed Software Pipelining for cyclic unitary RCPSP with precedence delays. In *MISTA*, 2009.

Abir Benabid and Claire Hanen. Worst case analysis of decomposed software pipelining for cyclic unitary RCPSP with precedence delays. *Journal of Scheduling*, pages 1–12, 2011.

Frédéric Boussemart, G. Cavory, and Christophe Lecoutre. Solving the cyclic job shop scheduling problem with linear precedence constraints using CP techniques. In *IEEE International Conference on Systems, Man and Cybernetics(SMC'02)*, Hammamet, Tunisia, oct 2002.

N. Brauner and G. Finke. Cycles and permutations in robotic cells. *Mathematical and Computer Modelling*, 34(5-6) :565 – 591, 2001.

Nadia Brauner. Identical part production in cyclic robotic cells : Concepts, overview and open questions. *Discrete Applied Mathematics*, 156(13) :2480–2492, 2008.

Peter Brucker and Thomas Kampmeyer. Tabu search algorithms for cyclic machine scheduling problems. *Journal of Scheduling*, 8 :303–322, July 2005.

Peter Brucker and Thomas Kampmeyer. A general model for cyclic machine scheduling problems. *Discrete Applied Mathematics.*, 156 :2561–2572, July 2008.

Pierre-Yves Calland, Alain Darte, and Yves Robert. A new guaranteed heuristic for the software pipelining problem. In *Proceedings of the 10th international conference on Supercomputing*, ICS '96, pages 261–269, New York, NY, USA, 1996. ACM.

Pierre-Yves Calland, Alain Darte, and Yves Robert. Circuit retiming applied to decomposed software pipelining. *IEEE Trans. Parallel Distributed System.*, 9 :24–35, January 1998.

P. M. Camerini, L. Fratta, and F. Maffioli. On improving relaxation methods by modified gradient techniques. In *Nondifferentiable Optimization*, volume 3 of *Mathematical Programming Studies*, pages 26–34. Springer Berlin Heidelberg, 1975.

Jacqes Carlier, Philippe Chrétienne, Jacques Erschler, Claire Hanen, Pierre Lopez, Alix Munier, Eric Pinson, Marie Claude Portmann, Christians Prins, Christian Proust, and Pierre Villon. Les problèmes d'ordonnancement. *RAIRO-Recherche Opérationnelle*, 27 :77–150, 1993.

Samit Chaudhuri, Robert A. Walker, and J. E. Mitchell. Analyzing and exploiting the structure of the constraints in the ILP approach to the scheduling problem. *IEEE Trans. VLSI Systems*, 2(4) :456–471, 1994.

Ada Che and Chengbin Chu. A polynomial algorithm for no-wait cyclic hoist scheduling in an extended electroplating line. *Operations Research Letters*, 33(3) :274–284, 2005.

Ada Che and Chengbin Chu. Multi-degree cyclic scheduling of a no-wait robotic cell with multiple robots. *European Journal of Operational Research*, 199(1) :77–88, 2009.

Haoxun Chen, Chengbin Chu, and Jean-Marie Proth. Cyclic hoist scheduling based on graph theory. In *IEEE*, pages 451–459, 1995.

Philippe Chrétienne. On Graham's bound for cyclic scheduling. *Parallel Computing*, 26(9) : 1163–1174, 2000.

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

V. Chvátal. Edmonds polytopes and a hierarchy of combinatorial problems. *Discrete Mathematics*, 306(10-11) :886 – 904, 2006. 35th Special Anniversary Issue.

G. Cohen, D. Dubois, J.P. Quadrat, and M. Viot. A linear system theoretic view of discrete event processes and its use for performance evaluation in manufacturing. *IEEE Trans. on Automatic Control*, AC-30 :210–220, 1985.

Y. Crama, V. Kats, J. van de Klunder, and E. Levner. Cyclic scheduling in robotic flowshops. Technical report, Maastricht University, 2000.

George Dantzig. *Maximization of a linear function of variables subject to linear inequalities*, pages 339–347. Wiley, 1951.

George B. Dantzig and Philip Wolfe. Decomposition principle for linear programs. *Operations Research*, 8(1) :101–111, 1960.

Alain Darte and Guillaume Huard. Loop shifting for loop compaction. *Int. J. Parallel Program.*, 28 :499–534, October 2000.

Milind Dawande, H. Neil Geismar, Suresh P. Sethi, and Chelliah Sriskandarajah. Sequencing and scheduling in robotic cells : Recent developments. *Journal of Scheduling*, 8 :387–426, October 2005.

Benoît Dupont de Dinechin. Insertion scheduling : An alternative to list scheduling for modulo schedulers. In *LCPC*, pages 31–45, 1995.

Benoît Dupont de Dinechin. A unified software pipeline construction scheme for modulo scheduled loops. In *LCPC*, pages 382–393, 1997.

Benoît Dupont de Dinechin. Extending modulo scheduling with memory reference merging. In *Compiler Construction 8th International Conference*, pages 274–287, 1999.

Benoît Dupont de Dinechin. Modulo scheduling with regular unwinding. In *DOM*, 2004.

Benoît Dupont de Dinechin. Time-indexed formulations and a large neighborhood search for the resource-constrained modulo scheduling problem. In *MISTA*, pages 144–151, 2007.

Benoît Dupont de Dinechin, Christian Artigues, and Sadia Azem. *Resource-Constrained Modulo Scheduling*, pages 267–277. ISTE, 2010.

Sophie Demassey. *Méthodes hybrides de programmation par contraintes et programmation linéaire pour le problème d'ordonnancement de projet à contraintes de ressources.* PhD thesis, Université d'Avignon, 2003.

Benoît Dupont De Dinechin. From machine scheduling to VLIW instruction scheduling. *Advances*, 1(2) :1–35, 2004.

Noll Dominikus. Notes optimisation. Technical report, Institute de Mathématiques, Université Paul Sabatier, France, 2007.

Rémy Dupas. *Amélioration de performance des systémes de production : apport des algorithmes évolutionnistes aux problémes d'ordonnancement cycliques.* PhD thesis, HDR, Université d'Artois, 2004.

Imed Eddine, Bennour et El, and Mostapha Aboulhamid. Les problémes d'ordonnancement cyclique dans la synthèse des circuits numériques. Technical report, Université de Montréal, Montréal Canada, 2002.

Alexandre E. Eichenberger and Edward S. Davidson. Efficient formulation for optimal modulo schedulers. In *In Proc. of the SIGPLAN 97 Conference on Programming Language Design and Implementation*, pages 194–205, 1997.

Franco Gasperoni and Uwe Schwiegelshohn. Scheduling loops on parallel processors : A simple algorithm with close to optimum performance. In *CONPAR*, pages 625–636, 1992.

Franco Gasperoni and Uwe Schwiegelshohn. Generating close to optimum loop schedules on parallel processors. *Parallel Processing Letters*, 4 :391–403, 1994.

Arthur M. Geoffrion. Lagrangian relaxation for integer programming. *Mathematical Programming Study*, 2 :625–636, 1974.

Ralph E. Gomory. Outline of an algorithm for integer solutions to linear programs. *Bulletin of The American Mathematical Society*, 64 :275–279, 1958.

Michel Gondran and Michel Minoux. *Graphes et algorithmes*. Eyrolles, 1995.

Ramaswamy Govindarajan, Erik R. Altman, and Guang R. Gao. A framework for resource-constrained rate-optimal software pipelining. In *Proceedings of the Third Joint International Conference on Vector and Parallel Processing : Parallel Processing*, CONPAR 94 - VAPP VI, pages 640–651, London, UK, 1994. Springer-Verlag.

Hakan Gultekin, M. Selim Akturk, and Oya Ekin Karasan. Cyclic scheduling of a 2-machine robotic cell with tooling constraints. *European Journal of Operational Research*, pages 777–796, 2006.

Claire Hanen and Alix Munier. Study of a NP-hard cyclic scheduling problem : The recurrent job-shop. *Journal of Operational Research*, 72 :82–101, 1994a.

Claire Hanen and Alix Munier. Cyclic scheduling on parallel processors : An overview. In Philippe Chrétienne, Edward Coffman, Jan Karel Lenstra, and Zhen Liu, editors, *Scheduling theory and its applications*, volume 3, pages 193–226. Wiley, 1994b.

Claire Hanen and Alix Munier. A study of the cyclic scheduling problem on parallel processors. *Discrete Applied Mathematics*, pages 167–192, 1995.

Claire Hanen and Alix Munier. Periodic schedules for linear precedence constraints. *Discrete Applied Mathematics*, 157(2) :280–291, 2009.

Hervé Hillion and Jean-Marie Proth. Performance evaluation of a job-shop system using timed event graph. *IEEE Transactions on Automatic Control*, 34(1) :3–9, 1989.

Guillaume Huard. Retiming et parallélisation automatique. Technical report, Ecole Normale Supérieure de Lyon, France, 2008.

Thomas Kampmeyer. *Cyclic Scheduling Problems*. PhD thesis, Universitat Osnabruck, 2006.

Vladimir Kats and Eugene Levner. A strongly polynomial algorithm for no-wait cyclic robotic flowshop scheduling. *Oper. Res. Lett.*, pages 171–179, 1997.

Yura Kenji. Cyclic scheduling for re-entrant manufacturing systems. *International Journal of Production Economics*, 60-61 :523–528, 1999.

Ouajdi Korbaa and Jean-Claude Gentina. Cyclic scheduling in flexible manufacturing systems. *Revue internationale d'ingénierie des systèmes de production mécanique*, pages 47–54, 2000.

Marek Kubale and Adam Nadolski. Chromatic scheduling in a cyclic open shop. *European Journal of Operational Research*, 164(3) :585–591, 2005.

Monica Lam. Software pipelining : An effective scheduling technique for VLIW machines. In *In Proc. of the SIGPLAN 97 Conference on Programming Language Design and Implementation*, pages 318–328, 1988.

L. Lei and T.J. Wang. A proof : the cyclic hoist scheduling problem is NP-complete. Technical report, School of Management, Rutgers University, New Jersey USA, 1989.

Charles E. Leiserson and James B. Saxe. Retiming synchronous circuitry. *Algorithmica*, 6(1) : 5–35, 1991.

Eugene Levner, Vladimir Kats, Alcaide López de Pablo, and David Cheng. Complexity of cyclic scheduling problems : A state-of-the-art survey. *Computers & Industrial Engineering*, 59 : 352–362, 2010.

M. Lombardi, A. Bonfietti, M. Milano, and L. Benini. Precedence constraint posting for cyclic scheduling problems. In *CPAIOR*, 2011.

M. Middendorf and V.G. Timkovsky. On scheduling cycle shops : classification, complexity and approximation. *Computers & Industrial Engineering*, 5 :135–169, 2002.

Aristide Mingozzi, Vittorio Maniezzo, Salvatore Ricciardelli, and Lucio Bianco. An exact algorithm for the resource constrained project scheduling problem based on a new mathematical formulation. *Management Science*, 44 :714–729, 1997.

Rolf H. Möhring, Andreas S. Schulz, Frederik Stork, and Marc Uetz. Solving project scheduling problems by minimum cut computations. *Management Science*, 49 :330–350, 2000.

Alix Munier. Résolution d'un problème d'ordonnancement cyclique à itérations indépendantes et contraintes de ressources. *RAIRO. Recherche opérationnelle*, 25 :161–182, 1991.

Alix Munier. The basic cyclic scheduling problem with linear precedence constraints. *Discrete Applied Mathematics*, 64 :219–238, 1996a.

Alix Munier. The complexity of a cyclic scheduling problem with identical machines and precedence constraints. *European Journal of Operational Research*, 91(3) :471–480, 1996b.

Korbaa Ouajdi, H. Camus, and Jean-Claude Gentin. Heuristic for the resolution of the general FMS cyclic scheduling problem. In *Systems, Man, and Cybernetics. Computational Cybernetics and Simulation. IEEE Conference*, pages 2903–2908, 1997.

Christos H. Papadimitriou and Kenneth Steiglitz. *Combinatorial Optimization : Algorithms and Complexity*. Prentice-Hall, New York, 1982.

L. W. Phillipsa and P. S. Ungerb. Mathematical programming solution of a hoist scheduling program. *IIE Transactions*, 8 :219–225, 1976.

A. Pritsker, L. Watters, and P. Wolfe. Multi-project scheduling with limited resources : a zero-one programming approach. *Management Science*, 16 :93–108, 1969.

Subramanian Rajagopalan, Sreeranga P. Rajan, Sharad Malik, Sandro Rigo, Guido Araujo, and Koichiro Takayama. A retargetable VLIW compiler framework for DSPs withinstruction-level parallelism. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 20(11) :1319–1328, 2001.

C. V. Ramamoorthy and G. S. Ho. Performance evaluation of asynchronous concurrent systems using Petri nets. *IEEE Trans. Softw. Eng.*, 6 :440–449, September 1980.

Chander Ramchandani. *Analysis of Asynchronous Concurrent Systems by Timed Petri*. PhD thesis, Universität Osnabrück, Fachbereich, Fachbereich Mathematk/Informatik, 2006.

B. Ramakrishna Rau. Iterative modulo scheduling : an algorithm for software pipelining loops. In *Proceedings of the 27th annual international symposium on Microarchitecture*, MICRO 27, pages 63–74, New York, NY, USA, 1994. ACM.

B. Ramakrishna Rau and Joseph A. Fisher. Instruction-level parallel processing : history, overview, and perspective. *J. Supercomput.*, 7 :9–50, May 1993.

Andre Rossi and Marc Sevaux. Mixed-integer linear programming formulation for high level synthesis. In *International Workshop on Project Management and Scheduling, PMS 2008*, pages 222–226, Istanbul Turkey, 2008.

Robin Roundy. Cyclic schedules for job shops with identical jobs. *Mathematics of Operations Research*, 17 :842–865, September 1992.

Jeong-Won Seo and Tae-Eog Lee. Steady-state analysis and scheduling of cyclic job shops with overtaking. *International Journal of Flexible Manufacturing Systems*, 14 :291–318, 2002.

P. Serafini and W. Ukovich. A mathematical for periodic scheduling problems. *SIAM J. Discret. Math.*, 2 :550–581, November 1989.

Marc Sevaux, Alok Singh, and André Rossi. Tabu search for multiprocessor scheduling : Application to high level synthesis. *APJOR*, 28(2) :201–212, 2011.

Francois Vanderbeck. On dantzig-wolfe decomposition in integer programming and ways to perform branching in a branch-and-price algorithm. *Operations Research*, 48 :111–128, 2000.

Přemysl Šůcha and Zdeněk Hanzálek. Deadline constrained cyclic scheduling on pipelined dedicated processors considering multiprocessor tasks and changeover times. *Mathematical and Computer Modelling*, 47(9-10) :925–942, 2008.

Přemysl Šůcha and Zdeněk Hanzálek. A cyclic scheduling problem with an undetermined number of parallel identical processors. *Comput. Optim. Appl.*, 48 :71–90, January 2011.

Jian Wang, Christine Eisenbeis, Martin Jourdan, and Bogong Su. Decomposed software pipelining : A new perspective and a new approach. *International Journal of Parallel Programming*, 22 :351–373, 1994.

Jian Xu, Tienté Hsu, and Gilles Goncalves. A genetic algorithm to solving the problem of flexible manufacturing system cyclic scheduling. In *Systems, Man and Cybernetics, 2002 IEEE International Conference*, pages 640–651, 2002.

Table des matières

Introduction **13**

1 Ordonnancement modulo sous contraintes de ressources **15**
1.1 Introduction 15
1.2 Problèmes d'ordonnancement cyclique classiques 16
1.2.1 Le problème central répétitif 16
1.2.2 Problèmes d'ordonnancement cyclique sous contraintes de ressources de type "machine" 20
1.3 Ordonnancement modulo sous contraintes de ressources 26
1.3.1 Contexte : parallélisme d'instructions et pipeline logiciels 26
1.3.2 Formulation du problème d'ordonnancement modulo sous contraintes de ressources 28
1.3.3 Modèle d'architecture ST200 31
1.3.4 Description des instances pour l'expérimentation 34
1.4 Conclusion 35

2 PLNE pour le RCMSP **37**
2.1 Introduction 37
2.2 Programmation linéaire en nombres entiers 38
2.2.1 Relaxations et bornes 39
2.2.2 Résolution des programmes linéaires en nombres entiers 40
2.3 Problème d'ordonnancement de projet sous contraintes de ressources 42
2.4 Programmation linéaire en nombres entiers pour l'ordonnancement modulo sous contraintes de ressources 44
2.4.1 Formulation décomposée 44
2.4.2 Formulation directe 46
2.4.3 Comparaison des relaxations de programmation linéaire 47
2.4.4 Calcul de bornes inférieures et résolution exacte 53
2.5 Résultats expérimentaux 54
2.6 Conclusion 62

3 Décomposition de Dantzig-Wolfe pour le RCMSP **63**
3.1 Introduction 63
3.2 Décomposition de Dantzig-Wolfe et génération de colonnes 64

3.3 Formulations renforcées pour le RCMSP 67
3.3.1 Formulations renforcées pour le RCMSP basées sur la décomposition de Dantzig-Wolfe 67
3.3.2 Réduction du nombre d'ensembles réalisables 69
3.4 Génération de colonnes pour le RCMSP 70
3.4.1 Ensemble initial des colonnes 72
3.4.2 Calcul des bornes inférieures 72
3.5 Résultats expérimentaux 72
3.6 Conclusion 77

4 Relaxation Lagrangienne et solutions réalisables 79
4.1 Introduction 79
4.2 Relaxation Lagrangienne 80
4.2.1 Solution Lagrangienne réalisable 81
4.3 Relaxation Lagrangienne de la formulation directe desagrégée 82
4.3.1 Définition et résolution du sous-problème Lagrangien 83
4.3.2 Résolution du problème dual Lagrangien 85
4.4 Heuristique et résultats expérimentaux 86
4.5 Conclusion 92

5 Heuristique hybride pour le RCMSP 93
5.1 Introduction 93
5.2 Pipeline logiciel décomposé (DSP) 94
5.2.1 Phase de décalage des instructions 94
5.2.2 Phase de compaction de la boucle 98
5.3 Formulation hybride pour le RCMSP 99
5.4 Résultats expérimentaux 100
5.5 Conclusion 106

Conclusion et perspectives 107

Table des figures

1.1 Graphe de précédences et ordonnancement périodique de période $\lambda = 1$ 17
1.2 Exemple de latence et distance. 18
1.3 Circuit critique de valeur $\lambda = 4$ 19
1.4 Exemple d'ordonnancement avec $\lambda = 4$. 20
1.5 Graphes avec des arcs caractérisés par la valeur $\theta_i^j - \lambda\omega_i^j$ pour λ fixée. 20
1.6 Exemple d'une séquence d'immersion avec 2 hoist ou robots. 23
1.7 Exemple de machine parallèle MIMD. 25
1.8 Ordonnancement modulo sous contraintes de ressources optimal. 30
1.9 Exemple de programme en langage C et ses opérations ST200 correspondantes. . 32
1.10 Graphe de précédence uniforme. 33
1.11 Graphe de précédence uniforme et ordonnancement périodique. 34

2.1 Exemple de programmation linéaire entière et de relaxation. 40
2.2 Exemple d'ordonnancement avec la formulation décomposée. 45
2.3 Exemple d'ordonnancement avec la formulation directe. 46

3.1 Bornes pour la période λ. Instances modifiées. 75
3.2 Bornes pour le Cmax. Instances modifiées. 76

4.1 Structure du graphe G'. 84

Liste des tableaux

1.1 Disponibilité et demande de ressources. 29
1.2 Disponibilité des ressources cumulatives et classes de ressources. 32
1.3 Disponibilité des ressources cumulatives 33
1.4 Caractéristiques des instances industrielles (processeur ST200) et des instances modifiées 35

2.1 Formulation **{decomp+}** instances industrielles 55
2.2 Formulation **{direct+}** instances industrielles 55
2.3 Formulation **{decomp+}** instances modifiées 55
2.4 Formulation **{direct+}** instances modifiées 56
2.5 Solutions optimales ou réalisables pour les instances industrielles 58
2.6 Solutions optimales ou réalisables pour les instances modifiées 59
2.7 Résultats des bornes inférieures obtenues à partir de la relaxation des formulations **{direct+}** et **{decomp+}** pour les instances industrielles 60
2.8 Résultats des bornes inférieures obtenues à partir de la relaxation des formulations **{direct+}** et **{decomp+}** pour les instances modifiées 61

3.1 Récapitulatif des bornes obtenues pour la période λ 73
3.2 Récapitulatif des bornes obtenues pour Cmax 74
3.3 Comparaison des bornes pour la période λ 74
3.4 Comparaison des bornes pour le Cmax 74
3.5 Comparaison du nombre d'instances résolues classées en fonction de taille et temps de calcul. 75
3.6 Bornes inférieures pour la période λ et Cmax obtenues par génération de colonnes pour les instances modifiées. 76

4.1 Récapitulatif des bornes supérieures obtenues pour la période λ et pour le $Cmax$. Instances industrielles 87
4.2 Récapitulatif des bornes supérieure obtenues pour la période λ et pour le $Cmax$. Instances modifiées 87
4.3 Bornes inférieures pour la période λ et pour le $Cmax$. Instances industrielles . . 88
4.4 Bornes inférieures pour la période λ et pour le $Cmax$. Instances modifiées 89
4.5 Bornes supérieures pour la période λ et pour le $Cmax$. Instances industrielles . . 90
4.6 Bornes supérieures pour la période λ et pour le $Cmax$. Instances modifiées . . . 91

5.1 Vecteur retiming légal ρ_i pour l'exemple 1.3. 95
5.2 Récapitulatif des résultats obtenus pour la période λ avec la méthode hybride. Instances industrielles 100
5.3 Récapitulatif des résultats obtenus pour la période λ avec de la méthode hybride. Instances modifiées 101
5.4 Comparaison de la méthode hybride avec des autres méthodes pour la recherche des solutions réalisables. Instances industrielles 101
5.5 Comparaison de la méthode hybride avec des autres méthodes pour la recherche des solutions réalisables. Instances modifiées 101
5.6 Distribution du temps de calcul avec la méthode hybride pour les deux ensembles d'instances. 102
5.7 Résultats de la méthode hybride pour les instances industrielles 103
5.8 Résultats de la méthode hybride pour les instances modifiées 104
5.9 Résultats obtenus avec les différents méthodes de la littérature. Instances modifiées105