Optimisation du fonctionnement d'un générateur de hiérarchies mémoires pour les systèmes de vision embarquée

Khadija Hadj Salem

Communauté
UNIVERSITÉ Grenoble Alpes

THÈSE

Pour obtenir le grade de

DOCTEUR DE LA COMMUNAUTE UNIVERSITÉ GRENOBLE ALPES

Spécialité : **Mathématiques et Informatique**

Arrêté ministériel : 25 Mai 2018

Présentée par

Khadija HADJ SALEM

Thèse dirigée par **Stéphane MANCINI**
et codirigée par **Yann KIEFFER**

préparée au sein **du laboratoire LCIS à Valence**
et de **l'École Doctorale MSTII**

Optimisation du Fonctionnement d'un Générateur de Hiérarchies Mémoires pour les Systèmes de Vision Embarquée

Thèse soutenue publiquement le **26 Avril 2018**,
devant le jury composé de :

Mme Nadia BRAUNER
Professeur des Universités, Université de Grenoble Alpes-G-SCOP, Présidente
M. André ROSSI
Professeur des Universités, Université d'Angers-LERIA, Rapporteur
M. Samy MEFTALI
Maître de Conférences-HDR, Université de Lille1-CRISTAL, Rapporteur
Mme Lilia ZAOURAR
Ingénieur de Recherche-PhD, CEA List, Examinatrice
M. Stéphane MANCINI
Maître de Conférences-HDR, Université de Grenoble Alpes-TIMA, Directeur de thèse
M. Yann KIEFFER
Maître de Conférences, Université de Grenoble Alpes-LCIS, Co-Directeur de thèse

Je dédie cette thèse :
À la mémoire de ma mère
Que je l'aime pour toujours ! Elle aurait, sans doute, été fière de moi.
À la mémoire de mon petit frère, le petit ange au paradis.
À ma belle famille
À tous ceux qui m'aiment
À tous ceux que j'aime.
*** Khadija HADJ SALEM (MARS) ***

Remerciements

Le travail présenté dans le manuscrit a été réalisé dans l'équipe Conception et Test de Systèmes embarqués (CTSYS) du laboratoire de Conception et d'Intégration des Systèmes (LCIS) à Valence en collaboration avec le laboratoire de "Techniques de l'Informatique et de la Microélectronique pour l'Architecture des systèmes intégrés"(TIMA) à Grenoble.

J'ai été principalement encadrée par Yann KIEFFER que je tiens tout particulièrement à lui remercier tout d'abord pour son accueil qu'a été toujours chaleureux, pour la confiance qu'il m'a accordée et pour ses nombreux conseils et son aide.

Mes remerciements vont également à Stéphane MANCINI d'avoir accepté de diriger ma thèse. Merci de m'avoir initié au plaisir de relever des défis pour comprendre quelques notions dans le domaine de l'électronique.

Je tiens aussi à remercier vivement :

— Monsieur André ROSSI, Professeur à l'Université d'Angers

— Monsieur Samy MEFTALI, Maître de Conférences (HDR) à l'Université de Lille 1

qui m'ont fait l'honneur d'étudier mes travaux de recherche et d'en être les rapporteurs de cette thèse,

— Madame Nadia BRAUNER VETTIER, Professeur à l'Université Grenoble Alpes (UJF-UFR IMAG)

— Madame Lilia ZAOURAR, Ingénieur de recherche (PhD) au laboratoire CEA LIST

pour avoir accepté de participer au jury.

Je n'oublie pas mes collègues et amis du laboratoire LCIS. J'adresse aussi une pensée particulière à tous les membres du personnel de LCIS et toutes les personnes (stagiaires, doctorants et post-doctorants) que j'ai eu l'occasion de croiser pendant les années de thèse. Je remercie également, pour l'accueil chaleureux qu'ils m'ont réservé, les membres du laboratoire des Sciences pour la Conception, l'Optimisation et la Production de Grenoble (G-SCOP), où j'ai passé mes derniers mois de thèse en tant qu'ATER à l'École Nationale Supérieure de Génie Industriel (ENS GI) de Grenoble-INP.

Je remercie aussi ma famille et mes ami(e)s pour leur soutien inconditionnel pendant toutes ces années de thèse.

Finalement, merci à toi, lecteur, de t'intéresser à mon travail.

" Tout Début est Difficile, Compliqué au Milieu, et Magnifique à la Fin, . . .
Voilà, c'est fini après 3 ans d'une belle aventure. Une page se tourne, mais une autre s'ouvre.
Merci à tous pour tout . . . "
(Khadija H.S.)

Résumé

Les recherches de cette thèse portent sur la mise en œuvre des méthodes de la recherche opérationnelle (RO) pour la conception de circuits numériques dans le domaine du traitement du signal et de l'image, plus spécifiquement pour des applications multimédia et de vision embarquée.

Face à la problématique de "Memory Wall", les concepteurs de systèmes de vision embarquée, Mancini et al. (Proc.DATE, 2012), ont proposé un générateur de hiérarchies mémoires ad-hoc dénommé Memory Management Optimization (MMOpt). Cet atelier de conception est destiné aux traitements non-linéaires afin d'optimiser la gestion des accès mémoires de ces traitements. Dans le cadre de l'outil MMOpt, nous abordons la problématique d'optimisation liée au fonctionnement efficace des circuits de traitement d'image générés par MMOpt visant l'amélioration des enjeux de performance (contrainte temps-réel), de consommation d'énergie et de coût de production.

Ce problème électronique a été modélisé comme un problème d'ordonnancement multi-objectif, appelé 3-objective Process Scheduling and Data Prefetching Problem (3-PSDPP), reflétant les 3 principaux enjeux électroniques considérés. À notre connaissance, ce problème n'a pas été étudié avant dans la littérature de RO. Une revue de l'état de l'art sur les principaux travaux liés à cette thèse, y compris les travaux antérieurs proposés par Mancini et al. (Proc.DATE, 2012) ainsi qu'un bref aperçu sur des problèmes voisins trouvés dans la littérature de RO, a ensuite été faite. En outre, la complexité de certaines variantes mono-objectif du problème d'origine 3-PSDPP a été établie. Des approches de résolution, y compris les méthodes exactes (PLNE) et les heuristiques constructives, sont alors proposées. Enfin, la performance de ces méthodes a été comparée par rapport à l'algorithme actuellement utilisé dans l'outil MMOpt, sur des benchmarks disponibles dans la littérature ainsi que ceux fournis par Mancini et al. (Proc.DATE, 2012).

Les solutions obtenues sont de très bonne qualité et présentent une piste prometteuse pour optimiser les performances des hiérarchies mémoires produites par MMOpt. En revanche, vu que les besoins de l'utilisateur de l'outil sont contradictoires, il est impossible de parler d'une solution unique en optimisant simultanément les trois critères considérés. Un ensemble de bonnes solutions de compromis entre ces trois critères a été fourni. L'utilisateur de l'outil MMOpt peut alors décider de la solution qui lui est la mieux adaptée.

Mots-clés : Systèmes de Vision Embarquée, Circuits Intégrés, Accès Mémoires, Memory Wall, Performance, Consommation d'énergie, Coût de Production, Optimisation Multi-objectif, Recherche Opérationnelle, Ordonnancement, Heuristiques Constructives.

Abstract

The research of this thesis focuses on the application of the Operations Research (OR) methodology to design new optimization algorithms to enable low cost and efficient embedded vision systems, or more generally devices for multimedia applications such as signal and image processing.

The design of embedded vision systems faces the "Memory Wall" challenge regarding the high latency of memories holding big image data. For the case of non-linear image accesses, one solution has been proposed by Mancini et al. (Proc. DATE 2012) in the form of a software tool, called Memory Management Optimization (MMOpt), that creates an ad-hoc memory hierarchies for such a treatment. It creates a circuit called a Tile Processing Unit (TPU) that contains the circuit for the treatment. In this context, we address the optimization challenge set by the efficient operation of the circuits produced by MMOpt to enhance the 3 main electronic design characteristics. They correspond to the energy consumption, performance and size/production cost of the circuit.

This electronic problem is formalized as a 3-objective scheduling problem, which is called 3-objective Process Scheduling and Data Prefetching Problem (3-PSDPP), reflecting the 3 main electronic design characteristics under consideration. To the best of our knowledge, this problem has not been studied before in the OR literature. A review of the state of the art, including the previous work proposed by Mancini et al. (Proc.DATE, 2012) as well as a brief overview on related problems found in the OR literature, is then made. In addition, the complexity of some of the mono-objective sub-problems of 3-PSDPP problem is established. Several resolution approaches, including exact methods (ILP) and polynomial constructive heuristics, are then proposed. Finally, the performance of these methods is compared, on benchmarks available in the literature, as well as those provided by Mancini et al. (Proc.DATE, 2012), against the one currently in use in the MMOpt tool.

The results show that our algorithms perform well in terms of computational efficiency and solution quality. They present a promising track to optimize the performance of the TPUs produced by MMOpt. However, since the user's needs of the MMOpt tool are contradictory, such as low cost, low energy and high performance, it is difficult to find a unique and optimal solution to optimize simultaneously the three criteria under consideration. A set of good compromise solutions between these three criteria was provided. The MMOpt's user can then choose the best compromise solution he wants or needs.

Keywords : Embedded Vision Systems, Integrated Circuit, Memory Accesses, Memory Wall, Performance, Energy Consumption, Production Cost, Multi-Objective Problem, Scheduling, Operations Research, Optimization Algorithms, Constructive Heuristics.

Table des matières

Liste des figures

Liste des tableaux

Liste des notations

Notation	Signification
$\mathscr{X} = \{1, \ldots, \mathrm{X}\}, \mathrm{X} \in \mathbb{N}^*$	Ensemble de X tuiles d'entrée
x	Une tuile d'entrée à pré-charger
$\mathscr{Y} = \{1, \ldots, \mathrm{Y}\}, \mathrm{Y} \in \mathbb{N}^*$	Ensemble de Y tuiles de sortie
y	Une tuile de sortie à calculer
$\mathrm{X}' \in \mathbb{N}^*$	Nombre de tuiles d'entrée requises par au moins une tuile de sortie
$\mathscr{X}' = \{1, \ldots, \mathrm{X}'\}$	Ensemble de X' tuiles d'entrée
Ω	l'ensemble des tuiles d'entrée qui ne sont jamais requises pour le calcul d'une tuile de sortie
$\mathscr{Z} = \{1, \ldots, \mathrm{Z}\}, \mathrm{Z} \in \mathbb{N}^*$	Ensemble de buffers
z	Un buffer (mémoire interne)
$\mathscr{R}_y, \forall\ y \in \mathscr{Y}$	Ensemble de tuiles d'entrée requises par chaque tuile de sortie y (R : Requirement)
$\mathscr{Y}_x, \forall x \in \mathscr{X}$	Ensemble de tuiles de sortie qui nécessitent la tuile d'entrée x
α	Durée nécessaire pour pré-charger une tuile d'entrée
β	Durée nécessaire pour calculer une tuile de sortie
$\mathscr{N} = \{1, \ldots, \mathrm{N}\}, \mathrm{N} \in \mathbb{N}^*$	Nombre total de pré-chargements
$(p_i)_{i \in \mathscr{N}} = (d_i, b_i, t_i)$	Séquencement des pré-chargements (tuile, buffer, date début de pré-chargement)
i	Une étape du pré-chargement
$(d_i)_{i \in \mathscr{N}}$	Ensemble de tuiles d'entrée pour l'étape de pré-chargement i ($d_i = x, \forall\ x \in \mathscr{X}$)
$(b_i)_{i \in \mathscr{N}}$	Destination (buffer) de tuiles d'entrée pour l'étape de pré-chargement i ($d_i = z, \forall\ z \in \mathscr{Z}$)
$(t_i)_{i \in \mathscr{N}}$	Date de début d'une étape de pré-chargement i
$(v_i)_{i \in \mathscr{N}}$	Date de fin d'une étape de pré-chargement i
$\mathscr{M} = \{1, \ldots, \mathrm{M}\}, \mathrm{M} \in \mathbb{N}^*$	Nombre total de tuiles de sortie calculées (M = Y)

$(s_j)_{j\in\mathscr{M}} = (c_j, u_j)$	Séquencement des calculs (tuile de sortie, date début du calcul)
j	Une étape du calcul
$(c_j)_{j\in\mathscr{M}}$	Séquence des calculs
$(u_j)_{j\in\mathscr{M}}$	Date de début d'une étape de calcul j
$(w_j)_{j\in\mathscr{M}}$	Date de fin d'une étape de calcul j
Δ	Temps total de traitement
lb_{N}	Borne inférieure sur N
ub_{N}	Borne supérieure sur N
lb_{Z}	Borne inférieure sur Z
ub_{Z}	Borne supérieure sur Z
lb_{Δ}	Borne inférieure sur Δ
ub_{Δ}	Borne supérieure sur Δ
N^*	Solution optimale en N
Z^*	Solution optimale en Z
Δ^*	Solution optimale en Δ
$r_{xy}, \forall (x,y) \in (\mathscr{X}, \mathscr{Y})$	Matrice d'incidence Tuiles d'entrée/Tuiles de sortie $\mathrm{X}\times\mathrm{Y}$ (matrice booléenne $0-1$)
$\mathrm{H_T} = (\mathrm{V_T}, \mathrm{E_T})$	"Tile Hypregraph" : hypergraphe de $\lvert\mathrm{V_T}\rvert$ sommets et $\lvert\mathrm{E_T}\rvert$ hyper-arêtes
$\mathscr{H}_{\mathrm{XY}}$	Matrice d'incidence correspondante à l'hypergraphe $\mathrm{H_T}$ (matrice booléenne $0-1$)
$\overrightarrow{\mathrm{G}}_{\mathrm{T}} = (\mathrm{V_T}, \mathrm{A_T})$	"Tile Graph" : graphe orienté de $\lvert\mathrm{V_T}\rvert$ sommets et $\lvert\mathrm{A_T}\rvert$ arcs
Λ	Première borne supérieure sur Δ : le "Big M", où $\Lambda = \alpha * \mathrm{X} + \beta * \mathrm{Y}$
$\mathscr{T} = \{1, \ldots, \mathrm{T}\}$	Intervalle du temps pour effectuer toutes les étapes de pré-chargements et de calculs
T	Seconde borne supérieure sur Δ : $\mathrm{T} = \alpha * \sum_{y\in\mathscr{Y}} \lvert\mathscr{R}_y\rvert + \beta * \mathrm{Y}$
$\mathscr{K} = \{1, \ldots, \alpha\}$	Intervalle du temps où une étape de pré-chargement s'effectue
$\mathscr{L} = \{1, \ldots, \beta\}$	Intervalle du temps où une étape de calcul s'effectue

3-PSDPP	3-objective Process Scheduling and Data Prefetching Problem
2-PSDPP	2-objective Process Scheduling and Data Prefetching Problem
MB-PSDPP	Minimum Buffers of 3-PSDPP Problem
MP-PSDPP	Minimum Prefetches of 3-PSDPP Problem
MCT-PSDPP	Minimum Completion Time of 3-PSDPP Problem
PSP	Prefetching and Scheduling Problem
DPP	Data Prefetching Problem
B-C-MCTP	Buffer-Constrained Minimum Completion Time Problem
P-C-MBP	Prefetch-Constrained Minimum Buffers Problem
P-C-MCTP	Prefetch-Constrained Minimum Completion Time Problem

Introduction

Contexte de ce travail

Les travaux de cette thèse ont été menés dans le cadre du projet AGIR de l'université Grenoble-INP. Ils ont été réalisés dans l'équipe Conception et Test de Systèmes embarqués (CTSYS) du Laboratoire de Conception et d'Intégration des Systèmes (LCIS) à Valence en collaboration avec le laboratoire de Techniques de l'Informatique et de la Micro-électronique pour l'Architecture des systèmes intégrés (TIMA) à Grenoble.

Contexte : problématique et motivations

Les systèmes de vision embarquée sont des systèmes numériques dotés de la capacité de traiter et de visualiser des sources d'image (appareils photo, vidéos, téléphones portables, etc). De part les progrès technologiques au niveau de l'intégration de ces systèmes et de part la complexité croissante des applications en traitement du signal et de l'image, il existe aujourd'hui une grande demande pour ces systèmes, avec de fortes contraintes "temps-réel" ainsi que des contraintes sur l'autonomie en énergie des circuits (limitation de la puissance disponible pour le calcul). Confrontés à ces exigences de plus en plus fortes, les concepteurs des circuits intégrés pour la vision embarquée cherchent de nouvelles méthodes pour obtenir des systèmes plus efficaces en termes de performance et de robustesse.

En outre, le volume des données impliquées dans les traitements d'images, combiné aux contraintes de performance, nécessitent d'accélérer les accès aux mémoires contenant ces images. Or pour certains traitements (traitements non-linéaires), les stratégies usuelles de gestion des données, basées sur les systèmes de caches génériques, sont inefficaces. C'est pourquoi il est souhaitable de mettre au point une méthodologie de conception automatisée et optimisée pour définir l'architecture mémoire de ces circuits, c'est-à-dire une chaîne d'outils logiciels génériques capable de générer, pour un traitement non-linéaire spécifique, une architecture mémoire adaptée et optimisée.

Dans ce contexte, un outil de conception dénommé Memory Management Optimization (MMOpt) a été proposé par les concepteurs de systèmes de vision embarquée : Mancini et al. (2012) [89]. Cet atelier de conception est un outil de génération de hiérarchies mémoires "ad-hoc" pour les traitements visuels avec des schémas d'accès non-linéaires à la mémoire afin d'optimiser la gestion des accès mémoire de ces traitements non-linéaires. Son principe consiste à décomposer les données de l'image d'entrée et celle de sortie en tuiles, puis d'ordonner les transferts de données entre l'unité de calcul et la mémoire principale ainsi que le séquencement des calculs pour optimiser le système.

Les premiers travaux d'optimisation sur l'outil MMOpt [90], sans avoir à appliquer la démarche de la recherche opérationnelle, sont assez encourageants, mais une analyse fine des résultats obtenus pose des défis d'optimisation à relever. Cette thèse décrit la phase d'optimisation du fonctionnement de circuits générés par l'outil MMOpt, dénommés Tile Processing

Unit (TPU)s, en faisant référence à des méthodes et outils d'optimisation issus de la recherche opérationnelle. Bien que la recherche opérationnelle fait son apparition dans divers domaines, allant du domaine militaire, de la science (informatique, mathématiques, physique, biologie, chimie, etc) aux applications industrielles (le transport, l'urbanisme, le commerce, les finances, les services sanitaires, etc), elle présente une nouvelle méthodologie pour aborder les problèmes qui découlent de la conception de circuits intégrés dédiés aux applications multimédia et à la vision embarquée, en particulier le traitement du signal et de l'image.

Objectif de la thèse pour l'outil MMOpt

Prolongeant des travaux existants, cette thèse aborde la problématique d'optimisation visant l'amélioration des enjeux de performance, de consommation d'énergie et de coût pour les TPUs générés par l'outil MMOpt. Au travers de cette étude, nous mettons en œuvre des méthodes issues de la recherche opérationnelle pour la modélisation ainsi que la résolution de la problématique d'optimisation engendrée.

En particulier, nous nous intéressons :

- à formaliser les problèmes d'optimisation apparaissant dans ce contexte en proposant des modèles pertinents (les données d'entrées et de sorties, les contraintes liées et les objectifs à optimiser) qui les traduisent en des problèmes mathématiques équivalents ;
- d'étudier les modèles proposés (au sens de l'analyse de la complexité et du calcul des bornes) ;
- de proposer un séquencement pour leurs résolutions, puis des routines d'optimisation adaptées à chacun de ces problèmes dans le but de fournir des solutions utiles à l'utilisateur de l'outil MMOpt.

Organisation du manuscrit

Le présent rapport se découpe en trois grandes parties subdivisées chacune en deux chapitres.

La première partie pose le cadre scientifique dans lequel s'inscrit notre travail. Elle décrit les concepts nécessaires à la compréhension de la suite du manuscrit.

Dans le premier chapitre, nous présentons les notions de bases relatives au domaine de la micro-électronique, en particulier à la vision embarquée, en mettant l'accent sur la conception de circuits intégrés ainsi que la problématique du Memory Wall (MW) liée à la hiérarchie mémoire dans le cadre des applications multimédia. La description de l'outil MMOpt qui est le contexte spécifique de la thèse sera donnée par la suite. Nous terminons ce chapitre en dressant les grandes lignes de la problématique d'optimisation qui sera abordée tout au long de notre travail. On est alors confronté à un problème d'ordonnancement multi-objectif que l'on appelle 3-objective Process Scheduling and Data Prefetching Problem (3-PSDPP). Ce problème considère l'optimisation de trois critères : performance, consommation d'énergie et coût.

Le deuxième chapitre donne les définitions et les concepts de base relatifs à l'optimisation combinatoire et à la recherche opérationnelle. Il rassemble également les techniques de modélisation et de résolution utilisées dans ces domaines pour traiter des problèmes d'optimisation issus de problématiques industrielles réelles. La dernière partie de ce chapitre donne quelques notions de base de l'optimisation combinatoire multi-objectif.

La seconde partie (chapitres 3 et 4) porte sur la phase d'analyse des modèles, y compris la formulation du modèle mathématique pour le problème 3-PSDPP, l'analyse de complexité de certaines de ses variantes mono-objectif ainsi qu'une revue de l'état de l'art.

Le troisième chapitre s'intéresse à l'étude du problème 3-PSDPP. Dans un premier temps, nous commençons par une spécification précise de ce problème, en citant la liste des différentes suppositions nécessaires imposées par le concepteur du circuit. Ensuite, nous proposons un modèle mathématique associé, c'est-à-dire les données d'entrée, les variables à déterminer, les contraintes et les objectifs à minimiser. Puis, nous présentons les différents sous-problèmes, mono- et bi-objectif, dérivés du problème 3-PSDPP. Dans un dernier temps, nous proposons des formulations mathématiques en termes de programmes linéaires en nombres entiers pour modéliser certaines variantes mono-objectif du problème 3-PSDPP.

Le quatrième chapitre donne d'abord une synthèse sur les principaux travaux de recherche liés à cette thèse, y compris les travaux antérieurs proposés par Mancini et al. [89] ainsi que certains problèmes voisins trouvés dans la littérature de recherche opérationnelle. Nous développons ensuite une liste de différentes bornes inférieures liées aux trois critères d'optimisation (surface, consommation d'énergie et performance). Enfin, nous présentons les principaux résultats de complexité établis pour certaines variantes du problème 3-PSDPP.

La dernière partie (chapitres 5 et 6) concerne la phase de résolution ainsi qu'une validation numérique des méthodes proposées.

Le cinquième chapitre présente les méthodes de résolution, exactes et approchées, que nous proposons pour le problème d'origine 3-PSDPP ainsi que certaines de ses variantes mono- et bi- objectif.

Dans le sixième chapitre, nous présentons les expérimentations numériques effectuées pour l'évaluation des performances de nos différentes approches de résolution qui mettent en évidence leur efficacité. En effet, les tests menés sont effectués sur des benchmarks disponibles dans la littérature ainsi que ceux fournis par le concepteur de l'outil MMOpt. À partir de cette étude expérimentale, nous donnons des éléments pour aider l'utilisateur de l'outil MMOpt à choisir sa meilleure approche parmi celles qui ont été présentées dans le chapitre précédent.

A la fin de ce rapport, nous présentons une conclusion générale qui résume notre travail tout en suggérant des pistes de réflexion pour des futurs travaux de recherche qui découlent de cette thèse.

Première partie

Notions Préliminaires et Contexte

« *Puisqu'on ne peut être universel et savoir tout ce qu'on peut savoir sur tout, il faut savoir un peu de tout. Car il est bien plus beau de savoir quelque chose de tout que de savoir tout d'une chose ; cette universalité est la plus belle.* »

Blaise Pascal, (1623-1662)

Chapitre 1

Brève Synthèse sur l'Outil de Conception MMOpt

Sommaire

1.1 Introduction

L'objectif de ce chapitre est de situer le premier contexte dans lequel s'inscrit cette thèse, soit le domaine de la vision embarquée et en particulier l'atelier de conception Memory Management Optimization (MMOpt).

Au cours de ce chapitre, nous donnons d'abord un bref aperçu sur quelques concepts de base relatifs à la vision embarquée, en particulier la problématique de Memory Wall (MW) liée à la hiérarchie mémoire dans le cadre des applications multimédia afin d'introduire le contexte dans lequel l'outil MMOpt s'inscrit. Nous présentons ensuite une vue idéalisée du flot de conception de cet outil, ainsi qu'une description détaillée de l'architecture cible générique de l'unité générée par l'outil MMOpt. Enfin, nous dressons les grandes lignes de la problématique d'optimisation qui sera abordée tout au long de la thèse.

La rédaction des différentes parties de ce chapitre est très inspirée des livres classiques d'électronique de base (le livre de W. Stallings [107], etc), certains polycopiés de cours tels que : [56], [47], [106], [73]et [86] ainsi que l'encyclopédie en ligne Wikipédia.

1.2 Introduction à la vision embarquée

Aujourd'hui, on entend de plus en plus parler de *systèmes embarqués*. Ces systèmes ont envahi notre vie quotidienne et sont désormais au cœur de nombreux types d'applications les plus omniprésentes. Nous pouvons en compter des dizaines croisées au cours d'une journée. Nous trouvons par exemple : téléphone portable, télévision, électroménager, systèmes audio, voiture, carte à puce, GPS, etc. Mais, concrètement, qu'est-ce qu'un système embarqué ?

Un Système Embarqué (SE), ou Embedded System (ES) en anglais, peut être défini comme un système électronique et informatique autonome, qui est dédié à une tâche bien précise.
Il s'agit d'un système contenant généralement un ou plusieurs microprocesseurs à basse consommation d'énergie ou des micro-contrôleurs destinés à exécuter un ensemble de programmes définis lors de la conception et stockés dans des mémoires généralement de type Read Only Memory (ROM). Plus simplement, il s'agit de systèmes électroniques intégrés à un autre objet. Contrairement à un environnement de travail classique de type Personnal Computer (PC), les SEs sont autonomes et ne possèdent pas d'entrées/sorties standards tels qu'un clavier ou un écran. Certains doivent répondre à des contraintes de temps réel pour des raisons de fiabilité et de rentabilité. D'autres ayant peu de contraintes au niveau des performances permettent d'économiser en énergie et/ou en mémoire (surface).

L'omniprésence des SEs dans notre vie est liée à la révolution numérique opérée dans les années 1970 avec l'avènement des processeurs. Ceci se confirme au travers de la fameuse *loi de Moore* [1]. Cette loi concerne ainsi l'évolution de la puissance des ordinateurs depuis plus de 50 ans. En effet, le raisonnement de Moore examine l'évolution de la fonction de coût des circuits, et considère la relation entre le coût moyen de production par composant et la complexité du circuit. Donc selon Moore tous les 18 mois il y a doublement du nombre de transistors, rendant les ordinateurs rapidement obsolètes. La Figure 1.1 montre l'évolution inexorable.

Ces évolutions technologiques de la micro-électronique ont rendu possible le déploiement d'applications de plus en plus complexes pour ces systèmes. En effet, cette hausse de la complexité des applications a également augmenté les besoins en termes de performance, de coût et de consommation énergétique, auxquels les concepteurs de systèmes embarqués doivent faire

1. Loi de Moore : loi relative à la croissance des performances des ordinateurs, fait en 1965 par Gordon E. Moore (un des co-fondateurs d'Intel).

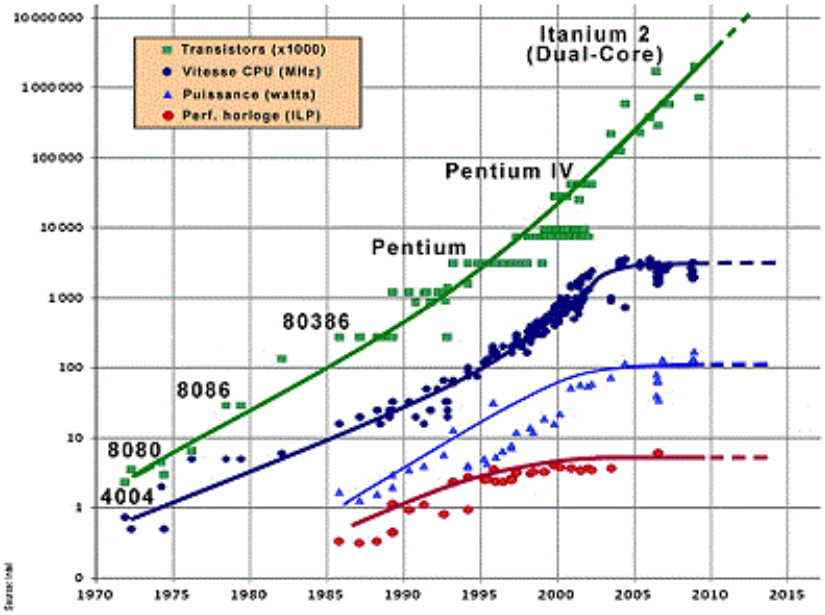


FIGURE 1.1 – Loi de Moore (1965 — 2015). *On constate depuis l'invention des processeurs multi-cœurs en 2001, c'est grâce à l'augmentation de la densité des transistors dans les CPUs que les performances des ordinateurs continuent d'augmenter et même de manière accélérée. (http ://www.astrosurf.com/luxorion/loi-moore.htm)*

face. Un défi majeur consiste alors à trouver le bon compromis entre ces trois contraintes technologiques.

Dans le cadre des travaux de cette thèse, nous nous sommes plus particulièrement intéressés aux SEs utilisés pour des applications multimédia et de traitement d'image embarqué. Une des spécificités techniques de ces systèmes est leur vision embarquée. Autrement dit, il s'agit de *systèmes de vision embarquée* (SVEs), ou *Embedded Vision Systems* (EVSs) en anglais, qui peuvent être défini comme des systèmes numériques dotés de la capacité de traiter et de visualiser une ou plusieurs sources d'images. Les téléphones intelligents, les tablettes, les consoles de jeux et les appareils photo appartiennent à cette catégorie, ainsi que des dispositifs plus inhabituels tels que les instruments de diagnostic médical avancé et les robots avec des capacités de reconnaissance d'objets, comme le montre la Figure 1.2.

FIGURE 1.2 – Exemple de SVEs (www.embedded-vision.com)

En se focalisant sur ce type de systèmes, leur conception nécessite de les décomposer en unités spécialisées soit dans les traitements (les unités de calcul) soit dans le stockage de l'in-

formation (les mémoires) du fait des contraintes technologiques liées aux performances des circuits ainsi qu'à leur processus de fabrication. La figure 1.3 [89] illustre une chaîne de traitement typique dans le cas de la vision bas niveau. Cette chaîne de traitement d'images (ou pré-traitement) regroupe l'ensemble des processus visant à améliorer les caractéristiques d'une image. Ces traitements bas niveau sont des étapes clés qui interviennent dans les deux premières étapes d'un système d'analyse d'image [23]. Dans ces traitements, des données matricielles intermédiaires (image, etc.) sont produites à partir des images d'entrées.

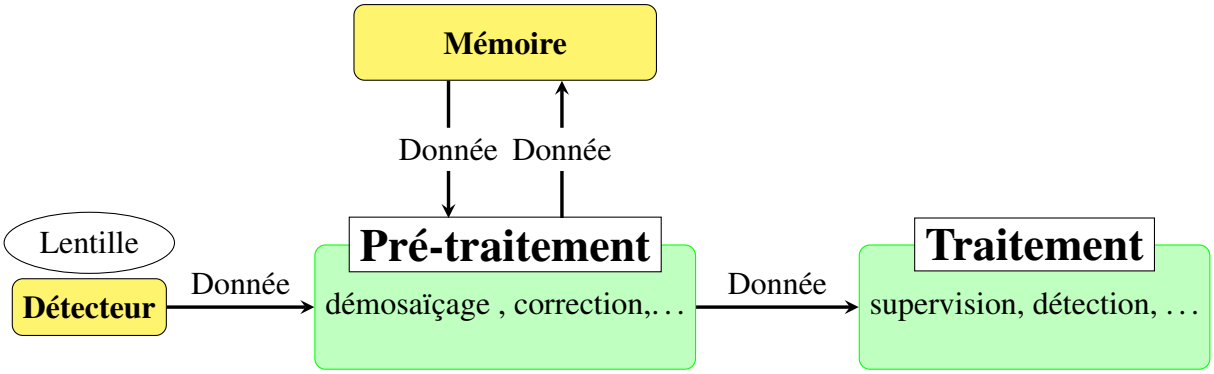


FIGURE 1.3 – Chaîne de traitement typique pour la vision bas niveau [89]

En revanche, la complexité de ces systèmes, vu le grand nombre d'éléments qui les composent et les relations entre ces éléments, pousse les architectes à trouver de nouvelles solutions plus efficaces que les mécanismes génériques implantés dans les processeurs généralistes pour augmenter l'efficacité des circuits. Afin de garantir cette efficacité, une telle solution technique est désormais nécessaire pour améliorer les performances de calcul tout en exploitant au mieux les différentes caractéristiques d'un SE. Une principale préoccupation qui ne doit pas être négligée est alors une *gestion efficace des données*.

Ainsi, la gestion des données est un enjeu majeur auquel les concepteurs de SEs doivent faire face car elle devient le goulot d'étranglement des systèmes complexes. Ce phénomène est décrit par le terme "Memory Wall".

Le concept du "Memory Wall" est la disparité croissante de la vitesse entre le Central Processins Unit (CPU) et la mémoire en dehors de la puce CPU. Une raison importante de cette disparité est la *bande passante* [2], ou *bandwidth* en anglais, de communication limitée au-delà des limites des puces, qui est également appelée *bandwidth wall*. Comme le montre la Figure 1.4, de 1986 à 2005, la vitesse du processeur s'est améliorée à un taux annuel de 55% alors que la vitesse de la mémoire ne s'est améliorée que de 10%. Compte tenu de ces tendances, la latence [3] de la mémoire devient un goulot d'étranglement (bottleneck) dans la performance de l'ordinateur. Cela signifie que la vitesse de calcul n'est plus alors limitée par les capacités du processeur mais par l'accès aux mémoires contenant les données à traiter.

Le phénomène du "Memory Wall" présente ainsi un enjeu majeur dans la gestion du transport et du stockage des données (au sens de la latence, de la bande passante et du coût) dans les systèmes de vision embarquée complexes. Par conséquent, de nombreuses recherches ont permis l'émergence de diverses solutions matérielles et logicielles fiables pour de grandes classes d'applications de calcul intensif en données. Parmi celles-ci se trouvent l'*exécution dans le désordre* (OOO pour *Out Of Order*), le *calcul dans la mémoire* (PIM pour *Processing In Memory*), ou le *pré-chargement* mais la plus importante reste la *hiérarchie mémoire*.

2. Bande passante, aussi appelé *débit* : est le taux auquel les opérations spécifiées peuvent être complétées.
3. Latence, aussi appelé *temps de réponse* : est le temps pour compléter une opération spécifique.

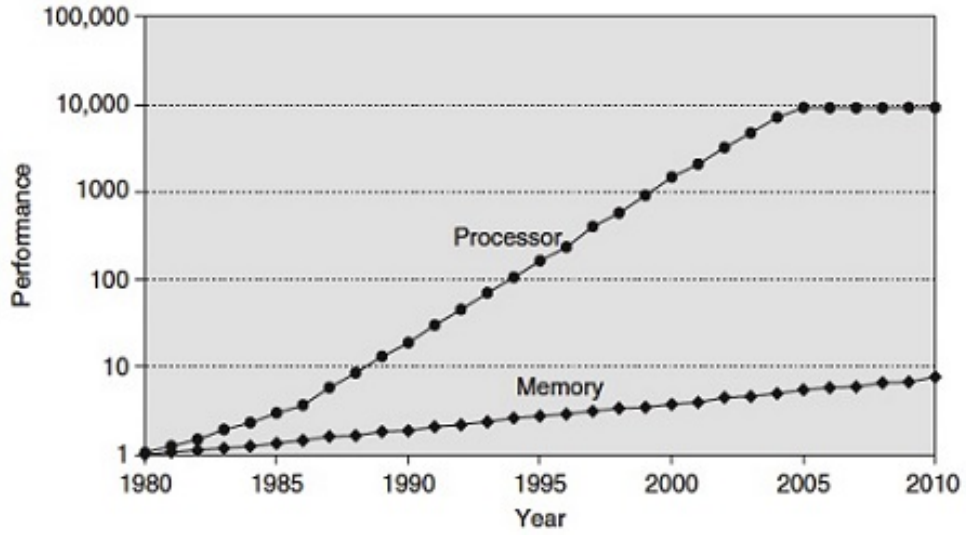


FIGURE 1.4 – Évolution asymétrique de la performance des processeurs et des mémoires depuis les années 1980

Avant d'aller plus loin, il convient de présenter, brièvement, les principaux composants de la *hiérarchie mémoire*. En effet, il existe plusieurs types de mémoires [4] qui se distinguent par leur mode d'enregistrement (électronique, magnétique, optique), leur capacité, leur rapidité (vitesse), leur prix (coût), le type de support, la densité d'information, la manière d'y accéder, le fait qu'elles soient volatiles ou non, etc. Il est alors d'usage de classifier les mémoires et de les hiérarchiser en les situant dans une représentation appelée *pyramide des mémoires*.

Comme le montre la Figure 1.5, les mémoires sont organisées de manière hiérarchique. En effet, au niveau supérieur se trouvent les *registres* du processeur (cellules mémoires internes au CPU). Ils sont suivis d'un ou de plusieurs niveau(x) de *mémoires cache*, libellés L1, L2, etc. Vient ensuite la *mémoire centrale*, généralement composée de Random Access Memory (RAM) dynamique, Dynamic Random Access Memory (DRAM) en anglais. Ces différents niveaux sont internes au système. Finalement, nous trouvons les *mémoires de masse* (appelées également mémoires physiques).

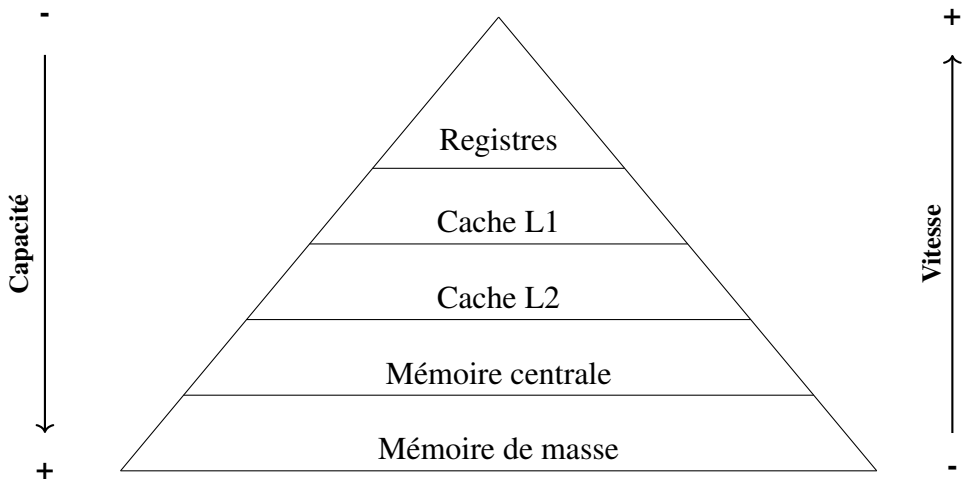


FIGURE 1.5 – Pyramide des mémoires

4. Mémoires : sont des circuits intégrés permettant d'enregistrer des mots binaires étant généralement des instructions d'un programme ou des données à sauvegarder temporairement.

Ces différents éléments sont ordonnés en fonction des deux critères suivants :

— **Temps d'accès** ou vitesse : c'est le temps nécessaire à l'écriture ou à la lecture d'un octet ;

— **Capacité** ou taille : c'est le volume global d'informations (en bits) que la mémoire peut stocker ;

D'une manière générale, quand on s'éloigne du CPU vers les mémoires auxiliaires, on constate que le temps d'accès et la capacité des mémoires augmentent, mais le coût par bit diminue. Une mémoire parfaite ou idéale serait une mémoire qui permette de stocker un maximum d'informations (de grande capacité) et qui possède un temps d'accès très faible afin de pouvoir travailler rapidement sur ces informations. Cependant il se trouve que les mémoires de grande capacité sont très lentes et que les mémoires rapides sont très coûteuses et de capacité réduite. Et pourtant, la vitesse d'accès à la mémoire détermine dans un premier temps les performances d'un système.

Plusieurs architectures ont été proposées pour la *hiérarchie mémoire*, à savoir l'*architecture Von Neuman* — définie comme une conception des processeurs qui utilise une structure de stockage unique pour conserver à la fois les instructions et les données demandées ou produites par le calcul — et l'*architecture Harvard* qui sépare physiquement la mémoire de données et la mémoire programme, où l'accès à chacune des deux mémoires s'effectue via deux bus distincts.

La mise en place d'une *hiérarchie mémoire* s'appuie sur une imbrication entre plusieurs mémoires permettant de fournir les données à l'unité de calcul sans temps mort afin de minimiser la latence et le coût. Mais, il est nécessaire de savoir comment dimensionner ces différentes mémoires qui permettent de stocker les données que l'on souhaite copier, quand et où les copier. En effet, la vitesse de réaction de la RAM et celle du bus système sont insuffisantes pour pouvoir répondre rapidement aux commandes du processeur. L'idée générale est d'anticiper les accès mémoire des unités de calcul en disposant de mémoires locales plus petites et plus rapides à côté du processeur, à savoir les *mémoires caches*, dans lesquelles sont stockées les données les plus réutilisées par le processeur en provenance de la mémoire externe. C'est la raison pour laquelle les données y sont lues par blocs mis à portée du processeur par l'entremise de la *mémoire cache* (voir Figure 1.10).

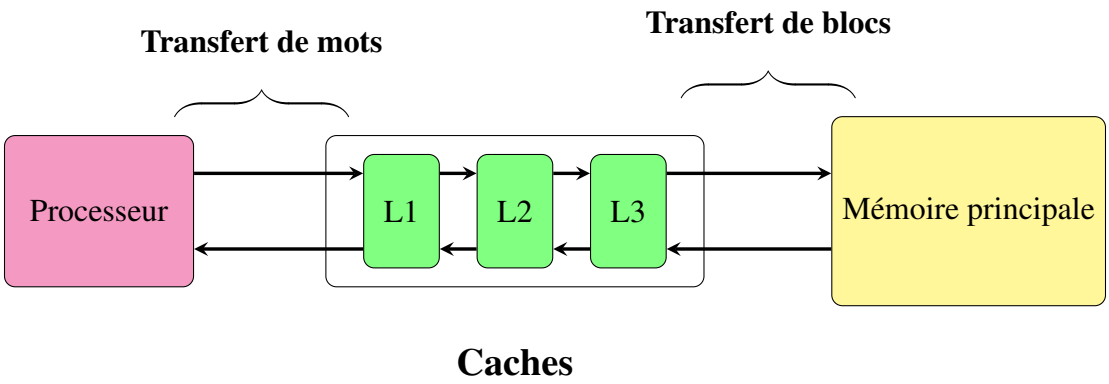


FIGURE 1.6 – Mémoire cache et mémoire principale [107]

Lorsque le microprocesseur demande l'accès à une information particulière de la mémoire centrale, ce n'est pas seulement celle-ci qui est transférée dans la *mémoire cache* mais un bloc de mémoire entier (souvent quelques dizaines d'octets). Le cache contient ainsi plusieurs blocs qui sont remplis au fur et à mesure des besoins du CPU. Quand tous les blocs sont remplis, un système de contrôle du cache détermine quel bloc va être écrasé par le nouveau (p.ex. le bloc le plus ancien dans le cache). En effet, les références aux données et surtout aux instructions ne sont pas indépendantes. Les programmes ont alors tendance à réutiliser les données et les instructions qu'ils ont utilisées récemment.

Ce principe utilise les phénomènes de :

— **Localité temporelle** : les données récemment référencées par le processeur seront probablement utilisées dans un futur proche (p.ex. boucles), ce qui signifie qu'il faut seulement stocker la donnée qui a été récemment utilisée.

— **Localité spatiale** : les données dont les adresses sont proches les unes des autres auront tendance à être référencées bientôt (p.ex. instructions, images), ce qui signifie que si on charge de la DRAM la donnée dont on a besoin, il faut également charger les données voisines (un bloc de plusieurs données).

Dans chacun des cas, le système contribue au gain de performance en plaçant ces données à un niveau supérieur de la *hiérarchie de mémoire*. Par contre, les transferts entre les niveaux coûtent très cher en temps d'exécution.

En résumé, les performances globales de la *mémoire cache* dépendent de plusieurs facteurs corrélés, tels que : la taille de la *mémoire cache* et l'organisation des niveaux (caches multi-niveaux), les méthodes d'association des lignes entre mémoire centrale et cache (set associative, direct-mapped, etc), la rapidité d'accès à la bonne ligne dans le cache, les algorithmes de *pre-fetching*, la méthode de remplacement des lignes, etc.

Bien que les caches classiques standard aient de meilleures performances dans le cas des applications qui y sont adaptées, dans le sens où elles garantissent le chargement d'une donnée tout en améliorant la rapidité des programmes et en réduisant l'utilisation du bus de données (la pénalité d'accès mémoire aux informations), elles restent inefficaces dans le contexte des systèmes de vision embarquée. En effet, la conception de ce type de systèmes face à la problématique du "Memory Wall" présente de nombreux défis, par exemple en termes de coût de conception, de consommation d'énergie et de performance. En outre, elle a cette difficulté intrinsèque lorsqu'on veut traiter des images et des fichiers vidéo de grande taille, où la latence des mémoires contenant ces traitements est très élevée, dans un temps très court.

C'est pourquoi il semble plus intéressant de proposer des méthodes de gestion des données adaptées à ces classes d'applications et plus efficaces que les mécanismes génériques. Parmi de nombreuses autres, une telle solution a été proposé par Mancini et al. (2012) [89]. Cette solution est présentée dans la section suivante (Section 1.3).

1.3 Synthèse sur l'atelier de conception MMOpt

Dans cette section, nous décrivons en détails l'atelier de conception de système de vision embarquée MMOpt qui présente le premier contexte de cette thèse. Les différentes parties de cette section ont été rédigées en s'inspirant du chapitre 3, intitulé *Gestion des données pour les noyaux non-linéaires dans un flot de conception HLS*, de la HDR de S. Mancini (2013) [88], ainsi que les différentes publications liées à ce sujet ([89], [90]).

1.3.1 Contexte

Dans les systèmes de vision embarquée, le traitement d'images avec de grandes données provoque une latence très élevée des mémoires contenant ces images. En effet, le volume de ces données, combiné aux contraintes de performance, nécessite d'accélérer les accès aux mémoires contenant ces images. Lorsque le traitement sur l'image a une certaine régularité, aussi appelé noyau linéaire [5], il peut être fait assez facilement, soit par des conceptions complémentaires ad-hoc, soit par des stratégies plus génériques (mémoires caches). Cependant, lorsque les

5. Traitement linéaire : est un algorithme dont les références au signal d'entrée sont des fonction linéaires.

accès à la mémoire ne présentent pas une telle régularité, avec des schémas d'accès aux mémoires non-linéaires, aussi appelés noyaux non-linéaires [6] (voir Figure 1.7), la réalisation des performances souhaitées est un véritable défi, souvent impossible à satisfaire. Les systèmes de caches génériques sont alors peu efficaces.

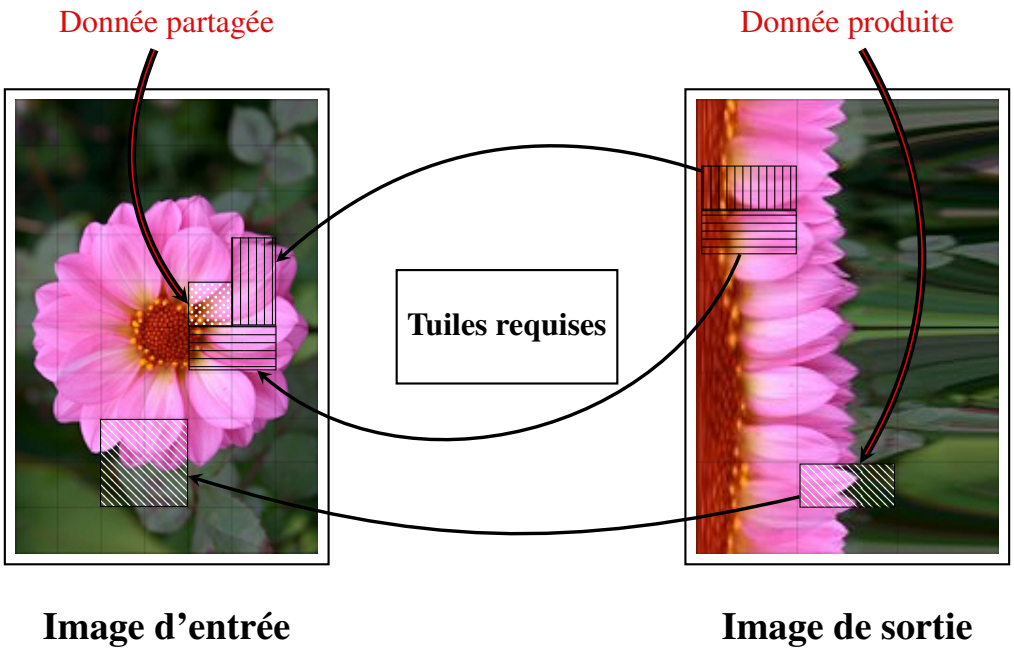


FIGURE 1.7 – Exemple d'accès mémoire d'un noyau non-linéaire : Transformation Polaire

La Figure 1.7 illustre un exemple des accès mémoires effectués par une transformation relativement simple, à savoir la *transformation polaire*. D'autres exemples seraient un noyau de rectification en stéréoscopie 3D, une correction fisheye, etc.

Typiquement, les noyaux calculent un élément de données de sortie à partir d'un ensemble de données d'entrées. Cet ensemble, dénommé *empreinte*, peut être de forme régulière et invariante (cas d'un filtre convolutif) ou bien sa forme peut varier selon les coordonnées de l'élément calculé tel que illustré. En effet, dans cet exemple, le calcul d'un pixel de sortie nécessite un ensemble de pixels d'entrée dont les coordonnées sont calculées à l'aide de fonctions trigonométriques.

La Figure 1.7 représente l'ensemble des pixels d'entrée nécessaires au calcul de différents blocs de l'image de sortie (à droite), où à chaque zone identifiée par un motif de remplissage, correspond une zone de motif assorti de l'image d'entrée (à gauche). En outre, les empreintes de zones de sorties voisines se recouvrent plus ou moins. Ceci peut être expliqué par le partage des données d'entrée au cours du calcul des sorties.

A partir de cet exemple, nous pouvons voir que :

— Il n'existe pas de relation linéaire entre les positions et surfaces des empreintes ;

— Le séquencement des tuiles de sortie qui minimise la quantité de données chargées (partage des empreintes maximum) n'est pas linéaire.

En conséquence, il n'est pas possible d'utiliser les méthodes classiques, à savoir les caches standards ou bien les méthodes issues des modèles dits "polyédriques" [43], pour effectuer les préchargements des données d'entrée. Il est ainsi nécessaire de définir des stratégies de gestion des données qui soient adaptées à ce type de traitement. C'est pourquoi, un système

6. Noyau non-linéaire : est un algorithme dont les références au signal d'entrée sont des fonctions non-linéaires d'indices de boucle : fisheye, reconstruction 3D, autofocus, etc.

de cache intelligent dénommé MMOpt a été proposé par Mancini et al. (2012) [89]. Cet atelier de conception de système de vision embarquée a pour objectif d'optimiser la gestion des accès mémoire des traitements de type *noyaux non-linéaires*.

1.3.2 Principe et Stratégie

L'atelier de conception MMOpt est un générateur de *hiérarchies mémoires*. C'est un outil logiciel de conception de caches ad-hoc pour un type particulier de traitements d'images, dit *noyaux non-linéaires* qui sont des traitements sur des tableaux multi-dimensionnels de pixels avec des accès mémoires complexes mais aussi déterministes et indépendants des données. Il a pour objectif d'optimiser les gestions de transferts et d'accès mémoire des traitements en amont de la synthèse High Level Synthesis (HLS) [7] (réduire la latence mémoire).

La figure 1.8 représente une vue idéalisée du flot de conception de l'outil MMOpt qui se décompose en deux grandes phases : l'analyse et l'optimisation.

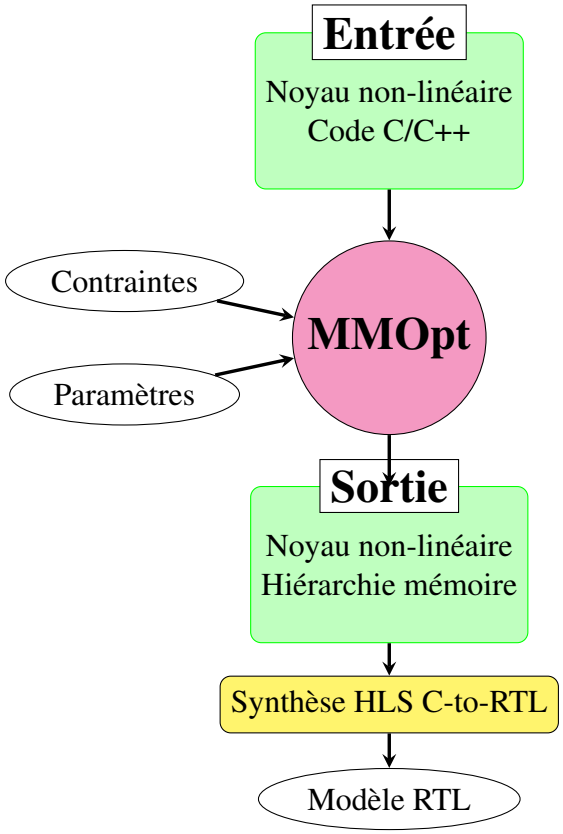


FIGURE 1.8 – Vue générale du flot de conception [89]

L'outil MMOpt prend en entrée un code C/C++ du traitement synthétisable par HLS et génère une matrice d'incidence qui relie chaque zone de l'image d'entrée à la ou les zones de l'image de sortie qui l'utilise. Les optimisations sont ensuite réalisées à partir de cette matrice d'incidence. Contrairement aux méthodes issues des modèles polyédriques [43], l'outil MMOpt repose sur des méthodes d'optimisation combinatoire.

L'outil MMOpt réalise les optimisations en ciblant un canevas typique d'une implémentation matérielle d'une unité de calcul de noyau, dénommée TPU. Cette unité intègre le noyau

7. High Level Synthesis (HLS) : est une synthèse de haut niveau qui consiste à générer une architecture de traitement et son séquencement à partir d'une description fonctionnelle du traitement, purement séquentielle.

original auquel sont accolés des mécanismes optimisés de gestion de données et du séquencement des calculs. L'outil MMOpt produit alors un code optimisé qui est conçu pour être synthétisé par un outil de HLS standard comme CatapultC. Finalement, le code résultant est l'entrée d'un outil de HLS classique pour générer un circuit Register Transfer Level (RTL).

L'outil MMOpt se base sur la technique de *tuilage* qui consiste à séparer l'espace d'itération en zones appelées des *tuiles*. Une *tuile* est un ensemble de pixels, suffisamment petit pour s'adapter à la mémoire locale et qui peut être traitée de manière indépendante. En effet, l'outil MMOpt consiste à transformer une image en une image découpée en tuiles (Mancini et al. [89]). On accroît ainsi la localité spatiale et temporelle des accès mémoire. Par exemple, une image sera séparée en petites images, de taille fixe, et chaque petite image sera traitée par le bloc de calcul avant de passer à la suivante. Ainsi, au lieu de parcourir l'image horizontalement puis verticalement, on parcourt l'image par tuiles, lesquelles sont parcourues horizontalement puis verticalement, comme l'illustre la Figure 1.9.

(a) (b)

FIGURE 1.9 – Parcours d'une zone mémoire : classique (à gauche Fig. 1.9a) et avec tuilage (à droite Fig. 1.9b). Ici, l'espace a été divisé en quatre tuiles de 4×4 pixels, qui sont encadrées en rouge sur le schéma de droite.

On calcule ainsi les valeurs de sortie tuile par tuile, l'ordre de calcul des tuiles étant un paramètre variable. Un parcours linéaire consiste à calculer l'ensemble des tuiles de sortie de gauche à droite et de haut en bas. Pour améliorer la gestion du flux de tuiles demandées par le bloc, l'outil MMOpt a besoin d'une étape réalisée hors-ligne qui consiste à donner, pour le bloc en question, les lois d'accès aux données d'entrées en fonction des données de sortie à calculer. Cette étape est effectuée grâce à l'unité de calcul TPU produite par l'outil MMOpt.

À titre d'exemple, dans le cas où l'on souhaite effectuer une symétrie centrale sur une image, pour chaque pixel de sortie, on aura besoin d'un pixel, situé à la position opposée par rapport au centre de l'image, sur l'image d'entrée. Grâce à ces données, l'outil MMOpt connaît à l'avance l'ensemble des accès qui vont devoir être effectués, et c'est ainsi qu'il peut optimiser la gestion de ces accès.

1.3.3 Architecture cible

L'architecture de l'unité de traitement TPU produite par l'outil de synthèse après l'utilisation de l'outil MMOpt est illustrée sur la Figure 1.10. Dans cette architecture, le TPU se compose :

- d'une unité de traitement qui implémente le noyau, appelé Processing Engine (PE) ;
- d'un contrôleur de buffers (mémoires locales) où sont stockées des tuiles d'entrées ;
- d'une unité de préchargement, appelée Prefetching Unit (PU), qui charge des tuiles d'entrée depuis la mémoire centrale.

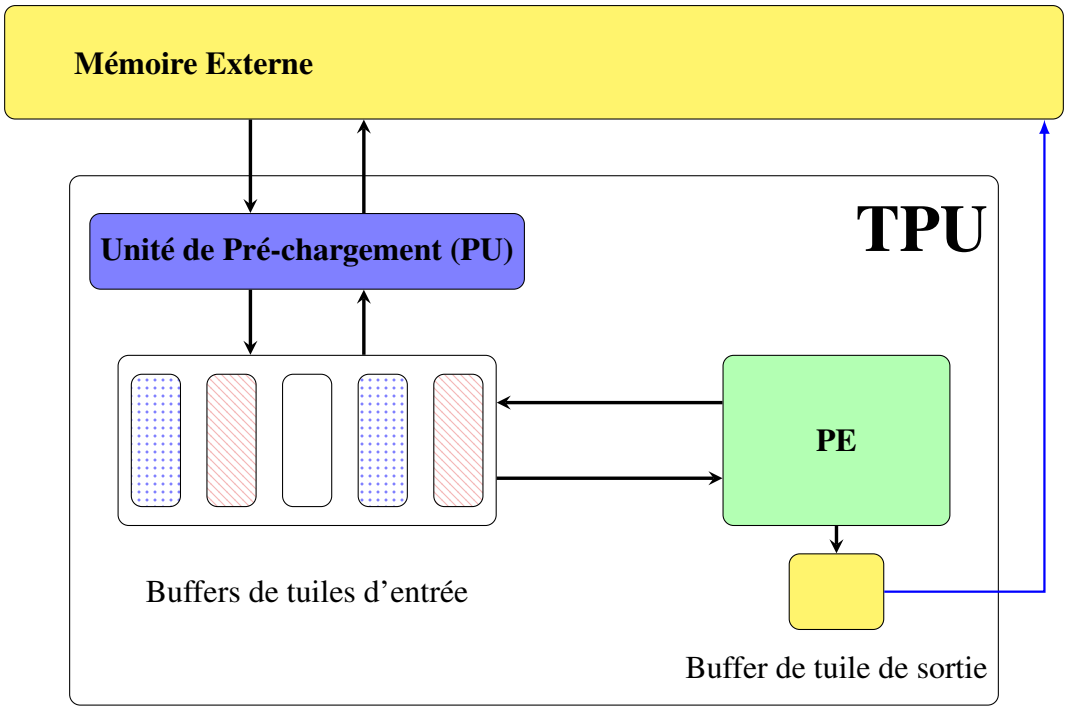


FIGURE 1.10 – Architecture cible simplifiée d'une unité du traitement TPU

Dans le TPU, les images d'entrée et de sortie sont découpées en tuiles. Le calcul d'une tuile de sortie nécessite en général plusieurs tuiles de l'image d'entrée. Il est alors impératif que ces tuiles soient accessibles dans les buffers internes du système au moment du calcul. Les unités, PU et PE, fonctionnent en parallèle de façon à recouvrir les calculs et les communications. Le séquencement des calculs des tuiles de l'image de sortie, des transferts depuis la mémoire centrale (ordre des chargements) et le placement des données en mémoire locale (destination — buffers internes — des tuiles d'entrée) sont ainsi calculés par MMOpt à partir de la matrice d'incidence.

Un séquencement type du fonctionnement d'une unité de traitement TPU générée par MMOpt est décrit dans la figure 1.11.

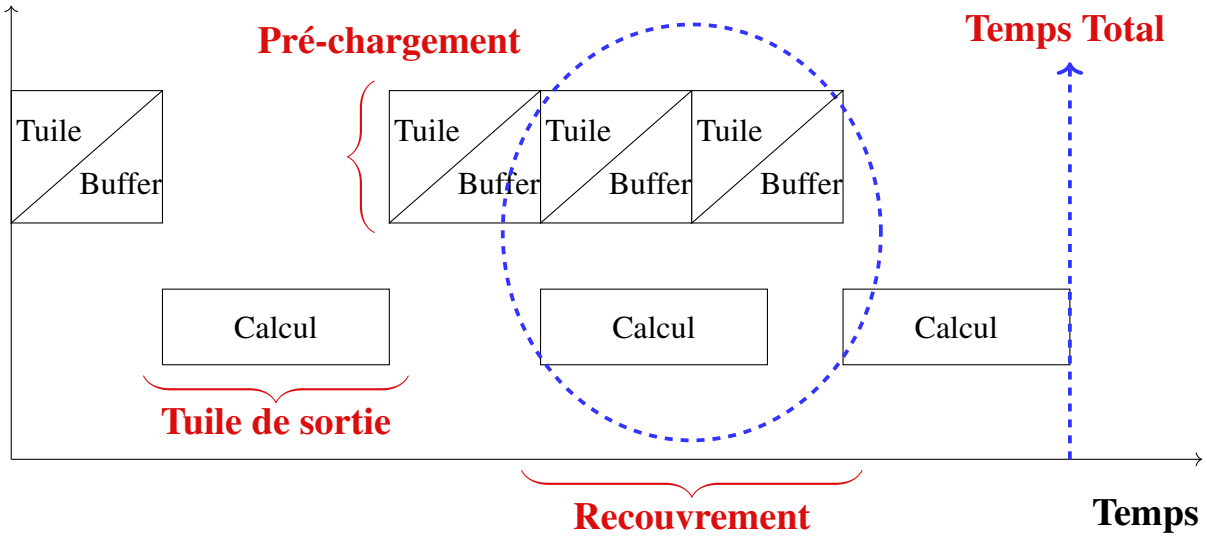


FIGURE 1.11 – Séquencement type d'un TPU

De façon à paralléliser les calculs de tuiles de sortie et les transferts depuis la mémoire

externe (les chargements de tuiles d'entrée), le traitement est découpé en phases pendant lesquelles : les *calculs* (tuile de sortie) soit, recouvrent les *préchargements* (tuile d'entrée, buffer), soit les débordent, et en cas de besoin, les *calculs* peuvent être retardés en insérant des "temps morts", aussi appelés *calculs fictifs* [89, 90].

1.4 Problématique d'optimisation

L'atelier MMOpt vise à optimiser des systèmes de traitement de données en faveur d'une synthèse HLS. Il s'intéresse aux noyaux non-linéaires dont l'objectif est d'optimiser la gestion des accès mémoire de ces traitements visant l'amélioration des enjeux de performance, de consommation d'énergie et de coût.

Comparativement à un cache classique, l'atelier MMOpt est plus efficace en termes qualitatifs car il supporte mieux de grandes latence à la mémoire centrale.

L'optimisation du fonctionnement des TPUs générés par l'outil MMOpt engendre une problématique d'optimisation riche. Plus exactement, nous sommes face à un problème d'optimisation multi-objectifs, à savoir minimiser :

— La quantité de mémoire interne (en termes de nombre des buffers de l'unité obtenu) qui représente en grande partie la surface du circuit : encombrement et coût de production ;

— Le temps total de traitement (performance et/ou contrainte temps-réel) ;

— mais aussi réduire le trafic depuis la mémoire centrale (en termes de nombre de préchargements effectués) qui représente la consommation d'énergie du circuit produit.

Nous considérons alors dans cette étude le problème d'ordonnancement multi-objectif, dénommé 3-PSDPP, qui considère l'optimisation de ces trois critères.

Les premiers travaux sur MMOpt [89] ont proposé une méthode d'optimisation qui s'avère meilleure qu'un ordonnancement linéaire classique qui parcourt les tuiles de sortie l'une après l'autre sans avoir à y ajouter des "temps morts", mais l'analyse fine des résultats montre qu'il est encore possible d'optimiser certaines métriques. Par exemple, le processus d'optimisation ne tient pas assez compte de la durée de vie des tuiles d'entrée dans les buffers internes. En effet, il existe de nombreuses situations où une tuile est préchargée inutilement depuis la mémoire externe alors qu'elle est encore présente dans un buffer. Il est alors encore possible de mieux utiliser les ressources matérielles et réduire le trafic vers la mémoire externe.

Dans ce contexte, pour rendre le fonctionnement des TPUs générés par l'outil MMOpt plus efficace, l'objectif de la thèse est d'aborder la problématique des accès mémoires visant l'amélioration de trois enjeux électroniques cités ci-avant, coût, énergie et performance, tout en respectant un certain nombre de contraintes dues à l'électronique. Les techniques utilisées sont issues de l'optimisation combinatoire et de la RO.

1.5 Conclusion

Ce premier chapitre présente une vue d'ensemble des concepts fondamentaux liés au premier contexte de cette thèse. Nous avons en effet brièvement présenté quelques notions de base liées au domaine de la micro-électronique, en particulier à la vision embarquée. Nous avons ensuite introduit la notion de "hiérarchie des mémoires" ainsi que la problématique du "Memory Wall", en particulier dans le contexte de la conception des systèmes de vision embarquée. Enfin, nous avons décrit l'atelier de conception MMOpt, dont l'étude fait l'objet de cette thèse,

ainsi que la problématique d'optimisation engendrée 3-PSDPP. Cette problématique sera abordée tout au long de cette thèse en utilisant des méthodes d'outils d'optimisation et la RO qui seront détaillées dans le prochain chapitre.

Pour en savoir plus sur le domaine de la vision embarquée, en plus des références citées dans ce chapitre, le lecteur pourra se référer à différents sites web, en particulier le site : https ://www.embedded-vision.com, http ://www.vision-systems.com, http ://www.argondesign.com, http ://www.embeddedvisionsystems.it, etc.

Chapitre 2

Qu'est-ce que la Recherche Opérationnelle ?

Sommaire

2.1 Introduction

Ce chapitre vise à décrire le second contexte de notre étude, soit le domaine de la Recherche Opérationnelle (RO) et de l'Optimisation Combinatoire (OC). L'objectif est de donner une vue d'ensemble de ce qu'est la RO, ainsi que quelques notions sur l' OC.

Dans un premier temps, nous présentons l'apparition de la RO, ses domaines d'application, et sa démarche scientifique. Ensuite, nous rappelons les concepts de base relatifs aux outils de modélisation et à la théorie de la complexité. Puis, nous donnons un aperçu sur les approches classiques de résolution. Enfin, nous introduisons brièvement l'Optimisation Combinatoire Multi-Objectif (OCM) en exposant les principales définitions et propriétés. Ce chapitre est inspiré de livres classiques de la RO tels que le livre blanc de la RO, édité par la ROADEF (Recherche Opérationnelle et Aide à la Décision Française) [102] et les livres de M. Sakarovitch [103, 104], de M.R. Garey et D.S. Johnson [48], de J.C Fournier [46], etc.

2.2 La Recherche Opérationnelle : origine et démarche

2.2.1 Naissance de la Recherche Opérationnelle

La véritable naissance de la RO a eu lieu pendant la seconde guerre mondiale grâce à des efforts conjugués d'un groupe de mathématiciens éminents (dont Von Neumann, Dantzig, Metropolis, Wald, Wiener et Bellman). Elle portait l'appellation Operationnal Research (OR) (UK) et Operations Research (OR) (US). Elle fait aussi partie des *aides à la décision* dans la mesure où elle propose des modèles conceptuels en vue d'analyser et de maîtriser des situations complexes pour permettre aux décideurs de faire les choix les plus efficaces. Cela signifie que la RO peut se résumer en deux mots clés : *modélisation* et *optimisation*.

C'est P.M.S. Blackett (1897 - 1974) qui est le fondateur de la RO. En effet, sur la demande du gouvernement britannique, Blackett a dirigé le premier groupe de la RO qui a pour but de résoudre certains problèmes tels que l'implantation optimale de radars de surveillance ou la gestion des convois d'approvisionnement. Puis, les Américains ont emboîté le pas.

Nous donnons ici une définition courte en anglais de la RO de telle sorte qu'elle résume d'une manière globale ce qu'est cette notion.

Qu'est-ce que la « Recherche Opérationnelle » ?

Operations Research (OR) is the use of mathematical models, statistics and algorithms to aid in decision-making. (source wikipédia)

En fait, il n'y a pas une définition réellement exhaustive de la RO car il n'y a pas vraiment un domaine spécifique ni une méthode de résolution unique mais une variété de nombreux champs d'applications. La RO est définie alors comme une approche scientifique pour produire les meilleurs décisions face à des problèmes de gestion de systèmes complexes. C'est une boîte à outils pour aborder sainement et sereinement les problèmes d'optimisation. Elle fait largement appel aux notions des mathématiques et de l'informatique.

La RO se situe au carrefour de différentes sciences et technologies. Elle a été utilisée premièrement dans des buts militaires. Cependant, elle a été adoptée dans d'autres domaines allant de la science (informatique, mathématiques, physique, biologie, chimie, etc) aux applications industrielles (le transport, l'urbanisme, le commerce, les finances, les services sanitaires, etc) afin d'optimiser l'utilisation des ressources disponibles pour obtenir des bénéfices, essentiellement économiques.

Dans cette thèse, nous abordons le domaine de la micro-électronique et plus précisément la conception des circuits intégrés (CCI) pour la vision embarquée (VE). Notre travail consiste en l'application des outils et des méthodes de la RO pour l'amélioration du fonctionnement d'un générateur de hiérarchies mémoires dénommé MMOpt [89] (cet outil a été décrit en détail dans le Chapitre 1), visant l'amélioration des enjeux électroniques : performance, consommation d'énergie et coût.

2.2.2 Le processus de la Recherche Opérationnelle

La Figure 2.1 représente un schéma général de la démarche scientifique de la RO et d'aide à la décision (AD).

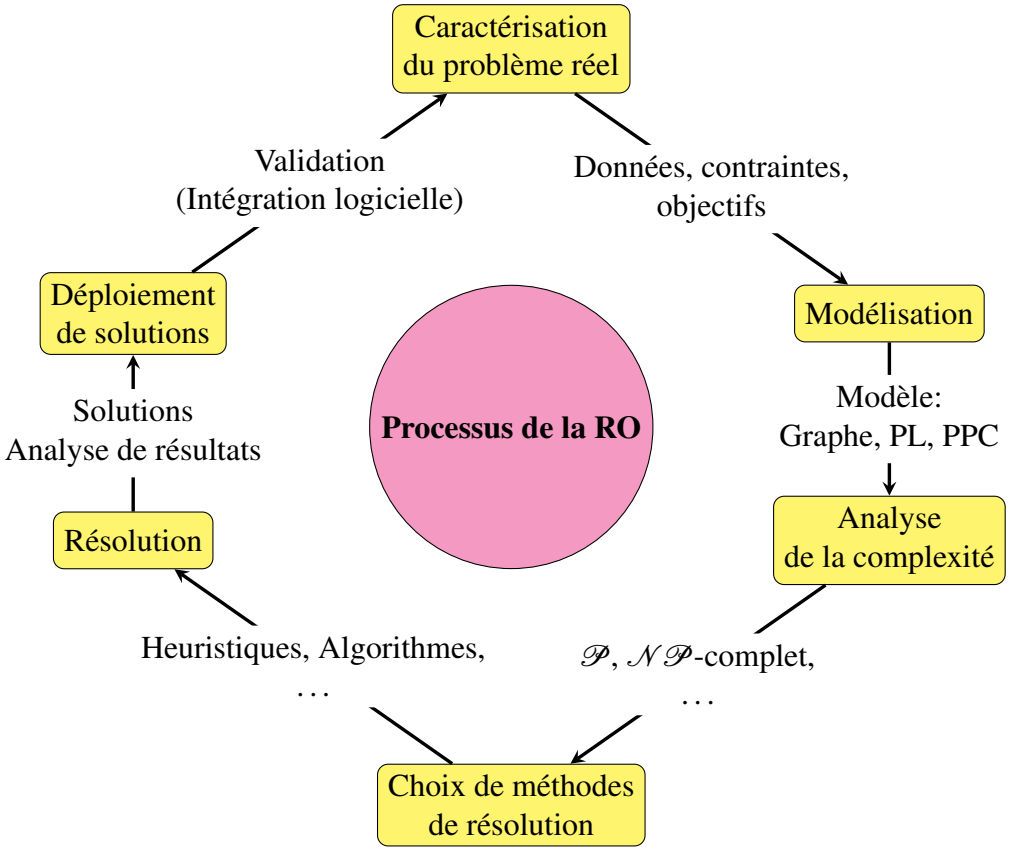


FIGURE 2.1 – Processus de la RO et Aide à la Décision

De manière générale, l'approche de la RO face à un problème du monde réel consiste à :

a) **Élaborer un modèle** : un *modèle* est une représentation idéalisée du problème réel sous la forme d'un problème mathématique équivalent en termes de données, contraintes et objectifs. Un même problème peut être modélisé de plusieurs façons différentes. Dans ce cas de figure, il conviendra de trouver la modélisation en rapport avec les attendus du problème à l'aide de divers outils de la RO tels que la programmation linéaire (PL), la théorie des graphes (TG), la programmation par contraintes (PPC), etc (qui seront décrits dans la Section 2.3).

b) **Analyser sa complexité** : établir la difficulté du problème (facile ou difficile à résoudre) afin de choisir la méthode la plus efficace et la plus adaptée pour sa résolution. Les différents concepts de base de la théorie de complexité (TC) seront décrits dans la Section 2.4.

c) **Développer un algorithme de résolution** : chercher un algorithme pour fournir une solution. Dans la réalité, nous trouvons une solution à un modèle du problème. Une brève introduction aux méthodes de résolution (exactes ou approchées) sera présentée dans la Section 2.5.

d) **Évaluer la qualité des solutions produites** : une fois le problème résolu, il est impératif d'évaluer la pertinence des résultats obtenus et de les interpréter dans le monde réel (mise en œuvre opérationnelle). En cas d'échec, il est nécessaire de s'interroger sur les choix adoptés lors des différentes étapes précédentes.

2.3 Les outils de modélisation

2.3.1 La programmation linéaire

La Programmation Linéaire (PL), ou LP pour *Linear Programming* en anglais, est un outil très puissant de la RO permettant de modéliser et résoudre certains problèmes d'optimisation concrets. Mathématiquement, elle consiste en la recherche de l'optimum d'une *fonction linéaire* (appelée aussi *fonction objectif, fonction de coût* ou *fonction économique*) comportant plusieurs inconnues réelles (*variables de décision*) liées entre elles par des relations linéaires indépendantes appelées *contraintes* exprimées sous forme d'équations ou d'inéquations linéaires.

Historiquement, la PL a été introduite en 1947 par G.B. Dantzig, M. Wood et leurs collaborateurs au U.S. Department of the Air Force. Les premières applications se situaient dans le domaine militaire mais elles se sont rapidement déplacées vers l'industrie et la planification économique.

Qu'est-ce qu'un « Programme Linéaire » ?

Un programme linéaire (PL) est défini par le triplet suivant :

— Variables réelles : $x_1, x_2, \ldots, x_n$

— Fonction objectif linéaire (max ou min) : $z = \sum_{j=1}^{n} c_j x_j$

— Contraintes linéaires : $\sum_{j=1}^{n} a_{ij} x_j \ (\leq, \geq, =) \ b_j$

Un PL peut parfois être résolu par la méthode graphique. L'idée est de représenter graphiquement les contraintes et visualiser l'ensemble des solutions afin d'identifier une solution optimale. Une autre méthode de résolution d'un PL, développée en 1947 par G.B. Dantzig, est la méthode du « Simplexe ». Il s'agit d'une méthode algébrique basée sur la résolution de systèmes d'équations linéaires. Cet algorithme est facile à comprendre et à implanter. De plus, il est efficace en pratique même pour un nombre important de variables et de contraintes. Un livre de référence sur ce sujet est celui de Guéret et al. [57].

Pour voir concrètement en quoi consiste la modélisation d'un problème en termes d'un PL, nous considérons l'exemple 2.1 d'un problème de production présenté ci-dessous :

Exemple 2.1. *Une entreprise, spécialisée dans la fabrication de matériel informatique, dispose de trois ateliers se répartissant la fabrication de deux nouveaux types de carte mère :* CM_1 *et* CM_2*. Chaque carte possède un temps de fabrication différent suivant l'atelier où elle est fabriquée. De plus, chaque atelier a un temps limite de fabrication (par mois) au dessus duquel il ne peut plus produire.*

Les données de ce problème sont représentées dans le Tableau 2.1 :

Quelles quantités de CM_1 *et* CM_2 *doit fabriquer l'entreprise pour maximiser le bénéfice total venant de la vente de ces nouvelles cartes ?*

	Atelier A	Atelier B	Atelier C	Profit ($)/unité
CM_1	2 heures	1 heure	-	180
CM_2	1 heure	5 heures	3 heures	250
Capacité de fabrication (h)	150	210	190	-

TABLEAU 2.1 – Données du problème de production

Ce problème se modélise alors par le programme linéaire suivant, où les deux variables de décision x et y représentent les quantités produites pour chaque type de carte mère CM_1 *et* CM_2 *:*

$$\begin{cases} \max\ Z = 180x + 250y & (1) \\ 2x + y \leq 150 & (2) \\ x + 5y \leq 210 & (3) \\ 3y \leq 190 & (4) \\ x, y \geq 0 & (5) \end{cases} \tag{2.1}$$

Dans ce PL (2.1), la ligne (1) représente le profit total Z qui est le critère à optimiser, où max *signifie que ce critère doit être maximisé. Les lignes (2), (3) et (4) désignent les contraintes de fabrication pour les ateliers A, B et C. Les contraintes (5) précisent le domaine de définition des variables.*

Remarque 2.1. Extensions *:*

1. *Lorsqu'un problème d'optimisation peut être modélisé comme un programme linéaire dont les variables sont contraintes à être entières alors le programme est dit linéaire en nombre entiers : PLNE, ou ILP pour Integer Linear Programming en anglais.*
2. *Quand certaines variables seulement sont contraintes à être entières, on parle alors de Programmation Linéaire en Variables Mixtes (PLM), ou MIP pour Mixed Integer Programming en anglais.*

Remarque 2.2. *Lorsque le problème contient plusieurs objectifs à optimiser (≥ 2), on parle de l'Optimisation Multi-Objectif (OMO), ou MOO pour Multi-Objective Optimization en anglais. Cette notion sera abordée, en exposant les principales définitions et propriétés, dans la Section 2.6.*

2.3.2 La théorie des graphes

Une seconde brique de base de la modélisation est la Théorie des Graphes (TG). Cette théorie est un outil privilégié de résolution de nombreux problèmes concrets.

L'histoire de la TG débute avec les travaux de L. Euler (1707-1738) au XVIII^e siècle et trouve son origine dans l'étude du problème des sept ponts de "Königsberg" (en 1736), schématisée ci-dessous par la Figure. 2.3a. Un problème similaire, celui de la marche du cavalier sur l'échiquier, a été étudié par les mathématiciens français A-T. Vandermonde, P. Rémond de Montmort et A. de Moivre. Ensuite, un autre cas particulier, où on cherche à parcourir la ville en passant qu'une seule fois par un sommet, prit le nom de chemin hamiltonien d'après le mathématicien W.R. Hamilton. On accorde donc à L. Euler l'origine de la théorie des graphes parce qu'il fut le premier à proposer un traitement mathématique de la question, suivi par A-T. Vandermonde.

Au milieu du XIXe siècle, plusieurs mathématiciens s'intéressent à des travaux sur d'autres notions liées à la théorie des graphes. Parmi ceux-ci, nous citons A. Cayley qui développa la *théorie des arbres* [1], qui est un type particulier de graphe n'ayant pas de cycle. En 1852, le mathématicien F. Guthrie énonça l'un des problèmes les plus connus dans la théorie des graphes qui vient de la coloration de graphe, ainsi que A. De Morgan et A.B. Kempe. L'étude de ce problème entraîna de nombreux développements en théorie des graphes, par plusieurs autres mathématiciens et physiciens. Parmi eux, nous citons P.G. Tait, P.J. Heawood, F.P. Ramsey, ainsi que H. Hadwiger et P. Erdös.

Au-delà des années 1946, la TG a connu un développement intense relatif à des travaux consacrés pour la résolution des problèmes concrets tels que les travaux de H.W. Kuhn en 1955, suivi par les travaux de L.R. Ford et D.R. Fulkerson en 1956 et B. Roy en 1959. Parallèlement, C. Berge, qui introduit la notion de graphe parfait, a publié son premier ouvrage intitulé "Théorie des graphes et ses applications" en 1958 [19]. Cet ouvrage constitue une synthèse sur la TG.

Depuis, cette théorie a pris sa place, en subissant de très nombreux développements essentiellement dus à l'apparition des ordinateurs, au sein d'un ensemble plus vaste d'outils et de méthodes généralement regroupées sous l'appellation RO ou *mathématiques discrètes*.

La TG s'est alors développée dans diverses disciplines telles que la chimie, la biologie, les sciences sociales, l'informatique et les applications industrielles. Elle constitue l'un des instruments les plus efficaces en RO pour modéliser et résoudre une grande variété des problèmes concrets. En mathématiques, on retrouve les graphes dans la combinatoire, la théorie des ensembles, l'algèbre linéaire, la théorie des polyèdres, la théorie des jeux, l'algorithmique, les probabilités, etc. Comme livres de référence sur ce sujet, nous citons celui d'Aldous et al. [6] et celui de J.C Fournier [46].

De manière générale, un *graphe* permet de représenter des objets ainsi que les relations entre ces éléments (par exemple réseau de communication, réseaux routiers, interaction de diverses espèces animales, circuits électriques, etc).

Qu'est-ce qu'un « Graphe » ?

Un *graphe* G est un couple formé de deux ensembles (S, A), où

- S : un ensemble fini $\{s_1, s_2, \ldots, s_n\}$ dont les éléments sont appelés **sommets**.
- A : un ensemble $\{a_1, a_2, \ldots, a_k\}$ d'**arêtes** tels qu'à chaque arête a_i sont associés deux éléments de S, appelés **extrémités**.

Graphiquement, les sommets peuvent être représentés par des points et les arêtes par des traits reliant ces points. On distingue alors deux familles des graphes, *orientés* et *non orientés*, qui peuvent être définies de la manière suivante :

Un *graphe orienté* (Figure 2.2a) est un graphe dont les *arcs* sont orientées (fléchées ou paires ordonnées de sommets). On distingue alors le sommet origine de l'arête et son extrémité. En revanche, dans le cas où les sommets sont reliés par de simples *arêtes*, on l'appelle un *graphe non-orienté* (Figure 2.2b).

Plusieurs problèmes découlent de la TG tels que le *problème du plus court chemin*, le *problème du flot maximum* et le *couplage*, etc [46], [103]. Ces problèmes seront décrits en détail dans la Section 2.4.4.

À titre d'exemple, nous considérons ici le problème mathématique historique, qui fit la célébrité de la ville de "Königsberg" : le problème des 7 ponts de "Königsberg".

1. En théorie des graphes, un arbre est un graphe non orienté, acyclique et connexe dont le nombre de sommets excède le nombre d'arêtes d'une unité exactement

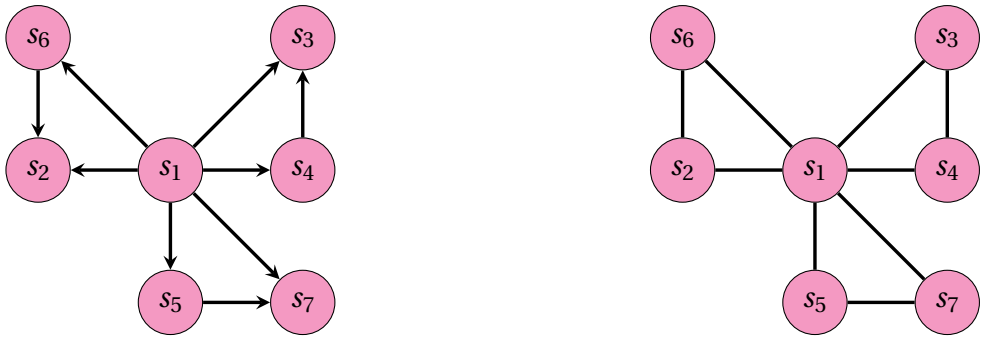


FIGURE 2.2 – Représentation graphique de graphe orienté (a) et non-orienté (b)

Exemple 2.2. Les 7 Ponts de "Königsberg"

La ville de "Königsberg" (aujourd'hui "Kaliningrad") est située sur les bords de la rivière "Pregel" (aujourd'hui "Pregolia") en Prusse orientale. Elle en occupe les deux rives ainsi que deux îles. Au XVIII^{e} *siècle, les 4 parties de la ville étaient réunies par 7 ponts, conformément au plan illustré par la Figure 2.3a. Le problème consiste à déterminer s'il existe ou non une promenade, à partir d'un point de départ au choix, permettant de passer une et une seule fois par chaque pont de la ville de "Königsberg", et de revenir à son point de départ, étant entendu qu'on ne peut traverser le "Pregel" qu'en passant par les ponts.*

L. Euler représenta ce problème à l'aide d'un graphe, donné par la Figure 2.3b, où à chaque portion de terre correspond un sommet (A, B, C, D) et à chaque pont une arête (1—7). L. Euler a démontré que le problème de sept ponts de "Königsberg" n'a pas de solution. C'est-à-dire, une telle promenade en empruntant une et une seule fois chaque pont n'existe pas. En effet, ce graphe possède 4 sommets de degré[2] *impair. Par conséquent, il n'existe pas de chemin eulérien.*[3]

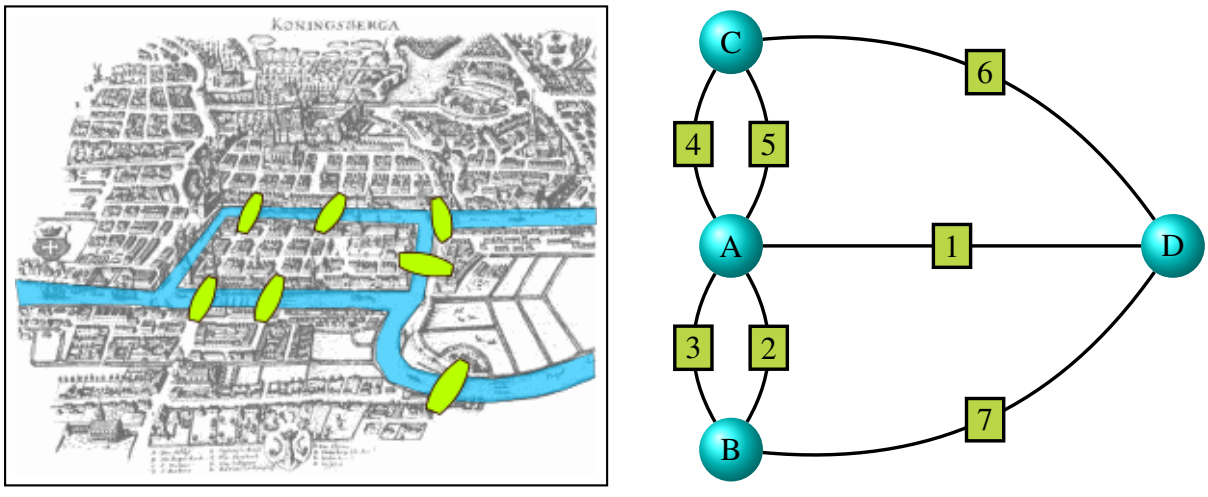


FIGURE 2.3 – La carte de "Königsberg" au temps d'Euler

2. Le degré d'un sommet d'un graphe est le nombre d'arêtes qui atteignent ce point
3. Théorème d'Euler (1736) : Un graphe connexe est eulérien si et seulement si chacun de ses sommets est incident à un nombre pair d'arêtes.

2.3.3 La programmation par contraintes

Une troisième brique de base de la modélisation est la Programmation Par Contraintes (PPC), ou CP pour *Constraint Programming* en anglais. C'est une technique mathématique, apparue à la fin des années 1980, pour modéliser et résoudre des problèmes combinatoires complexes issus de la programmation logique et de l'intelligence artificielle.

La PPC est pratiquement adaptée à la résolution des problèmes de décision. Tout problème de décision (et par extension tout problème de recherche) spécifié de cette manière est appelé problème de satisfaction de contraintes (PSC), ou CSP pour *Constraint Satisfaction Problem* en anglais.

Qu'est-ce qu'un « Problème de Satisfaction de Contraintes » ?

Un PSC est défini par un triplet (X, D, C) tel que :

- $X = \{X_1, \ldots, X_n\}$ est l'ensemble des variables (les inconnues) du problème.
- D est la fonction qui associe à chaque variable X_i son domaine $D(X_i)$, c'est-à-dire l'ensemble des valeurs que peut prendre X_i.
- $C = \{C_1, \ldots, C_k\}$ est l'ensemble des contraintes. Chaque contrainte C_j est une relation entre certaines variables de X, restreignant les valeurs que peuvent prendre simultanément ces variables.

De manière générale, modéliser un problème sous la forme d'un PSC consiste à identifier l'ensemble des variables X (les inconnues du problème), leurs domaines D et les contraintes C entre ces variables. Un *domaine* peut être défini par un ensemble fini, un intervalle ou des arbres. Une *contrainte* est relationnelle, déclarative et peut être définie en extension ou en intention. Elle peut être sous forme logique, arithmétique, explicite (n-tuples) ou complexe (global : all-different($X_1, \ldots, X_n$)). De plus, chaque contrainte peut avoir une arité quelconque (unaire, binaire ou n-aire). Dans la modélisation par un PPC, il n'y a pas qu'une formulation possible.

En revanche, résoudre un PSC revient à déterminer s'il est *satisfiable*. Autrement dit, un PSC est *satisfiable* s'il possède une solution, c'est-à-dire une affectation (une assignation) des valeurs $(V_1, \ldots, V_n)$ aux variables $(X_1, \ldots, X_n)$ de telle sorte que les valeurs soient dans les domaines $(V_j \in D_{X_j}, \forall j)$ et que toutes les contraintes (C_j) soient satisfaites. Un PSC est dit *consistant* s'il admet au moins une solution. Dans le cas contraire, il est dit *inconsistant* (ou *surcontraint*).

En effet, la résolution d'un PSC est un problème $\mathcal{NP}$-Complet dans le cas général. Les méthodes de résolution des PSCs sont génériques, c'est-à-dire qu'elles ne dépendent pas de l'instance à résoudre. Cependant, des techniques dédiées améliorent la résolution de différentes classes de problèmes. Dans ce contexte, plusieurs algorithmes ont été proposés. Parmi ceux-ci, nous citons l'algorithme Generate-And-Test (GAT), l'algorithme Simple Retour Arrière (SRA) (aussi appelé backtrack), les techniques de *propagation de contraintes* comme Forward Checking (FC), les techniques de *filtrages* comme Maintaining Arc Consistency (MAC) et les *heuristiques de branchement*. Aussi, il existe de nombreux solveurs pour la PPC (libres & payants) tels que : CP Solver (https ://code.google.com/p/or-tools/), Choco (https ://choco.sourceforge.net/), ILOG CP (https ://www.ilog.com/products/cp/), Gecode (https ://www.gecode.org), Xpress Kalis (www.dashoptimization.com), etc.

Dans la littérature de la RO, parmi les problèmes qu'on résout souvent avec la PPC, on peut citer les problèmes liés à l'ordonnancement, allocation des ressources, emplois du temps, conception de circuits, séquençage de l'ADN, etc.

Le lecteur intéressé par de plus amples informations pourra consulter différents ouvrages dédiés à la PPC dont les livres de F. Fages [42], de K. Marriott et P.J. Stuckey [91] et de K.R. Apt [12].

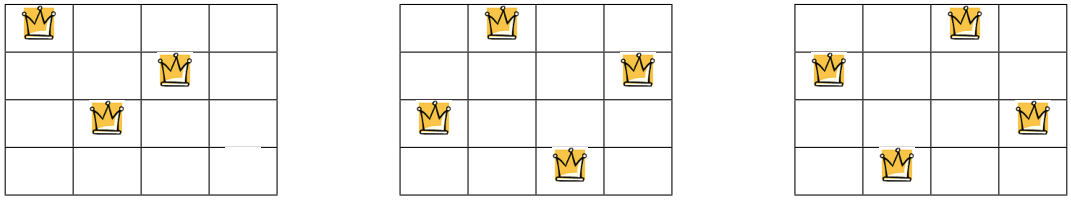
Nous considérons, comme exemple, le problème des N reines avec N = 4.

Exemple 2.3. Le problème des 4 reines

Le but de ce problème, inspiré du jeu d'échec, est de placer 4 reines sur un échiquier de dimension 4×4 *de manière à ce qu'aucune ne soit en prise. Deux reines sont en prises si elles sont sur la même ligne, la même colonne ou la même diagonale. Un échiquier classique comporte 8 lignes et 8 colonnes. Ce problème désormais classique est devenu une référence de mesure de performance des systèmes grâce à un énoncé simple masquant sa difficulté.*

Un modèle classique sans contraintes globales est défini par le triplet (X, D, C)*, où* $X = \{x_1, x_2, x_3, x_4\}$*,* $D_1 = D_2 = D_3 = D_4 = \{1,2,3,4\}$ *et* $C = \{C_1, C_2, C_3\}$*. En observant que deux reines ne peuvent pas être placées sur la même colonne, on peut imposer que la reine* i *soit sur la colonne* i*. Ainsi, la variable* x_i *de domaine* $\{1,2,3,4\}$ *représente la ligne où est placée la reine dans la colonne* i*. Les contraintes* $(C_1 : x_i \neq x_j, 1 \leq i < j \leq n)$ *imposent que les reines soient sur des lignes différentes alors que les contraintes* $(C_2 : x_i \neq x_j + (j-i), 1 \leq i < j \leq n)$ *et* $(C_3 : x_i \neq x_j - (j-i), 1 \leq i < j \leq n)$ *imposent que deux reines soient placées sur des diagonales différentes.*

La Figure 2.4 montre plusieurs affectations pour le problème des 4 reines dans lesquelles le placement d'une reine sur l'échiquier est représenté par une couronne dans la case correspondante. Les reines sont ordonnées de gauche à droite et les positions de haut en bas. On peut observer en Figure 2.4a que l'affectation partielle $x_1 \leftarrow 1, x_2 \leftarrow 3, x_3 \leftarrow 2$ *n'est pas consistante car elle viole une des contraintes* (C_2)*. Au contraire, les affectations totales* $x_1 \leftarrow 2, x_2 \leftarrow 4, x_3 \leftarrow 1, x_4 \leftarrow 3$ *(Figure 2.4b) et* $x_1 \leftarrow 3, x_2 \leftarrow 1, x_3 \leftarrow 4, x_4 \leftarrow 2$ *(Figure 2.4c) sont les deux solutions symétriques des 4 reines. Si l'on définit un problème d'optimisation sous contraintes à partir du problème des* N *reines en définissant la fonction objectif* $\min(x_1)$*, alors le problème admet une unique solution optimale :* $x_1 \leftarrow 2, x_2 \leftarrow 4, x_3 \leftarrow 1, x_4 \leftarrow 3$.



(a) (b) (c)

FIGURE 2.4 – Problème des 4 reines : exemples d'affectation

2.4 La théorie de la complexité

Après avoir caractérisé le problème réel sous la forme d'un *modèle*, la question qui se pose maintenant est : *comment juger la difficulté de ce problème afin de le résoudre de manière efficace ?*

2.4.1 Les classes de complexité

La Théorie de la Complexité (TC), développée au cours des années 1970 à partir des travaux de S. Cook (1970), de R.M. Karp (1972) et de L. Levin (1973), porte sur la classification des problèmes en fonction de leur difficulté de résolution et les frontières existant entre ces différentes classes. En d'autres termes, la TC définit l'étude formelle de l'efficacité des algorithmes ainsi que de la difficulté inhérente aux problèmes. C'est-à-dire, elle mesure le temps et l'espace nécessaires à la résolution d'un problème donné. La TC se limite à l'étude des problèmes de décision.

Qu'est-ce qu'un « Problème de Décision » ?

Un *problème de décision* est un énoncé dont la réponse est « OUI » ou « NON ». La formulation classique d'un tel problème est : " Existe-t-il une solution telle que <conditions> ?". Les <conditions> sont un ensemble de contraintes que doit respecter la solution.

En d'autres termes, le problème de décision pose la question de l'existence d'un algorithme pour trouver une réponse. Pour un problème P, on note par O(P) l'ensemble des instances de P dont la réponse est « OUI ». Pour autant, la TC peut facilement s'étendre aux problèmes d'optimisation. En effet, les *problèmes d'optimisation*[4] généralisent en quelques sorte les *problèmes de décisions*. En d'autres termes, il est possible d'associer à chaque *problème d'optimisation* un *problème de décision* en introduisant un seuil K correspondant à la fonction objectif f. Le *problème de décision* associé devient alors : existe-t-il une solution S telle que $f(S) \leq K$?, pour une fonction à minimiser.

Pour ces problèmes, on définit notamment les deux classes $\mathscr{P}$ et $\mathscr{NP}$:

Qu'est-ce que la « Classe $\mathscr{P}$ » ?

La classe $\mathscr{P}$ définit l'ensemble de tous les problèmes de décision pouvant être résolus par un *algorithme déterministe*[a] en un *temps polynomial*, appelés aussi *algorithmes polynomiaux*.

a. Étant donné un ensemble de données fixé en entrée, un algorithme est dit déterministe si à chaque exécution, il effectue toujours la même suite d'opérations

En effet, un algorithme est dit *polynomial* si son temps d'exécution est borné par $O(p(x))$, où p un polynôme et x la taille de l'entrée (le nombre de bits nécessaires pour stocker une instance d'entrée du problème dans un ordinateur). On dit alors que les problèmes de la classe $\mathscr{P}$ sont « faciles » à résoudre.

Qu'est-ce que la « Classe $\mathscr{NP}$ » ?

C'est l'abréviation pour *Non-Determinist Polynomial Time*. La classe $\mathscr{NP}$ est formée des problèmes de décision D dont une instance est dans OUI(D) si et seulement si elle admet un certificat qui peut être vérifié par un algorithme en temps polynomial (un *algorithme non-déterministe* qui comporte des instructions de choix aléatoires). En particulier le certificat doit être de taille polynomiale. Plus formellement : $D \in \mathscr{NP}$ s'il existe $D' \in \mathscr{P}$, tel que pour toute instance w de D : $w \in$ OUI(D) si et seulement s'il existe w' tq $(w, w') \in$ OUI(D').

Une autre manière de caractériser la classe $\mathscr{NP}$ consiste à dire que ce sont les problèmes de décision P pour lesquels les seuls algorithmes connus de résolution sont ceux ayant une *complexité exponentielle*.

En outre, les algorithmes "ordinaires" sont évidemment des cas particuliers des *algorithmes non-déterministes*. Aussi, tout *problème de décision* qui peut être résolu par un *algorithme polynomial*, donc appartenant à la classe $\mathscr{P}$, appartient également à la classe $\mathscr{NP}$, d'où $\mathscr{P} \subseteq \mathscr{NP}$.

Cependant, certains problèmes de la classe $\mathscr{NP}$ apparaissent plus « difficiles » à résoudre dans le sens où l'on ne trouve pas d'*algorithmes polynomiaux* pour les résoudre avec une machine déterministe. Ces problèmes forment alors la classe des problèmes $\mathscr{NP}$-Complets.

4. Un problème est dit combinatoire lorsqu'il comprend un grand nombre de solutions admissibles satisfaisant une certaines propriétés parmi lesquelles on cherche une solution réalisable optimale ou proche de l'optimum.

Qu'est-ce que la « Classe $\mathcal{NP}$-Complet » ?

La classe $\mathcal{NP}$-Complet ($\mathcal{NPC}$) est l'ensemble de tous les problèmes de décision P tel que :

— $P \in \mathcal{NP}$

— Pour tous problèmes P' dans $\mathcal{NP}$, P' est polynomialement réductible à P : $P' \propto P$, $\forall\, P' \in \mathcal{NP}$.

La notion de *transformation polynomiale* permet de traduire le fait qu'un problème n'est pas plus dur qu'un autre, comme le montre la Figure 2.5. Cette notion représente la troisième étape de la preuve de la $\mathcal{NP}$-Complétude d'un problème donné qui sera détaillée d'une manière formelle dans la section 2.4.2. On peut alors la définir comme suit :

Qu'est-ce qu'une « Transformation Polynomiale » ?

Il existe une *transformation polynomiale* f d'un problème P' en un problème de décision P si étant donnée toute instance I' de P', il est possible de construire une instance I de P en un temps polynomial, et tel que la réponse à I' est « OUI » si et seulement si la réponse à I est « OUI » également.

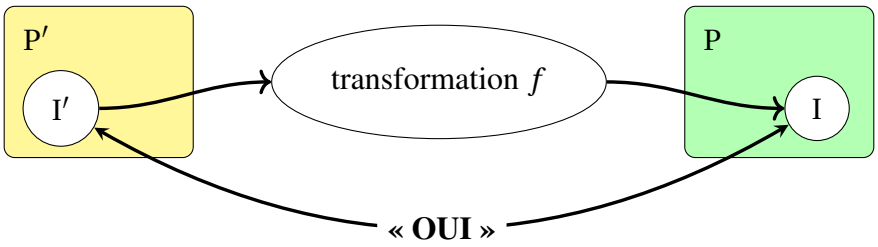


FIGURE 2.5 – Transformation Polynomiale de P' vers P

Remarque 2.3. *La transformation polynomiale* $\propto$ *est transitive, c'est à dire*

$$P_1 \propto P_2 \text{ et } P_2 \propto P_3 \Rightarrow P_1 \propto P_3$$

La Figure 2.6 illustre les relations entre les différentes classes de complexité.

D'autres classes de complexité sont définies. La classe *PSPACE* est l'ensemble des problèmes pouvant être résolus en utilisant une quantité d'espace mémoire raisonnable de taille $\in O(p(n))$ pour chaque instance de taille n (définie comme une expression polynomiale en fonction de la taille des données) sans considération du temps d'exécution que cela prendra, alors que la classe *LogSPACE* définit l'ensemble des problèmes de décision qui peuvent être résolus à l'aide d'un algorithme qui requiert une place mémoire de taille $\in O(log(n))$ pour chaque instance de taille n. En outre, la classe co-$\mathcal{NP}$ est l'ensemble des problèmes de décision P tel que $\bar{P} \in \mathcal{NP}$, où $\bar{P}$ définit son problème complémentaire tel que pour toute instance p on a $p \in O(P) \Leftrightarrow p \neq O(\bar{P})$.

On sait que $LogSPACE \subset PSPACE$. On sait aussi que $LogSPACE \subseteq \mathcal{P} \subseteq \mathcal{NP} \subseteq PSPACE$. En ce qui concerne co-$\mathcal{NP}$, personne ne sait si $\mathcal{NP} \neq$ co-$\mathcal{NP}$. Par contre, il a été démontré que ($\mathcal{NP}$-Complet $\cap$ co-$\mathcal{NP} \neq \varnothing$) $\Leftrightarrow$ ($\mathcal{NP}$ = co-$\mathcal{NP}$).

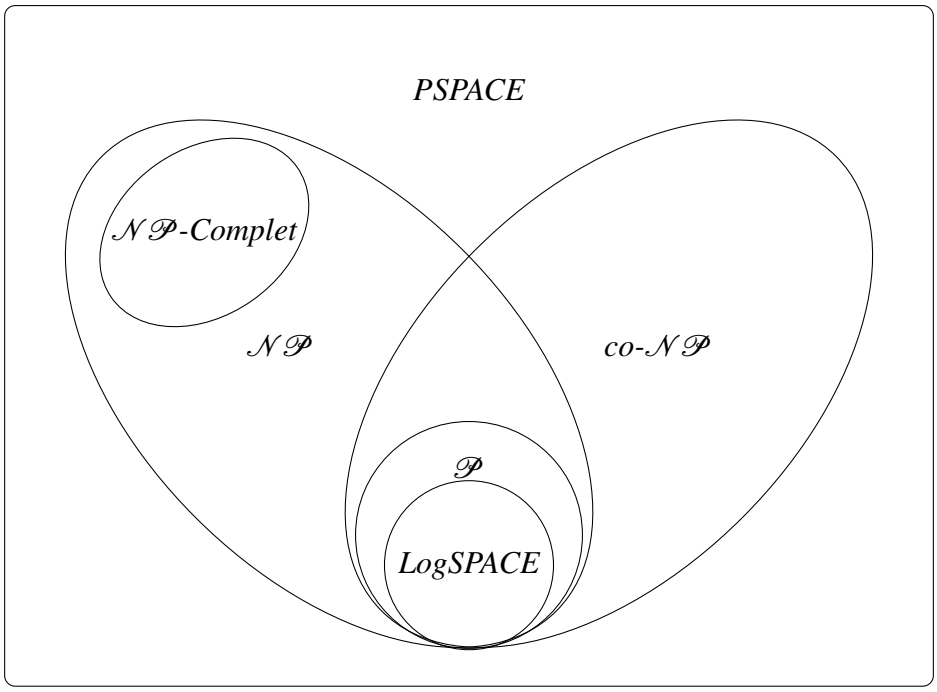


FIGURE 2.6 – Diagramme des différentes classes de complexité et ensembles de langages

2.4.2 Prouver la NP-Complétude d'un problème

La notion de $\mathcal{NP}$-Complétude concerne la reconnaissance des problèmes les plus durs de la classe $\mathcal{NP}$. La notion de la difficulté d'un problème qui est introduite dans cette classe est celle d'une classe de problèmes qui sont équivalents en ce sens que si l'un d'eux est prouvé être « facile » alors tous les problèmes de $\mathcal{NP}$ le sont. Inversement, si l'un d'eux est prouvé être difficile, alors la classe $\mathcal{NP}$ est distincte de la classe $\mathcal{P}$. Le concept central relié à la définition de la $\mathcal{NP}$-Complétude est celui de la *réduction polynomiale* entre problèmes. En d'autres termes, la *réduction polynomiale* peut être vue comme un moyen pour affirmer qu'un problème est aussi « facile » ou « difficile » qu'un autre problème.

Sachant que la *transformation polynomiale* (concept défini à la Section 2.4.1 et illustré par la Figure 2.5) est transitive, pour montrer qu'un nouveau problème P est $\mathcal{NP}$-Complet, on procède comme suit :

1. Montrer que P est dans $\mathcal{NP}$.
2. Choisir un problème P′, $\mathcal{NP}$-Complet approprié.
3. Construire une *réduction polynomiale* f, qui transforme P′ vers P (Figure 2.5), et prouver que f est bien une réduction polynomiale $\mathcal{R}$ (voir Figure 2.7) :
   C'est-à-dire, un problème de décision P′ se réduit à un problème de décision P s'il existe un *algorithme polynomial* qui transforme toute entrée I′ de P′ en une entrée I = $\mathcal{R}$(I′) de P telle que : I′ **une « OUI-instance » de** P′ $\Leftrightarrow$ I **est une « OUI-instance » de** P

Remarque 2.4. *Si* P′ *se réduit à* P (P′ $\propto$ P)*, alors*

— P′ *est plus facile que* P,

— *Si on trouve un algorithme polynomial pour* P*, on en a un aussi pour* P′.

Un des plus importants résultats en *théorie de la complexité* est celui de Stephen Cook, en 1971. Dans son article, il montre le premier problème $\mathcal{NP}$-Complet, soit le problème SAT (une description détaillée du problème sera donnée dans la Section 2.4.5).

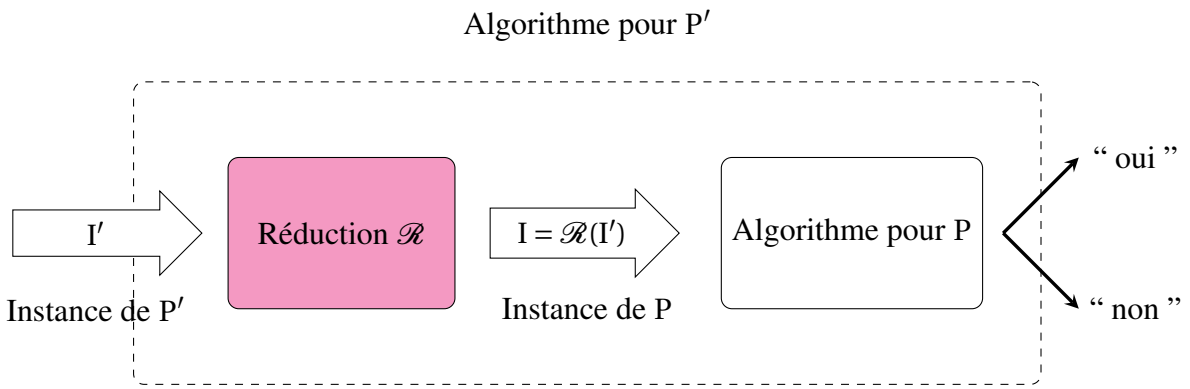


FIGURE 2.7 – Réduction polynomiale $P' \propto P$

2.4.3 Synthèse sur la théorie de complexité

La *théorie de la complexité* permet de classer mathématiquement les problèmes en plusieurs classes de difficulté. En résumé, être dans $\mathscr{P}$, c'est trouver une solution en *temps polynomial*, tandis qu'être dans $\mathscr{NP}$, c'est prouver qu'il est solvable avec un *algorithme polynomial non-déterministe* (*Non-Determinist Polynomial Time*). Par ailleurs, la théorie de $\mathscr{NP}$-Complétude nous permet d'identifier les propriétés qui font qu'un problème soit difficile (le plus dur de la classe $\mathscr{NP}$). Cela devrait nous aider à mieux cerner les différentes directions de résolution.

Les relations d'inclusions entre les classes $\mathscr{P}$ et $\mathscr{NP}$ sont à l'origine d'une très célèbre conjecture que l'on peut définir par « $\mathscr{P} \neq \mathscr{NP}$ ». En effet, si la classe $\mathscr{P}$ est manifestement incluse dans la classe $\mathscr{NP}$, la relation inverse « $\mathscr{P} = \mathscr{NP}$ » n'a jamais été ni montrée ni infirmée. Autrement dit, déterminer si « $\mathscr{P} = \mathscr{NP}$ » revient à trouver un *algorithme polynomial* qui résout au moins un problème dans la classe $\mathscr{NP}$-Complet (une liste restreinte de problèmes $\mathscr{NP}$-Complets sera décrite dans la section 2.4.5). Alors, on pourrait résoudre tous les problèmes de $\mathscr{NP}$ en un *temps polynomial* sur une machine déterministe. Dans ce cas, la question de savoir si « $\mathscr{P} = \mathscr{NP}$ » serait résolue de fait. Mais, on considère généralement que « $\mathscr{P} \neq \mathscr{NP}$ ».

2.4.4 Exemples de problèmes de la classe P

Dans l'étude de la complexité des problèmes, le point important est de savoir si, pour un problème donné, il existe un algorithme de complexité polynomiale pour le résoudre. Dans ce contexte, on peut citer un ensemble restreint de problèmes d'optimisation combinatoire faciles à résoudre tels que :

a) **Le problème du plus court chemin**
Les problèmes de cheminement dans les graphes comptent parmi les problèmes les plus anciens de la *théorie des graphes*. Ce type de problème se rencontre soit directement, soit comme sous-problème dans de nombreuses applications dont les problèmes de tournées, de gestion de stock, d'investissement, les problèmes d'optimisation de réseaux, etc.

De façon générale, l'objectif est de calculer un *chemin* entre des sommets d'un graphe qui minimise ou maximise une certaine fonction. Il existe de nombreuses variantes dérivées du *problème de plus court chemin* (PPCC, ou SPP pour *Shortest Path Problem* en anglais). La plus classique est définie comme suit :

Plus court chemin

DONNÉES : un graphe orienté $\overrightarrow{G} = (V, A, l)$, où chaque arc (u, v) est valué par la longueur $l(u, v)$, et deux sommets s et t ($s, t \in V$).
QUESTION : existe t-il un chemin $C_{s,t}$ de s à t dont la longueur totale $l(C)$ est minimale ?

Il existe plusieurs algorithmes de complexité polynomiale pour résoudre le problème classique du plus court chemin.

Le premier algorithme est celui de *Dijkstra*, qui a été publié par E.W. Dijkstra en 1959 [35]. Cet algorithme calcule des plus courts chemins à partir d'une source dans un graphe orienté $\overrightarrow{G} = (V, A)$ pondéré par des réels positifs. L'algorithme de *Dijkstra* est de complexité à l'ordre de $m + nlogn$, où m est le nombre des arcs et n est le nombre de sommets dans $\overrightarrow{G}$. Il utilise une stratégie gloutonne lorsqu'il choisit le sommet le moins coûteux à chaque étape.

En revanche, lorsque le graphe ne comporte pas de cycle et dont les arcs peuvent avoir des poids négatifs, un algorithme fréquemment utilisé est celui de *Bellman-Ford*, qui a été publié par R. Bellman, S. End et L.R. Ford en 1956 et 1958. La complexité de cet algorithme est en $O(nm)$. Contrairement à l'algorithme de *Dijkstra*, l'algorithme de *Bellman-Ford* autorise la présence de certains arcs de poids négatif et permet de détecter l'existence d'un circuit absorbant, c'est-à-dire de poids total strictement négatif, accessible depuis le sommet source.

b) **Le problème du flot maximum**

Le problème du *flot maximum* (FM, ou MFP pour *Maximum Flow Problem* en anglais) est un problème d'optimisation classique dans le domaine de la RO. Il consiste à trouver, dans un réseau de flot (aussi appelé réseau de transport), un flot réalisable depuis une source unique et vers un puits unique qui soit maximum.

Flot maximum

DONNÉES : un graphe orienté $\overrightarrow{G} = (V, A, c)$, où chaque arc (u, v) a une capacité $c(u, v)$, et deux sommets s et t, ($s, t \in V$).
QUESTION : quel est le flot maximum f qu'il est possible de faire passer dans ce réseau depuis la source s vers le puits t, sous la contrainte de capacité ?

Différents algorithmes, avec des complexités polynomiales différentes, ont été développés pour résoudre ce problème. Parmi ceux-ci, nous citons l'algorithme de *Ford-Fulkerson*, qui a été publié par L.R. Ford et D.R. Fulkerson en 1954. Il s'agit d'un algorithme itératif. À chaque itération, la solution courante est un flot qui satisfait les contraintes de capacité (c'est donc un flot réalisable) et l'algorithme essaie d'augmenter la valeur de ce flot. Cet algorithme est alors de complexité, dans le pire des cas, en $O(m^2 n)$ pour un graphe $\overrightarrow{G}$ avec n sommets et m arcs [46].

Nous citons aussi l'algorithme d'*Edmonds–Karp* qui a été publié par J. Edmonds et R. Karp en 1972. Cet algorithme est une spécialisation de l'algorithme de *Ford-Fulkerson* de telle sorte que le chemin trouvé doit être le chemin le plus court qui possède une capacité positive. Il est de complexité en $O(mn^2)$, où n est le nombre de sommets et m est le nombre des arcs dans le graphe $\overrightarrow{G}$.

Une liste plus complète des différents algorithmes proposés pour la résolution du problème du flot maximum, est disponible dans le livre "Introduction to algorithms, Chapter 26 : Maximum Flow" de Thomas et al. [112].

c) **Le couplage**
En théorie des graphes, un *couplage* ou *appariement* (ou en anglais *matching*) d'un graphe non-orienté G = (V, E) est l'ensemble d'arêtes M ⊆ E qui n'ont pas d'extrémités en commun.

Couplage

DONNÉES : un graphe non-orienté G = (V, E)
QUESTION : existe t-il un couplage M telles que les arêtes sont deux à deux non adjacentes ?

On distingue deux types de couplage : le *couplage maximum* et le *couplage maximal*. Le *couplage maximum* est le couplage couvrant le plus grand nombre de sommets possibles, en laissant donc le moins de sommets isolés. Alors que le *couplage maximal* est un couplage M du graphe tel que toute arête du graphe possède au moins une extrémité commune avec une arête de M. Un couplage, dit *parfait* (ou *couplage complet*), est un couplage couvrant tous les sommets (c'est-à-dire tout sommet est l'extrémité d'une des arêtes du couplage).

Le premier algorithme de complexité polynomiale, proposé pour résoudre le problème du couplage maximum dans un graphe quelconque, est l'algorithme d'*Edmonds pour les couplages* (appelé aussi *Blossom algorithm*). Il a été défini par J. Edmonds en 1965 [40].

Dans le cas d'un *graphe biparti* [5], la recherche d'un couplage parfait de poids minimum dans ce graphe est appelée le *problème d'affectation*. Il existe plusieurs algorithmes pour résoudre ce problème en temps polynomial dont l'algorithme *Hongrois*, parfois appelé aussi *algorithme de Kuhn-Munkres*.

2.4.5 Exemples de problèmes NP-Complets

Le *problème de satisfaisabilité booléenne* (SAT) est le premier problème qui a été montré $\mathcal{NP}$-Complet par Stephen Cook en 1971 [28]. Il a aussi montré que tous les problèmes de la classe $\mathcal{NP}$ sont réductibles au problème SAT.

Le propos ici n'est pas de présenter en détail le problème SAT, mais de donner sa définition formelle. Le problème SAT est un problème de logique propositionnelle, encore appelée logique d'ordre 0 ou logique des prédicats. Rappelons qu'un prédicat est défini sur un ensemble de variables logiques, à l'aide des 3 opérations élémentaires que sont la négation *NON* ($\neg x$ que nous noterons aussi $\bar{x}$), la conjonction *ET* ($x \wedge y$) et la disjonction *OU* ($x \vee y$). Un *littéral* est un prédicat formé d'une seule variable (x) ou de sa négation $\bar{x}$. Une *clause* est un prédicat particulier, formée uniquement de la disjonction de littéraux, par exemple C = $x \vee \bar{y} \vee z$. Une formule est sous forme normale conjonctive si elle s'écrit comme la conjonction de clauses. Le problème SAT consiste à décider si une formule en forme normale conjonctive est satisfiable, c'est-à-dire si il existe une assignation Γ de valeurs de vérité (vrai, faux) aux variables telles que toutes les clauses sont satisfaites (c'est-à-dire elles prennent la valeur vrai).

Théorème 4.1: Théorème de Cook en 1971

Le problème SAT est $\mathcal{NP}$-Complet.

5. Un graphe G = (V, E) est dit *biparti* s'il existe une partition de son ensemble de sommets V en deux sous-ensembles V_1 et V_2 telle que chaque arête de E ait une extrémité dans V_1 et l'autre dans V_2. Un graphe est *biparti* si et seulement s'il ne contient pas de cycle impair.

Nous ne donnons pas ici la preuve complète de ce théorème. En revanche, il est facile de vérifier que SAT $\in \mathcal{NP}$.

Preuve 4.1

— **Instance** : m clauses $C_i, \forall i \in \{1,\dots,m\}$ formées à l'aide de n littéraux.

— **Question** : la formule $\Phi = C_1 \wedge \cdots \wedge C_m$ est-elle satisfiable ?
Par exemple la formule $(x \vee \bar{y} \vee z) \wedge (\bar{x} \vee \bar{y}) \wedge (x \vee \bar{z})$ est satisfaite avec l'assignation $\Gamma(x) = true$ et $\Gamma(y) = false$. Il est facile de se convaincre que le problème SAT est dans $\mathcal{NP}$. Un certificat de positivité consiste à donner une assignation des variables, codable sur un vecteur de n bits (n est le nombre de littéraux). La vérification que toutes les clauses sont satisfaites est alors clairement polynomiale (plus précisément, dans $O(nm)$).
En effet, S. Cook a montré en 1971 que ce problème est $\mathcal{NP}$-Complet en codant l'exécution de tout algorithme sur une machine de Turing non-déterministe par une expression en forme normale conjonctive

□

Comme l'indique la Figure 2.8 ci-dessous, Richard M. Karp en 1972 [75] a prouvé, à l'aide de transformations polynomiales successives, que les différents problèmes illustrés sont $\mathcal{NP}$-Complets.

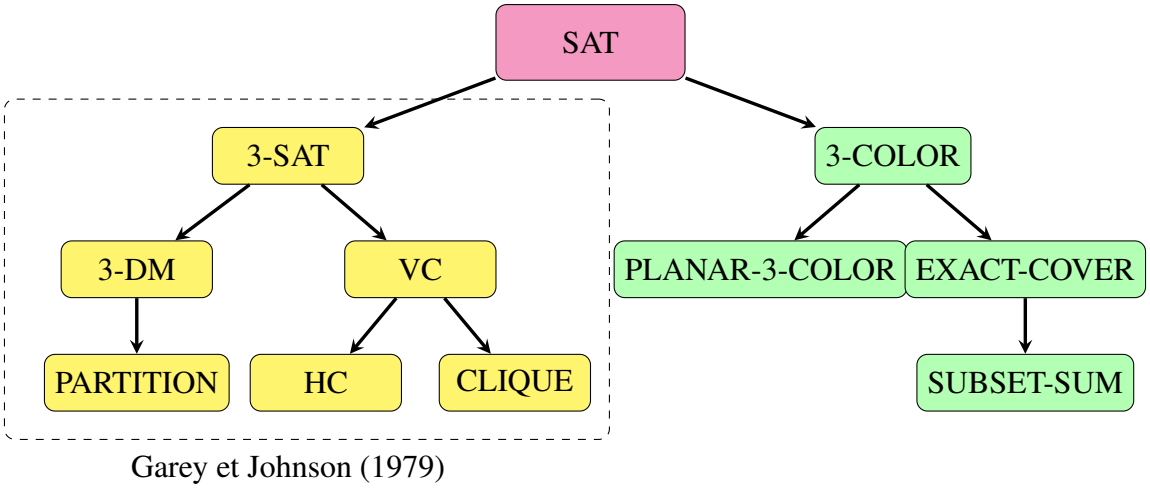


FIGURE 2.8 – Illustration des transformations polynomiales successives de R.M. Karp (1972) [75]

La $\mathcal{NP}$-Complétude des six problèmes de base suivants (3-SAT, 3-DIMENSIONAL MATCHING (3DM), VERTEX COVER (VC), HAMILTONIAN CIRCUIT (HC), CLIQUE et PARTITION) a été exposée par M.R. Garey and D.S. Johnson en 1979 [48]. Nous présentons ici ces problèmes en termes de données d'entrée du problème et la question sur l'existence d'une solution réalisable.

3-SAT

DONNÉES : un ensemble de clauses de dimension 3 ;
QUESTION : existe-il une affectation de ces variables qui satisfait toutes les clauses ?

★ 3-DIMENSIONAL MATCHING (3DM)

DONNÉES : un ensemble M de triplets $M \subseteq W \times X \times Y$, où W, X et Y sont disjoints et de même cardinalité q ;
QUESTION : existe t-il $M' \subseteq M$, tel que $|M'| = q$ et les triplets de M' sont deux à deux disjoints ?

★ VERTEX COVER (VC)

DONNÉES : un graphe $G = (V, E)$ non-orienté et un entier positif $k \leq |V|$;
QUESTION : existe t-il $V' \subseteq V$, avec $|V'| \leq k$, tel que toute arête de G ait au moins une extrémité dans V' ?

★ PARTITION

DONNÉES : un ensemble fini de n entiers positifs $\{a_1, a_2, \ldots, a_n\}$;
QUESTION : existe t-il un sous-ensemble $A' \subseteq A = \{1, 2, \ldots, n\}$ tel que $\sum_{i \in A'} a_i = \sum_{i \in A' \setminus A} a_i$?

★ HAMILTONIAN CIRCUIT (HC)

DONNÉES : un graphe $G = (V, E)$ non-orienté ;
QUESTION : existe t-il un cycle passant une et une seule fois par tous les sommets de V ?

★ CLIQUE

DONNÉES : un graphe $G = (V, E)$ non-orienté et un entier positif $1 \leq k \leq |V|$;
QUESTION : existe t-il $V' \subseteq V$, avec $|V'| = k$, tel que $u, v \in V' \Rightarrow (u, v) \in E$?

Remarque 2.5. *Une description détaillée des problèmes (* 3-COLOR, PLANAR-3-COLOR, EXACT-COVER *et* SUBSET-SUM*) est donnée par Richard M. Karp (1972) dans son article intitulé : "Reducibility among combinatorial problems" [75]*

Remarque 2.6. *Une liste de centaines de problèmes $\mathcal{NP}$-Complets est produite dans le livre de M.R. Garey and D.S. Johnson (1979) : "Computers and intractability : a guide to the theory of $\mathcal{NP}$-Completeness" [48]. En outre, une liste plus actualisée est disponible à l'adresse : https ://www.csc.kth.se/ viggo/problemlist/.*

2.5 Aperçu des approches de résolution

Nombreuses sont les méthodes de résolution de problèmes d'optimisation combinatoire proposées dans la littérature de la RO. Elles se décomposent en deux grandes catégories : les *méthodes exactes* et les *méthodes approchées*. Un *algorithme exact* permet de garantir l'optimalité de la solution obtenue, à la différence d'un *algorithme approché* qui tente de s'en approcher le plus possible. Nous allons détailler quelques exemples d'algorithmes de résolution de chaque catégorie afin de donner brièvement une première vision sur leurs applications.

2.5.1 Résolution exacte

Une *méthode exacte* permet d'obtenir la solution optimale à chaque fois, mais le temps de calcul peut être long si le problème est compliqué à résoudre.

a) **La méthode par séparation et évaluation (PSE)**
Plus connue sous son appellation anglaise de Branch and Bound (B&B), est une méthode algorithmique classique pour résoudre un problème d'optimisation combinatoire. Elle repose sur une méthode arborescente de recherche d'une solution optimale par séparations et évaluations, en représentant les états solutions par un arbre d'états, avec des nœuds et des feuilles. Le B&B est basé sur trois axes principaux : la *séparation* (*Branch*), l'*évaluation* (*Bound*) et la *stratégie de parcours*. La *séparation* consiste à diviser le problème en sous-problèmes. Ainsi, en résolvant tous les sous-problèmes et en gardant la meilleure solution trouvée, on est assuré d'avoir résolu le problème initial. Cela revient à construire un arbre permettant d'énumérer toutes les solutions. L'exploration de ces solutions utilise ensuite une *évaluation* optimiste pour majorer les sous-ensembles, ce qui permet de ne plus considérer que ceux susceptibles de contenir une solution potentiellement meilleure que la solution courante. En ce qui concerne l'axe *stratégie de parcours*, il existe trois parcours possible de l'arbre : *la profondeur d'abord*, *le meilleur d'abord* ou *la largeur d'abord*.

En théorie, la complexité de l'algorithme B&B est exponentielle. En pratique, et en moyenne elle devient meilleure, souvent en $O(1.5^n)$ (faiblement exponentielle). Plusieurs références à ce sujet peuvent être consultées. Parmi celles-ci, nous citons l'article de J. Clausen [25] et le live de G. Desaulniers et al. [34].

b) **Génération de colonnes**
C'est une technique utilisée pour résoudre efficacement les programmes linéaires de grande taille. Elle repose sur la décomposition du modèle original par la méthode de G.B. Dantzig et P. Wolfe [31]. L'idée principale de cette méthode consiste à décomposer le modèle d'origine en un ou plusieurs sous-problèmes plus facile à résoudre. En termes simples, l'algorithme de *génération de colonne*s consiste à résoudre le problème original avec un sous-ensemble de colonnes (variables) de taille réduite, appelé un *problème maitre réduit* (PMR). Ce procédé est répété jusqu'à ce qu'aucune colonne ne puisse être ajoutée pour améliorer la solution courante, donnant ainsi la solution optimale du problème. L'efficacité des algorithmes basés sur la *génération de colonnes* est directement liée au nombre de colonnes générées. La méthode de *génération de colonnes* pourra être utilisée comme évaluation par défaut dans un B&B qui cherche la solution optimale entière du problème.

c) **Relaxation lagrangienne**
L'*approche lagrangienne* en optimisation combinatoire est un algorithme basé sur la *relaxation linéaire*. Elle permet d'obtenir des bornes inférieures de bonnes qualité pour certains problèmes d'optimisation combinatoire difficile. L'idée de cette technique consiste à supprimer (relaxer) les contraintes difficiles qui rendent le problème compliqué en les introduisant dans la fonction objectif sous la forme d'une pénalité (combinaison linéaire des contraintes relaxées) si ces contraintes ne sont pas respectées. Les coefficients de cette combinaison linéaire sont appelées les variables duales associées à la relaxation lagrangienne (ou *multiplicateurs de Lagrange*). La nouvelle fonction objectif est appelée le *Lagrangien* du problème. De manière générale, le nouveau problème se résout à l'aide de la méthode de *génération par colonnes* ou la plus utilisée, la descente de *sous-gradient* et ses variations.

d) **La programmation linéaire : PL et PLNE**

Nous avons mentionné dans la section 2.3 que la *programmation linéaire*, avec ses différentes extensions (PL, PLNE et PLM), est un outil de modélisation. Dans cette partie, nous la présentons en tant qu'une méthode exacte de résolution. La *programmation linéaire* (PL) est fréquemment utilisée pour résoudre d'une manière efficace des problèmes d'optimisation liés à des domaines d'application variés.

En effet, la plupart des problèmes de la PL ont été résolu par l'*algorithme du simplexe* développé en 1947 par G.B. Dantzig. Ce dernier étant très efficace la plupart du temps. D'autres algorithmes polynomiaux de résolution de *PL* ont été développés par la suite. Parmi ceux-ci, nous retrouvons la *méthode de l'ellipsoïde* proposée en 1979 par L.G Khachiian [77]. C'était le premier algorithme qui garantissait la résolution d'un *programme linéaire* en temps polynomial mais il reste inefficace en pratique pour des problèmes réels et jugé beaucoup plus lent que l'*algorithme du simplexe*. Une seconde méthode est celle des *points intérieurs* développée en 1984 par N. Karmarkar [74]. Son principe est d'utiliser des fonctions barrières pour décrire l'ensemble des solutions qui est convexe par définition du problème. À l'opposé de l'*algorithme du simplexe*, cette méthode atteint l'optimum du problème en passant par l'intérieur de l'ensemble des solutions réalisables.

Dans le cas de la PLNE, où l'espace de recherche n'est plus convexe mais discret, il existe de nombreuses approches de recherche de la solution optimale entière. Nous citons principalement la méthode de recherche arborescente par *séparation et évaluation* (B&B), la *méthode de coupes* (ou en anglais *Cutting planes method* dont algorithme de Gomory en 1958) et la méthode qui combine les deux dernières (*Branch and Cut*). La méthode de B&B a été introduite dans la Section 2.5.1 :a). La *méthode de coupes* consiste à trouver le plus petit "polyèdre" contenant toutes les solutions entières de la PLNE. En pratique, son principe consiste à relâcher les contraintes d'intégrité pour résoudre la *relaxation linéaire* du problème et ajouter des contraintes supplémentaires, appelées *coupes* ou *troncatures*, pour réduire l'enveloppe des solutions sans éliminer des solutions entières admissibles. La difficulté principale de cette méthode réside dans le choix de coupes efficaces pour la résolution des problèmes pratiques de taille importante.

Il existe une variété de langages de modélisation de la PLNE dont GNU Mathematical Programming, AMPL (A mathematical Programming Language) et IBM-ILOG OPL (Optimization Programming Language), etc, ainsi que de nombreux solveurs (libres et payants) pour sa résolution tels que Excel, COIN-OR (www.coin-or.org), GLPK (www.gnu.org/software/glpk), ILOG Cplex (www.ilog.com), Xpress-Optimizer (www.dashoptimization.com), LINDO (www.lindo.com) et Gurobi (www.gurobi.com), etc. Ils sont devenus remarquablement efficaces grâce à l'intégration récente de techniques de génération automatique de coupes, de préprocessing, d'heuristiques et des stratégies d'énumération bien pensées.

e) **La programmation dynamique**

La *programmation dynamique* (PD, ou DP pour *Dynamic Programming* en anglais), introduite par R. Bellman dans les années 50, est une méthode permettant de résoudre un problème d'optimisation en le décomposant en étapes par le principe d'optimalité. La *programmation dynamique* s'appuie sur un principe simple, appelé le *principe d'optimalité de Bellman* [38]. Ce principe est basé sur l'existence d'une équation récursive permettant de décrire la valeur optimale du critère à une étape en fonction de sa valeur à l'étape précédente. Il s'agit alors d'une *méthode récursive* dans laquelle la solution optimale d'un sous-problème sert à résoudre le sous-problème suivant. Les différentes étapes consistent en la résolution de problèmes de plus en plus gros, jusqu'à obtenir la solution du problème de départ. Cette méthode a une complexité polynomiale ou exponentielle, selon le

problème.

2.5.2 Résolution approchée

Une méthode approchée, appelée aussi *heuristique*, permet de solutionner un problème difficilement solvable par des méthodes exactes (les problèmes *$\mathcal{NP}$-Complets*) sans avoir aucune garantie sur l'optimalité de la solution recherchée. Lorsqu'une heuristique est adaptable sur des types de problème variés, sans changement majeurs de l'algorithme, le terme de *métaheuristiques* est employé. La performance de ces méthodes est généralement calculée par le rapport entre la valeur de la solution calculée par l'*heuristique/métaheuristique* et la valeur de la *solution optimale*. L'avantage d'une *méthode approchée* réside dans leur application à n'importe quelle classe de problèmes faciles ou difficiles. De plus, elles ont démontré leurs robustesses et efficacités face à plusieurs problèmes d'optimisation combinatoires.

Les grandes familles des méthodes heuristiques et métaheuristiques sont les suivantes :

a) **Les algorithmes gloutons**
Les *algorithmes gloutons* (AG, ou GA pour *Greedy Algorithm* en anglais) sont des méthodes constructives permettant de construire progressivement une solution locale. Ils sont couramment utilisés pour fournir rapidement des solutions initiales. Dans certains cas, cette approche permet d'arriver à un optimum global, l'*algorithme glouton* constitue alors une *méthode exacte*, mais dans le cas contraire, on parle d'une *heuristique gloutonne*.

b) **Les méthodes de recherche locale**
Connues aussi sous l'appellation de *métaheuristiques à base de voisinage*. Ces méthodes partent d'une solution de départ unique, puis passent d'une solution à une autre dans l'espace des solutions candidates (l'espace de recherche) jusqu'à ce qu'une solution soit trouvée ou que le temps imparti soit dépassé. La méthode de *recherche locale* la plus élémentaire est la *méthode de descente* pour laquelle une solution voisine doit obligatoirement améliorer le critère, c'est-à-dire, qu'elle s'arrête dès qu'il n'existe plus de meilleure solution dans le voisinage. Autrement dit, l'inconvénient majeur de cette méthode est son arrêt au premier minimum local rencontré. Pour améliorer les résultats, on peut lancer plusieurs fois l'algorithme en partant d'un jeu de solutions initiales différentes, mais la performance de cette technique décroît rapidement.

Une autre méthode est celle du *recuit simulé* (RS, ou SA pour *Simulated Annealing* en anglais). Cet algorithme est dédié à la recherche d'une configuration d'énergie minimale d'un champ de Gibbs. L'idée d'intégrer un paramètre de température et de simuler un recuit a été initialement introduite en 1983 par S. Kirkpatrick et al. [79]. Le principe du *recuit simulé* est de parcourir de manière itérative l'espace des solutions. Contrairement aux *méthodes de descente*, le *recuit simulé* évite le piège des optima locaux.

On peut citer aussi la méthode de *recherche tabou* (RT, ou TS pour *Tabu Search* en anglais), une métaheuristique itérative qualifiée de *recherche locale* au sens large. Elle est introduite en 1986 par F.W. Glover [50], [51]. A l'inverse du *recuit simulé* qui génère de manière aléatoire une solution voisine à chaque itération, la *recherche tabou* examine un échantillonnage de solutions et retient la meilleure solution. Elle ne s'arrête donc pas au premier optimum trouvé.

c) **Les métaheuristiques à base de population**
On les appelle aussi les *algorithmes évolutionnistes* qui travaillent sur une population de solutions plutôt que sur une solution unique. Parmi ces méthodes, nous trouvons les *algorithmes génétiques* (AG, ou GA pour *Genetic Algorithm* en anglais) qui sont des

algorithmes d'optimisation stochastique fondés sur les mécanismes de la sélection naturelle et de la génétique. Ils ont été adaptés à l'optimisation par J. Holland dans les années 1960 [70], également les travaux de D.E Goldberg dans les années 1989 [52], [53] ont largement contribué à les enrichir.

Une autre méthode à base de population est l'algorithme de *colonies de fourmis*, connu sous son appellation anglaise de *Ant Colony Optimisation* (ACO). Cette famille d'algorithmes a été introduite initialement par M. Dorigo at al. dans les années 1990 [27]. Ces algorithmes s'inspirent des comportements collectifs de dépôt et de suivi de piste observés dans les colonies de fourmis. Une colonie d'agents simples (les *fourmis*) communiquent indirectement via des modifications dynamiques de leur environnement (les *pistes de phéromones*) et construisent ainsi une solution à un problème en s'appuyant sur leur expérience collective. Le premier algorithme de ce type (*Ant System*) a été conçu pour la résolution du *problème du voyageur de commerce*, puis amélioré et appliqué à d'autres problèmes d'optimisation combinatoire liés à des domaines d'application variés.

Dans la littérature de la RO, il existe toutefois d'autres méthodes de même type mais moins référencées. Plusieurs articles se référant aux différentes méthodes (exactes, heuristiques ou métaheuristiques) sus-citées et de nombreux livres de référence consacrés entièrement à ces sujets peuvent être consultés. Nous citons par exemple les livres de E.L Johnson et al. [72], de J. Dreo et al. [37] et celui de E.G Talbi [109].

2.6 Brève introduction à l'Optimisation Multi-Objectif

2.6.1 Définition de base

L'Optimisation Multi-Objectif (OMO), ou MOO pour *Multi-Objective Optimization* en anglais, trouve ses racines dans les travaux de F.Y. Edgeworth (1881) [39] et de V. Pareto (1896) [96]. Cette théorie constitue une branche d'étude majeure de la RO qui permet de modéliser des problèmes réels faisant intervenir de nombreux critères (souvent conflictuels) et contraintes. Dans ce contexte, la solution optimale recherchée n'est plus un simple point, mais un ensemble de bons compromis satisfaisant toutes les contraintes.

Les problème d'optimisation multi-objectif (OMO) (ou *problèmes d'optimisation multi-critères*) ont un intérêt grandissant dans l'industrie (économie) et dans les sciences du management, puis graduellement dans les sciences pour l'ingénieur pendant ces dernières années.

Qu'est-ce qu'un « Problème d'Optimisation Multi-Objectif » ?

Mathématiquement, un problème d'optimisation multi-objectif (PMO) s'écrit de la manière suivante :

$$\begin{cases} minimiser & \vec{f}(\vec{x}) & & & \text{(fonction à optimiser)} \\ avec & \vec{g}(\vec{x}) & \leq & 0 & \text{(contraintes d'inégalité)} \\ et & \vec{h}(\vec{x}) & = & 0 & \text{(contraintes d'égalité)} \end{cases}$$

où $\vec{x} \in \mathbb{R}^n$, $\vec{f}(\vec{x}) \in \mathbb{R}^k$, $\vec{g}(\vec{x}) \in \mathbb{R}^m$ et $\vec{h}(\vec{x}) \in \mathbb{R}^p$

Comme on peut le voir dans cette formulation, on n'a plus un seul objectif à optimiser, mais k (le vecteur $\vec{f}$ regroupe k fonctions objectif). Ainsi, les vecteurs $\vec{g}(\vec{x})$ et $\vec{h}(\vec{x})$ représentent respectivement m contraintes d'inégalité et p contraintes d'égalité. Cet ensemble de contraintes délimite un espace restreint de recherche de la solution optimale.

2.6.2 Concepts de base et terminologie

Un PMO consiste à optimiser (à minimiser et/ou à maximiser) "au mieux" les différents objectifs qui sont souvent des objectifs contradictoires. En d'autres termes, deux objectifs sont contradictoires lorsque la diminution d'un objectif entraîne une augmentation de l'autre objectif. Contrairement à l'optimisation mono-objectif, la solution d'un problème multi-objectif n'est pas unique, mais est une multitude de solutions. Ces solutions ne sont pas optimales puisqu'elles minimisent un certain nombre d'objectifs tout en dégradant les performances sur d'autres objectifs.

De manière générale, l'OMO peut être simplement définie comme le domaine qui recherche un équilibre tel que l'on ne peut pas améliorer un critère sans détériorer au moins un des autres critères. Cet équilibre est appelé le Pareto Optimal (PO) (nommé d'après l'économiste italien Vilfredo Pareto).

« Relation de Dominance »

Une solution x domine ($\prec$) au sens de Pareto une solution z si et seulement si $\forall i \in \{1,\ldots,n\}$, $f_i(x) \leq f_i(z)$ et $\exists i \in \{1,\ldots,n\}$ tel que $f_i(x) < f_i(z)$.

« Pareto Optimal »

Une solution x est dite *Pareto optimale* (ou efficace) si elle n'est dominée par aucune autre solution appartenant au domaine des solutions.

Les solutions qui dominent les autres mais ne se dominent pas entre elles sont appelées *solutions optimales au sens de Pareto* (ou aussi *solutions non dominées*).

Le *front de Pareto*, appelé aussi la *surface de compromis* ou encore la *frontière de Pareto*, définit l'ensemble des *points Pareto optimaux*. Graphiquement, la forme la plus courante d'un *front de Pareto*, dans le cas de deux objectifs à minimiser simultanément (f_1 et f_2), est donnée par la Figure 2.9. Dans cette figure, la ligne rouge représente le *front de Pareto* et les deux points A et B représentent les solutions non dominées par aucune autre solution. En revanche, le point C ne se situe pas sur la *frontière de Pareto* parce qu'il est dominé par les points A et B.

À une *surface de compromis*, on observe deux points caractéristiques associés : le *point idéal* et le *point nadir* :

« Point Idéal »

Les coordonnées de ce point sont obtenues en optimisant chaque fonction objectif séparément.

« Point Nadir »

Les coordonnées de ce point correspondent aux pires valeurs obtenues par chaque fonction objectif lorsque l'on restreint l'espace des solutions à la surface de compromis.

Une visualisation de l'ensemble de ces définitions est donnée sur la Figure 2.9.

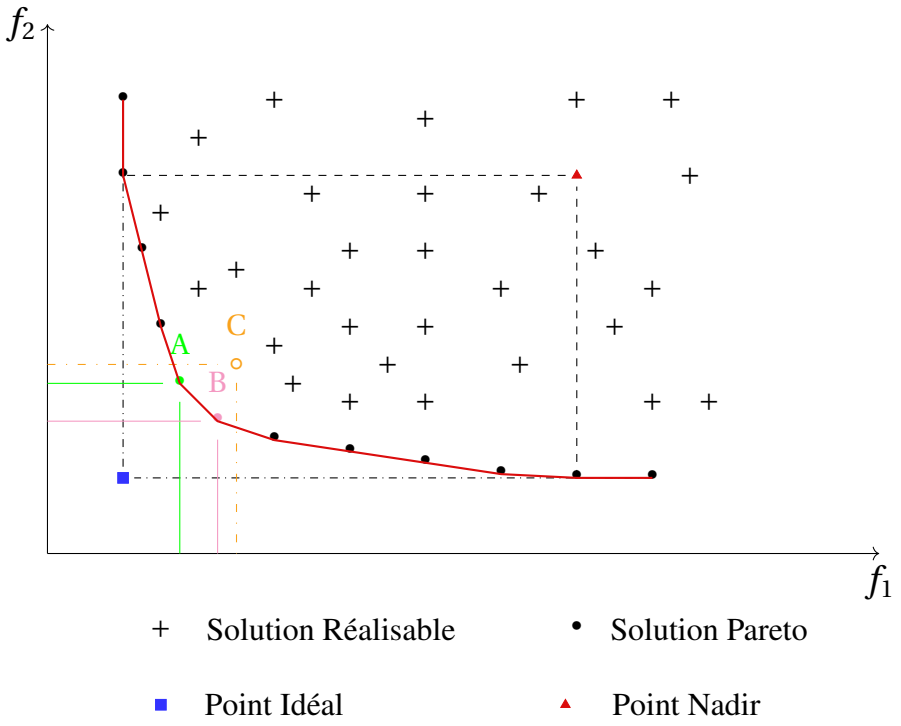


FIGURE 2.9 – Représentation du front de Pareto dans le cas de deux objectifs à minimiser

2.6.3 Approches de résolution

Il existe un nombre important de stratégies de résolution pour prendre en compte la présence de plusieurs objectifs qui peuvent être classées comme suit :

— La méthode de *pondération des fonctions objectif* : son principe est basé sur la transformation du problème en un problème mono-objectif (simple objectif) : on distingue les *approches scalaires*, telles que la méthode ϵ-contrainte, la somme pondérée et la méthode de H.P. Benson [18] et les *approches non-scalaires*. Pour en savoir plus sur ces différentes méthodes, le lecteur peut se référer à [66] et [118].

— Les approches *Pareto* : leur principe est l'optimisation simultanée d'objectifs contradictoires en utilisant la notion de dominance dans la sélection des solutions générées. Nous pouvons citer comme référence le livre de K. Deb [32] et les travaux de D. Goldberg [54].

— les approches *non-Pareto* : leur principe est de traiter séparément les différents objectifs. Pour avoir plus de détails, le lecteur peut se référer à [45], [80] et [120].

Ces différentes méthodes peuvent aussi être réparties en trois familles de méthodes d'OMO [116] :

— Les méthodes à préférence *a priori*, où le décideur ici définit ses préférences entre les différents objectifs avant d'utiliser la méthode d'optimisation.

— Les méthodes à préférence *progressive*, où le décideur affine son choix de compromis au fur et à mesure du déroulement de la méthode d'optimisation.

— Les méthodes à préférence *a posteriori*, où le décideur choisit une solution de compromis en examinant l'ensemble des solutions fournies par la méthode d'optimisation.

2.6.4 Utilité de l'optimisation multi-objectif

Globalement, les problèmes d'optimisation issus de problématiques réelles sont la plupart du temps de nature *multi-objectif* car plusieurs critères sont à considérer simultanément. Optimiser un tel problème relève donc de l'OMO. En effet, il n'est pas toujours possible de trouver un ordre d'importance sur les critères. Il est alors nécessaire de rechercher les solutions de meilleur compromis entre les objectifs.

Dans ce travail de thèse, nous serons confrontés à un *problème d'optimisation combinatoire multi-objectif* qui consiste à minimiser simultanément les trois fameux enjeux électroniques (coût, performance et autonomie) dans le cadre de l'optimisation du fonctionnement d'un générateur de la hiérarchie mémoires dénommé MMOpt (cet outil a été proposé par Mancini et al. (2012) [89] et a été décrit en détail dans le Chapitre 1). Dans ce contexte, il est impossible de parler d'une solution optimale. On cherche alors à identifier un ensemble de solutions de compromis entre ces trois objectifs.

L'utilité d'appliquer quelques outils de l'OMO réside alors dans la diversité des solutions obtenues afin de laisser au concepteur de circuits le choix de sa propre solution de compromis. En revanche, l'application de ces méthodes reste l'une des perspectives les plus importantes de cette thèse.

Pour une présentation plus complète sur la théorie de l'OMO, le lecteur est invité à consulter des documents de référence comme les livres de K.M. Miettinen [94], de M. Ehrgott [41] et de Y. Collette et al. [26], ainsi que des "surveys" tels que celui de E.L. Ulungu et de J. Teghem [115], etc.

2.7 Conclusion

Ce chapitre présente une brève introduction à la RO, une boîte à outils de méthodes de modélisation ainsi qu'un panorama de méthodes classiques de résolution et une synthèse introductive sur les principales définitions et propriétés relatives à l'OMO.

Un même problème réel peut être modélisé de plusieurs façons différentes et donc être résolu par plusieurs méthodes. Dans ce cas de figure, il conviendra de trouver la modélisation en rapport avec les attendus du problème et conduisant à la résolution la plus efficace et la plus adaptée. La qualité des résultats obtenus et des prédictions dépend de la pertinence du modèle proposé, de l'efficacité de l'algorithme choisi et des moyens utilisés pour le traitement numérique.

Pour en savoir davantage sur le domaine de la RO, plusieurs références bibliographiques peuvent être utiles pour le lecteur souhaitant approfondir certaines notions tels que le livre de Alj et al. [7], ainsi que les livres de M. Sakarovitch [103, 104, 105], le livre de C.J. Fournier [46] pour la théorie de graphes, le livre de M.L. Pinedo [97] pour les problèmes d'ordonnancement, les livres de C.H. Papadimitriou [95], de O. Goldreich [55] et de M.R. Garey and D.S. Johnson [48] pour la théorie de complexité et l'ouvrage anglophone d'introduction à la RO de F.S. Hiller et G.J. Lieberman [69], etc.

Le lecteur pourra aussi se référer à différents sites web tels que le site de la société savante ROADEF (*Recherche Opérationnelle et Aide à la Décision Française* : https ://www.roadef.org), le site de l'*Association Européenne des Sociétés de Recherche Opérationnelle* (EURO : https ://www.euro-online.org), et le site de INFORMS (*Institute for Operations Research and the Management Sciences* : https ://www.informs.org) qui est la plus grande société professionnelle dans le domaine de la RO.

Deuxième partie

Optimisation du Fonctionnement des TPUs

« Si j'avais une heure pour résoudre un problème, je passerais 55 minutes à réfléchir au problème et 5 minutes à réfléchir à des solutions. La formulation du problème est souvent plus essentielle que sa solution, qui peut être simplement une question de compétence mathématique ou expérimentale. »

Albert Einstein, (1879-1955)

Chapitre 3

Description et Modélisation du Problème d'Optimisation 3-PSDPP

Sommaire

3.1 Introduction

Comme nous l'avons vu dans le Chapitre 1, l'optimisation du fonctionnement de TPU généré par l'atelier MMOpt engendre une problématique d'optimisation riche à laquelle nous appliquons les méthodes et les techniques issues de l'optimisation combinatoire et de la RO. Ce troisième chapitre décrit la phase de modélisation de cette problèmatique. Cette phase est l'étape la plus cruciale dans le processus de la RO. Au cours de ce chapitre, nous allons d'abord décrire le problème d'optimisation abordé dans cette thèse et citer les différentes suppositions nécessaires imposées par le concepteur du circuit. Ensuite, nous présentons en détail une modélisation spécifique, sous la forme d'un *modèle mathématique* avec des données d'entrées et de sorties clairement définies, pour ce problème. Puis, nous dressons la liste des différents sous-problèmes, mono- et bi-objectif, dérivés de notre problème d'origine. Enfin, nous proposons diverses formulations en termes de PLNE pour certaines variantes mono-objectif.

Ce travail de modélisation a été complexe dans le sens où beaucoup d'échanges ont été nécessaires avec le concepteur de l'outil MMOpt afin de comprendre le problème correctement et de fixer les objectifs à atteindre ainsi que les contraintes à respecter.

3.2 Le problème d'optimisation 3-PSDPP

Comme première étape, nous donnons une description informelle du problème ainsi que la liste des différentes suppositions à prendre en compte afin de spécifier la formulation mathématique correspondante traduisant ce problème.

3.2.1 Description informelle du problème 3-PSDPP

Nous considérons dans cette étude le problème d'optimisation multi-objectif, dénommé 3-objective Process Scheduling and Data Prefetching Problem (3-PSDPP), qui consiste à optimiser simultanément les trois enjeux électroniques classiques : coût, consommation d'énergie et performance.

Étant donné une image d'entrée découpée en un ensemble de tuiles d'entrée (Te) de taille donnée, une image de sortie formée d'un ensemble de tuiles de sortie (Ts) à calculer et une durée nécessaire pour une étape du préchargement ainsi que pour une étape de calcul, le problème 3-PSDPP consiste à déterminer un séquencement des préchargements de tuiles d'entrée et leurs emplacements dans les mémoires locales qui sert à indiquer dans quel buffer (Br) se trouve chaque tuile, et un séquencement des calculs des tuiles de sortie. Le calcul d'une tuile de l'image de sortie nécessite en général plusieurs tuiles de l'image d'entrée. Ces tuiles doivent être préchargées depuis la mémoire externe vers les buffers internes du système des dénommés Input Tile Buffers (ITB). Il est aussi impératif que les tuiles requises soient accessibles dans les buffers au moment du calcul correspondant. En outre, une tuile d'entrée peut être préchargée une ou plusieurs fois à des instants distincts (tuile requise par une ou plusieurs tuiles de sortie). Notons que la contrainte liée aux tuiles partagées entre une ou plusieurs tuiles de sortie, dépend de tuiles utilisées et peut changer d'un calcul à un autre.

Les trois objectifs considérés sont alors la *quantité de buffers* qui permet de minimiser la surface du circuit produit par l'atelier MMOpt (encombrement et coût de production), le *nombre total de préchargements* qui représente en grande partie la consommation d'énergie du circuit produit dû au transfert de données entre la mémoire externe et le TPU et le *temps total de traitement* qui reflète la performance du TPU (contrainte temps-réel).

En utilisant certains concepts liés aux problèmes d'ordonnancement, nous définissons le problème 3-PSDPP comme étant un problème d'ordonnancement sur deux machines en paral-

lèle. En effet, comme le montre l'exemple de Figure 3.1 ci-dessous, les préchargements s'effectuent sur la machine M_1 d'une manière parallèle avec les calculs qui s'exécutent sur la machine M_2, où chaque étape de préchargement est donnée par une tuile d'entrée $Te_i, i \in \{1,\ldots,5\}$ qui doit être préchargée à un instant précis dans le buffer $Br_k, k \in \{1,\ldots,4\}$ ainsi que chaque étape de calcul est donnée par une tuile de sortie $Ts_j, j \in \{1,\ldots,3\}$ qui doit être calculée à un tel moment.

Le calcul d'une tuile de sortie n'est possible que si elle dispose des ressources nécessaires (tuiles d'entrée déjà préchargées dans leurs buffers) et si les calculs qui le précèdent sont terminés (une tuile de sortie à la fois). Lorsqu'une tuile de sortie a terminé son calcul, une autre étape du calcul peut être immédiatement commencée si toutes ses tuiles nécessaires sont déjà préchargées (une tuile d'entrée à la fois). En outre, un calcul ne peut être interrompu une fois qu'il est commencé pour en débuter un autre. Il s'agit donc de ne pas autoriser la préemption des calculs (contrainte de non-préemption).

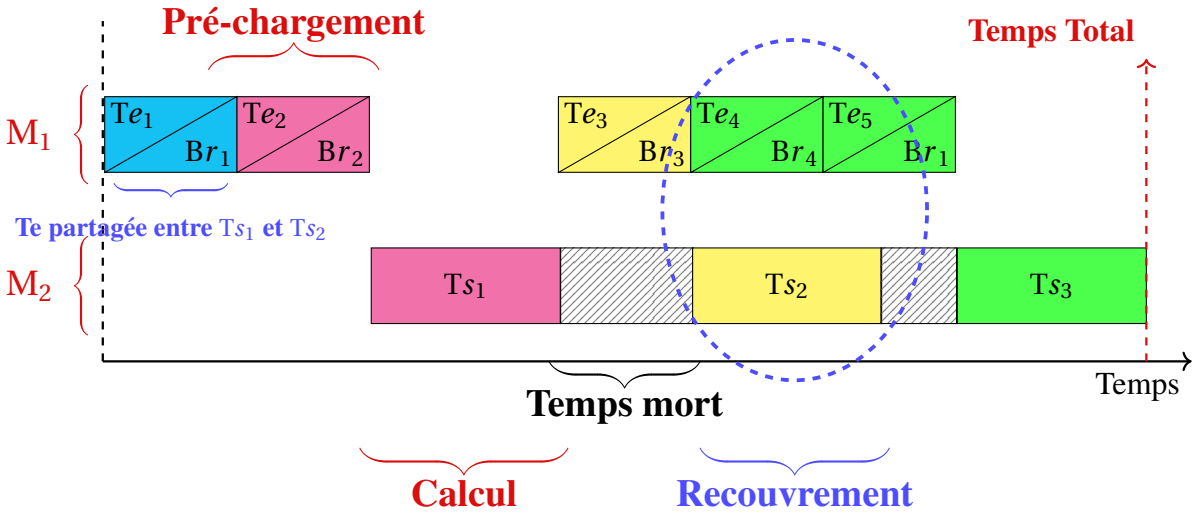


FIGURE 3.1 – Un séquencement type des préchargements et des calculs

Dans le séquencement type donné par la Figure 3.1, nous distinguons trois états possibles (cf. Tableau 3.1) de buffers :

— un buffer est dit « **libre** » s'il n'existe aucune tuile d'entrée qui est en cours de préchargement ou se trouve dans ce buffer, c-à-d il n'y a pas de préchargement (case vide).

— un buffer est dit « **utilisé** » si la tuile d'entrée à précharger est transférée depuis la mémoire externe vers ce buffer, c-à-d un préchargement est en cours (case apparaît en gris).

— un buffer est dit « **préchargé** » si la tuile d'entrée à précharger existe dans ce buffer, c-à-d un préchargement déjà fini (case apparaît en couleur : bleu, vert ou magenta).

Br_1			Te_1									Te_5
Br_2					Te_2							
Br_3										Te_3		
Br_4											Te_4	

TABLEAU 3.1 – États de buffers par rapport aux étapes de préchargement

« 3-PSDPP »

En résumé, le problème 3-PSDPP est défini comme un problème d'ordonnancement multi-objectif sur deux machines en parallèle, où les préchargements et les calculs peuvent être effectuées simultanément tout en respectant la contrainte liée à la configuration de l'image de sortie par rapport à celle de l'entrée.

3.2.2 Les différentes suppositions

Après plusieurs échanges avec le concepteur de l'atelier MMOpt (Mancini et al. [89]), une liste fine des différentes suppositions est définie par les points suivants :

- Les tuiles d'entrée sont toutes de taille uniforme.
- Les tuiles de sortie sont toutes de taille uniforme.
- La taille des tuiles d'entrée et la taille des tuiles de sortie peuvent être différentes.
- Les tuiles d'entrée ne peuvent pas être préchargées simultanément ; une tuile d'entrée à la fois.
- Le temps de préchargement des tuiles d'entrée est constant et identique pour toutes les étapes des préchargements.
- Chaque tuile d'entrée occupe exactement un et un seul buffer ; les tuiles ne se décomposent pas (tuiles non préemptives ou aussi non morcelables).
- Il n'y a pas de distinction entre les buffers, à savoir qu'une tuile d'entrée peut être préchargée dans n'importe quel buffer.
- Le sous-ensemble de tuiles d'entrée requises pour le calcul de chaque tuile de sortie est fixé à l'avance (données déterministes).
- Les tuiles de sortie doivent être successivement calculées ; une tuile de sortie à la fois.
- Le temps de calcul des tuiles de sortie est constant et identique pour toutes les étapes des calculs.
- Les préchargements et les calculs peuvent être effectuées simultanément. Cela signifie qu'une ou plusieurs étapes de préchargement peuvent s'effectuer séquentiellement pendant qu'une étape de calcul est en cours du traitement.

3.3 La formulation mathématique Vs. la PLNE

En RO, la formulation par la PLNE est l'approche la plus répondue pour la modélisation et la résolution d'une grande variété des problèmes d'optimisation. En effet, la PLNE traduit la modélisation naturelle du problème par la définition d'une série de variables entières qui peuvent être simplement booléennes, c'est-à-dire ne prendre que les valeurs 0 et 1, des contraintes linéaires et une fonction de coût.

La PLNE est aussi considérée comme une méthode efficace pour la résolution exacte de problèmes. En revanche, la contrainte d'intégrité sur les variables rend le problème plus complexe et demande des techniques particulières pour sa résolution en un temps raisonnable, tels que les méthodes de coupes et l'ajout des contraintes pour la rupture de symétrie.

Mais, la modélisation par la PLNE présente quelques inconvénients majeurs dans le sens où il est difficile pour le lecteur de distinguer les données d'entrée et de sortie en se basant uniquement sur les variables décrites dans la formulation donnée. En plus, plusieurs formulations

différentes avec de nombreuses variables peuvent être proposées pour un seul problème. Ceci reflète la complexité et l'illisibilité pour comprendre les données du problème face à la grande variété de ces modèles.

Pour simplifier la compréhension de notre problème d'origine 3-PSDPP, nous avons choisi de le formuler par un modèle mathématique avec des données d'entrées et de sorties clairement définies. Cette formulation mathématique permet de traduire le problème électronique réel afin de mieux caractériser les données d'entrée ainsi que les variables à déterminer et aussi de bien cerner les objectifs visant leur optimisation (à minimiser ou à maximiser) tout en respectant les différentes contraintes mises en œuvre.

En outre, elle peut être vue comme une approche de modélisation très souple qui permet de bien caractériser de nombreux sous-problèmes avec des données d'entrée et des variables de sortie spécifiques.

En revanche, nous proposons dans la Section 3.6, une liste des formulations par la PLNE pour certaines variantes mono-objectif dérivées du problème 3-PSDPP. Ces différents modèles seront utiles pour la phase de leurs résolutions de manière exacte qui sera abordée en détail dans le Chapitre 5.

3.4 Formulation mathématique du problème 3-PSDPP

Le but de cette troisième partie est de présenter la formulation mathématique associée à notre problème 3-PSDPP, c'est-à-dire les *données d'entrée*, les *variables à déterminer*, les *contraintes* ainsi que les *objectifs* à optimiser.

D'une manière plus formelle, cette modélisation est résumée dans le Tableau 3.2 ci-dessous, suivi d'une description complète et détaillée de chacune des composantes ainsi qu'un exemple illustrant ce modèle mathématique.

Entrées	$\mathscr{X} = \{1,\dots,X\}, \mathscr{Y} = \{1,\dots,Y\}$, où $X, Y \in \mathbb{N}^*$
	$\mathscr{R}_y \subseteq \mathscr{X}, \forall\ y \in \mathscr{Y}$
	$\alpha, \beta\ \in \mathbb{N}^*$
Sorties	$(p_i)_{i \in \mathscr{N}}$, où $p_i = (d_i, b_i, t_i)$, $\mathscr{N} = \{1,\dots,N\}$ et $N \in \mathbb{N}^*$
	$(c_j)_{j \in \mathscr{M}}$, où $c_j = (s_j, u_j)$, $\mathscr{M} = \{1,\dots,M\}$ et $M = Y$
	(N, Z, Δ), où $N, Z, \Delta \in \mathbb{N}^*$
Contraintes	**(1)** $\forall\ y\ \in \mathscr{Y}, \exists\ j \in \mathscr{M}\ /\ s_j = y$
	(2) $\forall\ j \in \mathscr{M}, \forall x \in \mathscr{X}, x \in \mathscr{R}_{s_j} \Rightarrow (\exists\ a \in \{1,\dots,u_j - \alpha\}, \exists\ i \in \mathscr{N}\ /\ t_i = a, d_i = x$
	$\&\ (\forall a' \in \{a+\alpha,\dots,u_j+\beta-1\}, \forall i' \in \{i+1,\dots,N\}, t_{i'} = a' \Rightarrow b_{i'} \neq b_i))$
	(3) $\forall\ i \in \mathscr{N}\setminus\{1\}, t_i\ \geq\ t_{i-1} + \alpha$
	(4) $\forall\ j \in \mathscr{M}\setminus\{1\}, u_j\ \geq\ u_{j-1} + \beta$
Objectifs	$\min Z, \min N, \min \Delta$

TABLEAU 3.2 – Formulation Mathématique pour le problème 3-PSDPP

3.4.1 Les données d'entrée

Une instance I du problème 3-PSDPP peut être définie par le quintuplet $(\mathscr{X}, \mathscr{Y}, \mathscr{R}_y, \alpha, \beta)$, où :

— $\mathscr{X}$ désigne l'ensemble de tuiles d'entrée pour l'image d'entrée à précharger de la mémoire externe vers les mémoires internes.

— $\mathscr{Y}$ désigne l'ensemble de tuiles de sortie pour l'image de sortie à calculer.

— $\mathscr{R}_y, y \in \mathscr{Y}$ désigne l'ensemble de tuiles d'entrée requises par chaque tuile de sortie y. Ces ensembles peuvent être présentés par une matrice d'incidence Tuiles d'entrée/Tuiles de sortie X × Y (matrice booléenne $0-1$), noté r_{xy}, où les tuiles d'entrée $\mathscr{X}$ présentent les X lignes et celles de sortie $\mathscr{Y}$ définissent les Y colonnes. Cette matrice est définie comme suit :

$$r_{xy} = \begin{cases} 1, \text{si la tuile d'entrée } x \text{ est requise pour calculer la tuile de sortie } y \\ 0, \text{sinon} \end{cases}$$

Autrement dit, $r_{xy} = 1$ si $x \in \mathscr{R}_y, \forall x \in \mathscr{X}, \forall y \in \mathscr{Y}$.

— α désigne la durée nécessaire pour précharger une tuile d'entrée (la durée est uniforme pour tous les préchargements).

— β désigne la durée nécessaire pour calculer une tuile de sortie (la durée est uniforme pour tous les calculs).

Remarque 3.1. *La relation binaire entre l'image d'entrée et celle de sortie, définie par l'ensemble* $\mathscr{R}_y, \forall y \in \mathscr{Y}$ *ou également la matrice d'incidence* $r_{xy}, \forall x \in \mathscr{X}$ *et* $y \in \mathscr{Y}$*, est représentée par un Hypergraphe* [1].

On note l'hypergraphe $H_T = (V_T, E_T)$ *représentant la description de "Tuiles Entrée/Tuiles Sorties", que nous appelons* **Tile Hypergraph***, tel que :*

— *l'ensemble de sommets* V_T *est l'ensemble de tuiles de l'image d'entrée, c-à-d* $V_T = \mathscr{X}$ *;*

— E_T *est l'ensemble de hyperarêtes de* H_T*, où chaque hyperarête* $E_y, \forall y \in \mathscr{Y}$ *est donnée par l' ensemble* $\mathscr{R}_y, \forall y \in \mathscr{Y}$.

L'hypergraphe H_T *correspond de manière univoque à la matrice* $\mathscr{H}_{XY}$ *telle que :*

$$\forall h_{xy} \in \mathscr{H}_{XY}, \quad h_{xy} = \begin{cases} 1 & si\ v_x \in E_y \\ 0 & sinon \end{cases}$$

Nous notons aussi que le graphe H_T *peut être aisément transformé en un graphe biparti* [2] *: l'ensemble de sommets* V *peut être divisé en deux sous-ensembles* V_T^1 *et* V_T^2*, où :*

— V_T^1 *: les tuiles d'entrée à précharger, c-à-d* $V_T^1 = \{X_x / x \in \mathscr{X}\}$.

— V_T^2 *: les tuiles de sortie à calculer, c-à-d* $V_T^2 = \{Y_y / y \in \mathscr{Y}\}$.

1. Hypergraphe H : est la donnée d'un couple (V, E), où $V = \{v_1, v_2, \ldots, v_n\}$ est un ensemble fini et $E = E_1, E_2, \ldots, E_m$ est un sous-ensemble de parties non vides de V. Les éléments de V sont les sommets et les éléments de E sont les hyperarêtes de H (ce sont des sous-ensembles de sommets).

2. Un graphe G = (V, E) est appelé biparti si son ensemble de sommets V peut être partitionné en deux ensembles de sommets non vide V_1 et V_2 tels qu'aucun couple de sommets V_1 (respectivement de V_2) ne soit adjacent. Il est bien connu qu'un graphe est *biparti* si et seulement s'il ne contient pas de cycle impair.

3.4.2 Les variables à déterminer

Une solution réalisable S à une telle instance I du problème 3-PSDPP est définie par le quintuplet $\big(\mathrm{N}, (p_i)_{i\in\mathcal{N}}, \mathrm{Z}, (c_j)_{j\in\mathcal{M}}, \Delta\big)$, où :

— Le séquencement des préchargements $(p_i)_{i\in\mathcal{N}}$, où i est une étape de préchargement et $p_i = (d_i, b_i, t_i)$, avec $\mathcal{N} = \{1, \ldots, \mathrm{N}\}$ et $\mathrm{N} \in \mathbb{N}^*$. Le triplet (d_i, b_i, t_i) désigne, pour toute étape de préchargement $i, i \in \mathcal{N}$, quelle tuile d'entrée d_i doit être préchargée, dans quel buffer b_i et à quel instant de préchargement t_i (date de début du préchargement i).

— Le séquencement des calculs $(c_j)_{j\in\mathcal{M}}$, où j est une étape de calcul et $c_j = (s_j, u_j)$, avec $\mathcal{M} = \{1, \ldots, \mathrm{M}\}$ et $\mathrm{M} = \mathrm{Y}$. Le couple (s_j, u_j) désigne, pour toute étape de calcul $j, j \in \mathcal{M}$, quelle tuile de sortie s_j doit être calculée et à quel instant de calcul u_j (date de début du calcul j).

— Le triplet $(\mathrm{Z}, \mathrm{N}, \Delta)$, où $\mathrm{Z}, \mathrm{N}, \Delta \in \mathbb{N}^*$ définissent les valeurs pour les trois critères à minimiser.

Remarque 3.2. *Il est nécessaire de noter que* N *et* Z *sont considérés comme données de sortie du modèle. En particulier,*

— Z *est utilisé pour déterminer la séquence de destinations* $(b_i)_{i\in\mathcal{N}}$ *;*

— N *est utilisé pour déterminer la séquence de destinations* $(d_i)_{i\in\mathcal{N}}$ *;*

3.4.3 Les contraintes

Ces différentes contraintes sont définies formellement par les équations (1), (2), (3) et (4) dans le *modèle mathématique* décrit par le Tableau 3.2 ci-dessus. La signification de chaque contrainte est donnée comme suit :

— La contrainte (1) assure que chaque tuile de sortie $y, \forall\, y \in \mathcal{Y}$ doit être calculée au moins une fois. C'est-à-dire, à chaque étape de calcul $j, \forall\, j \in \mathcal{M}$, on associe une unique tuile de sortie y.

— La contrainte (2) assure que les tuiles d'entrée requises pour calculer une tuile de sortie, données par $\mathcal{R}_y, \forall\, y, y \in \mathcal{Y}$, soient accessibles dans les buffers au moment de ce calcul (s_j). Autrement dit, pour un calcul à l'étape j, toutes les tuiles d'entrée x nécessaires pour ce calcul, où $x \in \mathcal{R}_{s_j}$, se trouvent dans les buffers à la date $t_i, t_i \in \{1, \ldots, u_j - \alpha\}$ et ne seront pas écrasées jusqu'à l'instant de fin de ce calcul. En particulier, les dates de fin de leurs préchargements doivent être inférieures ou égales à la date de début de ce calcul ainsi que les buffers utilisés ne seront pas réutilisés s'il existe des nouveaux préchargements en cours.

— La contrainte (3) garantit que le préchargement à l'étape i ne commence que lorsque le préchargement à l'étape $i-1$ est fini (pas de chevauchement entre les préchargements).

— La contrainte (4) garantit que le calcul à l'étape j ne commence que lorsque le calcul à l'étape $j-1$ est fini (pas de chevauchement entre les calculs).

Remarque 3.3. *Une tuile d'entrée x déjà préchargée peut être réutilisée si elle est encore présente dans un buffer.*

3.4.4 Les objectifs

Comme nous l'avons signalé dans la Section 3.2.1, nous avons identifié trois critères d'optimisation liés au problème 3-PSDPP, qui sont les suivants :

— Z : le nombre de buffers du TPU.

— N : le nombre total de préchargements, c'est-à-dire le nombre total de tuiles d'entrée préchargées (une tuile d'entrée peut être préchargée plusieurs fois).

— Δ : le temps total de traitement. C'est-à-dire le temps total nécessaire pour effectuer toute l'opération du TPU depuis la première étape de préchargement jusqu'à la fin du calcul de la dernière tuile.

3.4.5 Exemple

Afin d'illustrer la formulation mathématique de notre problème d'ordonnancement 3-PSDPP, où les trois critères d'optimisation Z, N et Δ doivent être simultanément minimisés, nous considérons l'exemple du problème P_1 donné ci-après.

Les données d'entrée du problème P_1 sont définies par le quintuplet $(\mathscr{X}, \mathscr{Y}, \mathscr{R}_y, \alpha, \beta)$, où :

— $\mathscr{Y} = \{Y_j / 1 \leq j \leq 5\}$ de Y = 5 tuiles de sortie à calculer,

— $\mathscr{X} = \{X_i / 1 \leq i \leq 4\}$ de X = 4 tuiles d'entrée disponibles à précharger,

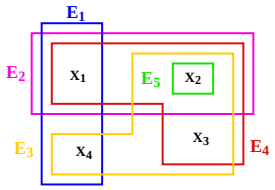
— $\mathscr{R}_y = [\{X_1, X_4\}, \{X_1, X_2\}, \{X_2, X_3, X_4\}, \{X_1, X_2, X_3\}, \{X_2\}]$

— $\alpha = 2$, $\beta = 3$ unités de temps.

La représentation correspondante du problème par une matrice d'incidence r_{xy}, respectivement, par un hypergraphe $H_T = (V_T, E_T)$ est illustrée par les deux Figures 3.2a et 3.2b.

$$M = \begin{array}{c} \\ X_1 \\ X_2 \\ X_3 \\ X_4 \end{array} \begin{array}{c} \begin{array}{ccccc} Y_1 & Y_2 & Y_3 & Y_4 & Y_5 \end{array} \\ \begin{pmatrix} 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 \end{pmatrix} \end{array}$$

(a) La matrice r_{xy}



(b) L'hypergraphe H_T

FIGURE 3.2 – Le problème P_1 - Représentation des données

Comme le montre la Figure 3.2b ci-dessus, l'ensemble $\mathscr{R}_1$ est représenté par l'arête (X_1, X_4), $\mathscr{R}_2$ est défini par l'arête (X_3, X_4), $\mathscr{R}_3$ est représentée par les arêtes (X_2, X_3), (X_2, X_4) et (X_3, X_4), les trois arêtes (X_1, X_2), (X_1, X_3) et (X_2, X_3) représentent $\mathscr{R}_4$ ainsi que $\mathscr{R}_5$ est défini uniquement par le sommet X_2.

Une solution réalisable (les données de sortie) pour le problème P_1 est donnée par la Figure 3.3. Elle utilise Z = 3 buffers pour précharger les tuiles de l'image d'entrée avec un nombre N = 6 de préchargements et un temps total de traitement Δ = 26 unités de temps.

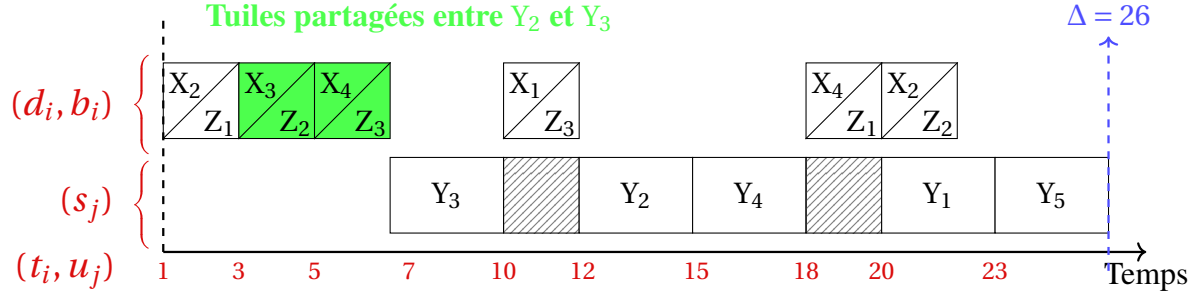


FIGURE 3.3 – Le problème P_1 - Séquencement des préchargements et des calculs

Les valeurs de différentes données de sortie correspondantes au problème P_1 sont résumées dans le tableau 3.3 suivant.

Séquencement des préchargements	$(d_i)_{i\in\mathcal{N}} = \{X_1, X_3, X_4, X_1, X_4, X_2\}$ $(b_i)_{i\in\mathcal{N}} = \{Z_1, Z_2, Z_3, Z_3, Z_1, Z_2\}$ $(t_i)_{i\in\mathcal{N}} = \{1, 3, 5, 10, 18, 20\}$
Séquencement des calculs	$(s_j)_{j\in\mathcal{M}} = \{Y_3, Y_2, Y_4, Y_5\}$ $(u_j)_{j\in\mathcal{M}} = \{7, 12, 15, 20, 23\}$
(Z, N, Δ)	$(3, 6, 26)$

TABLEAU 3.3 – Le problème P_1 - Données de sortie

3.5 Variantes dérivées du problème 3-PSDPP

À partir de notre problème 3-PSDPP, nous dérivons plusieurs variantes mono- et bi-objectif.

3.5.1 Sous-problèmes principaux

En considérant les mêmes données d'entrée que celles du problème d'origine 3-PSDPP, nous distinguons les problèmes mono-objectif sous-jacents suivants :

a) **Minimum Buffers of 3-PSDPP Problem (MB-PSDPP) :** cette variante mono-objectif consiste à minimiser uniquement le nombre de buffers Z, sans contrainte sur N ni sur Δ.

b) **Minimum Prefetches of 3-PSDPP Problem (MP-PSDPP) :** cette variante mono-objectif consiste à minimiser uniquement le nombre total de préchargements N, sans contrainte sur Z ni sur Δ.

c) **Minimum Completion Time of 3-PSDPP Problem (MCT-PSDPP) :** cette variante mono-objectif consiste à minimiser uniquement le temps total de traitement Δ, sans contrainte sur N ni sur Z.

3.5.2 Variantes, où le nombre de buffers Z est fixé

Pour des raisons technologiques comme pour des raisons de coût, le nombre de buffers est limité. En effet, l'atelier MMOpt, face à la problématique du MW, nécessite une quantité limitée de mémoires internes afin de garantir la performance de l'unité de traitement produite TPU. Il est alors important de garder un bon contrôle sur le paramètre Z.

En effet, en reprenant les mêmes données d'entrée du problème 3-PSDPP, avec un nombre de buffers Z fixé, nous considérons les variantes suivantes :

a) **Prefetching and Scheduling Problem (PSP)**, où seul le nombre total de préchargements N doit être minimisé.

b) **Data Prefetching Problem (DPP)** consiste à ordonnancer les préchargements des tuiles pour un séquencement de calculs $(s_j)_{j\in\mathcal{M}}$ donné en entrée afin de minimiser le nombre total de préchargements N.

Le problème DPP est un cas particulier du problème PSP décrit précédemment.

c) **Buffer-Constrained Minimum Completion Time Problem (B-C-MCTP)**, où seul le temps total de traitement Δ doit être minimisé.

d) **2-PSDPP**, où le nombre total de préchargements N et le temps total de traitement Δ doivent être simultanément minimisés.

Remarque 3.4. *Le fait que, dans le contexte du problème PSP, le nombre de buffers* Z *est fixé comme donnée d'entrée et le nombre de préchargements* N *est à minimiser ainsi que le temps total de traitement* Δ *est à déterminer comme donnée de sortie, il est alors encore possible de minimiser ce dernier. Pour cela, nous considérons la variante bi-objectif 2-PSDPP.*

3.5.3 Variantes, où le nombre de préchargements N est fixé

Nous considérons aussi, dans le cas où le nombre de préchargements N est fixé comme donnée d'entrée, les deux variantes suivantes :

a) **Prefetch-Constrained Minimum Buffers Problem (P-C-MBP)**, où seul le nombre de buffers Z doit être minimisé.

b) **Prefetch-Constrained Minimum Completion Time Problem (P-C-MCTP)**, où seul le temps total de traitement Δ doit être minimisé.

La Figure 3.4 ci-dessous définit l'arborescence qui résume les différentes variantes du problème d'origine 3-PSDPP :

Remarque 3.5. *Plusieurs autres variantes bi-objectif du problème d'origine 3-PSDPP peuvent être considérées en se basant sur la combinaison de 2 objectifs tout en fixant le troisième comme une donnée d'entrée ou lorsqu'on doit également le déterminer.*

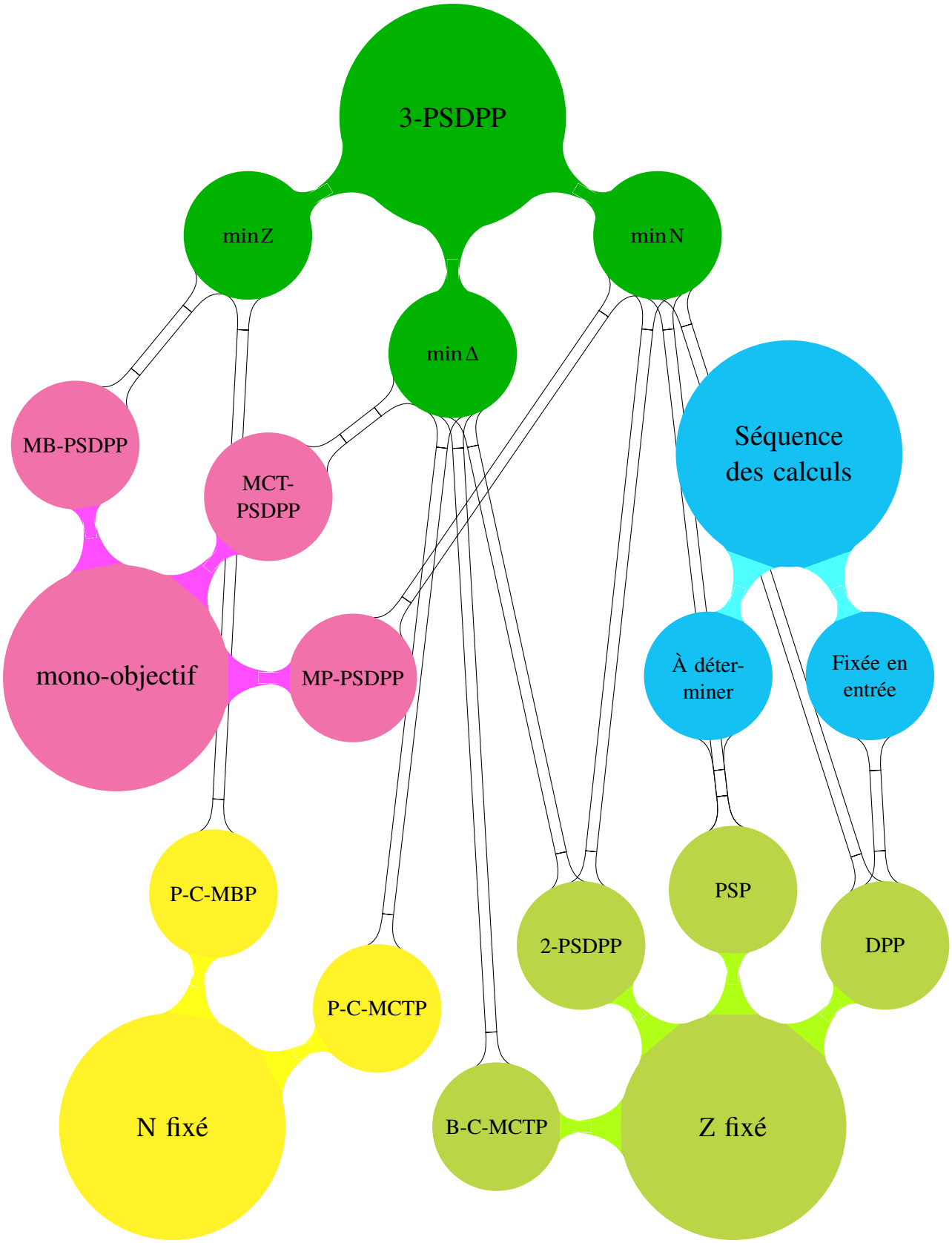


FIGURE 3.4 – Les différentes variantes du problème 3-PSDPP

3.6 Formulations en PLNE pour quelques variantes mono-objectif du problème 3-PSDPP

Dans cette section, nous introduisons les différentes formulations par la PLNE proposées pour les variantes mono-objectif, PSP, MCT-PSDPP et B-C-MCTP, décrites dans la section précédente.

Avant d'illustrer ces formulations, nous définissons la liste des différents ensembles et constantes utiles par la suite :

— $\mathscr{M} = \{1, \ldots, \mathrm{M}\}$, où $\mathrm{M} = \mathrm{Y}$, est l'ensemble de M étapes correspondantes aux tuiles de sortie dans la séquence de calculs à déterminer $(s_j)_{j \in \mathscr{M}}$.

— $\mathscr{N} = \{1, \ldots, \mathrm{N}\}$, où $\mathrm{N} \in \mathbb{N}^*$, est l'ensemble de N étapes correspondantes aux tuiles d'entrée dans la séquence de préchargements à déterminer $(d_i)_{i \in \mathscr{N}}$.

— $\mathscr{Z} = \{1, \ldots, \mathrm{Z}\}$, où $\mathrm{Z} \in \mathbb{N}^*$, est l'ensemble de Z buffers associés à la liste de destinations à déterminer $(b_i)_{i \in \mathscr{N}}$, où $\mathrm{Z} \geq \max_{y \in \mathscr{Y}} |\mathscr{R}_y|$.

— r_{xy} est la matrice d'incidence qui relie chaque tuile d'entrée à la ou les tuiles de sortie qui l'utilise, où :

$$r_{xy} : \begin{cases} 1 & \text{si la tuile d'entrée } x \text{ est requise par la tuile de sortie } y \\ 0 & \text{sinon} \end{cases}$$

Autrement dit, $r_{xy} = 1$ si $x \in \mathscr{R}_y, \forall x \in \mathscr{X}, \forall y \in \mathscr{Y}$.

— $(\mathscr{Y}_x)_{x \in \mathscr{X}}$: désigne l'ensemble de tuiles de sortie $y, \forall y \in \mathscr{Y}$ qui nécessitent la tuile d'entrée x.

3.6.1 Formulation PLNE pour le problème PSP

Dans cette section, nous considérons la formulation en PLNE, notée `PSP-PLNE`, pour modéliser le problème de PSP. Comme ce dernier est équivalent au ToSP, comme on le verra dans le Chapitre 4, la formulation proposée Tang et Denardo (1988) [110] pour le ToSP peut être appliquée directement au PSP. Parce que la formulation dans Tang et Denardo (1988) [110] traite la version sans prendre en compte le nombre de chargements, nous la modifions légèrement, en ajoutant ce nombre à la fonction objectif de la formulation `PSP-PLNE`. L'intérêt d'adapter cette formulation dans le contexte du problème PSP est d'avoir l'optimalité en termes de nombre de préchargements N.

La formulation `PSP-PLNE` est alors donnée ci-après en utilisant une série de trois variables de décision $\{0-1\}$ définie par :

⋆ $a_{yj} : \begin{cases} 1 & \text{si la tuile de sortie } y \text{ est calculée à l'étape } j, \forall y \in \mathscr{Y}, \forall j \in \mathscr{M} \\ 0 & \text{sinon} \end{cases}$

⋆ $b_{xj} : \begin{cases} 1 & \text{si la tuile d'entrée } x \text{ existe dans le buffer lorsque la tuile de sortie à l'étape } j, \\ & \text{est en cours du calcul, } \forall x \in \mathscr{X}, \forall j \in \mathscr{M} \\ 0 & \text{sinon} \end{cases}$

⋆ $c_{xj} : \begin{cases} 1 & \text{si la tuile d'entrée } x \text{ est préchargée dans le buffer au début du calcul de la} \\ & \text{tuile de sortie à l'étape } j, \forall x \in \mathscr{X}, \forall j \in \mathscr{M} \\ 0 & \text{sinon} \end{cases}$

Autrement dit, $c_{xj} = 1$ correspond à une tuile d'entrée nouvellement préchargée, où $j \in \mathscr{M} \setminus \{1\}$.

La formulation PSP−PLNE est alors décrite comme suit :

$$\min \sum_{x\in\mathscr{X}} b_{x1} + \sum_{j\in\mathscr{M}\setminus\{1\}} \sum_{x\in\mathscr{X}} c_{xj}$$

S.C

$$\sum_{y\in\mathscr{Y}} a_{yj} = 1 \qquad \forall\ j\in\mathscr{M} \tag{3.1}$$

$$\sum_{j\in\mathscr{M}} a_{yj} = 1 \qquad \forall\ y\in\mathscr{Y} \tag{3.2}$$

$$\sum_{x\in\mathscr{X}} b_{xj} \le \mathrm{Z} \qquad \forall\ j\in\mathscr{M} \tag{3.3}$$

$$a_{yj} \le b_{xj} \qquad \forall\ y\in\mathscr{Y},\ j\in\mathscr{M},\ x\in\mathscr{R}_y \tag{3.4}$$

$$b_{x,j} - b_{x,j-1} \le c_{xj} \qquad \forall\ x\in\mathscr{X},\ j\in\mathscr{M}\setminus\{1\} \tag{3.5}$$

$$a_{yj}\in\{0,1\} \qquad \forall\ y\in\mathscr{Y},\ j\in\mathscr{M} \tag{3.6}$$

$$b_{xj}, c_{xj}\in\{0,1\} \qquad \forall\ x\in\mathscr{X},\ j\in\mathscr{M} \tag{3.7}$$

Dans cette formulation, la fonction objectif est de minimiser le nombre total de préchargements N, où la valeur $\sum_{x\in\mathscr{X}} b_{x1}$ prend en compte les préchargements dans les buffers lors du premier calcul. En outre, les contraintes suivantes (3.1) — (3.7) doivent être satisfaites pour que la solution soit réalisable.

- La contrainte (3.1) signifie que pour chaque étape de calcul j, il existe une tuile de sortie y qui sera calculée à l'étape j.
- La contrainte (3.2) signifie que pour chaque tuile de sortie y, il existe une étape de calcul j à laquelle y sera calculée.
- La contrainte (3.3) garantit que le nombre de buffers n'est jamais dépassé.
- La contrainte (3.4) signifie que chaque tuile de sortie y peut être exécutée à l'étape j si chaque tuile d'entrée x requise par y est présente dans le buffer au moment du calcul correspondant.
  Autrement dit, on a : $\forall j\in\mathscr{M}, \forall x\in\mathscr{X}, \forall y\in\mathscr{Y}_x,\ a_{yj}=1 \Rightarrow b_{xj}=1$.
- La contrainte (3.5) impose que le chargement d'une tuile doit être compté à chaque fois qu'une tuile d'entrée est présente à l'étape j, mais ne figurait pas dans les buffers à l'étape $j-1$.
  Autrement dit, on a : $\forall x\in\mathscr{X}, \forall j\in\mathscr{M}\setminus\{1\},\ b_{xj}=1\ \&\ b_{xj-1}=0 \Rightarrow c_{xj}=1$.
- Les contraintes (3.6) — (3.7) définissent le domaine des variables.

Remarque 3.6. *La contrainte (3.4) peut être donnée par :*

$$a_{yj} \le b_{xj}, \forall\, j\in\mathscr{M}, x\in\mathscr{X},\ y\in\mathscr{Y}_x \tag{3.8}$$

Cette contrainte (3.4) peut également être réécrite comme suit :

$$\sum_{y\in\mathscr{Y}_x} a_{yj} \le b_{xj}, \forall\, j\in\mathscr{M},\ x\in\mathscr{X} \tag{3.9}$$

Cette nouvelle contrainte garantit que si la tuile de sortie y, où y appartient à l'ensemble $\mathscr{Y}_x$, est calculée à l'étape de calcul j (une et une seule tuile de sortie y est calculée à chaque étape j), alors la tuile d'entrée x existe dans le buffer au moment de ce calcul.

Autrement dit, on a : $\forall j\in\mathscr{M}, \forall x\in\mathscr{X}, \forall y\in\mathscr{Y}_x,\ \sum_{y\in\mathscr{Y}_x} a_{yj}=1 \Rightarrow b_{xj}=1$.

3.6.2 Formulation PLNE pour le problème MCT-PSDPP

La formulation que nous proposons pour le problème MCT-PSDPP, sera notée `MCTP-PLNE`. Dans cette formulation, nous choisissons de prendre comme borne sur le nombre total de préchargements N (borne inférieure) ainsi que sur le nombre de buffers Z (borne supérieure) la valeur X. Cela signifie que chaque tuile d'entrée sera préchargée dans son propre buffer. Par ailleurs, nous utilisons également les variables suivantes :

- ⋆ $c_{yj} : \begin{cases} 1 & \text{si la tuile de sortie } y \text{ est calculée à l'étape de calcul } j, \forall y \in \mathscr{Y}, \forall j \in \mathscr{M} \\ 0 & \text{sinon} \end{cases}$
- ⋆ $d_{xi} : \begin{cases} 1 & \text{si la tuile } x \text{ est préchargée à l'étape de préchargement } i, \forall x \in \mathscr{X}, \forall i \in \mathscr{N} \\ 0 & \text{sinon} \end{cases}$
- ⋆ $u_j, u_j \in \mathbb{N}^*$: l'instant où le calcul d'une tuile de sortie à l'étape $j, \forall j \in \mathscr{M}$ commence.
- ⋆ $t_i, t_i \in \mathbb{N}^*$: l'instant où le préchargement d'une tuile d'entrée à l'étape $i, \forall i \in \mathscr{N}$ commence.
- ⋆ Δ : le temps total de traitement.

De plus, on notera Λ la constante $\alpha * X + \beta * Y$. C'est une borne supérieure sur le temps total de traitement Δ. Nous nous servirons de cette constante dans des contraintes de type "Big M".

La formulation `MCTP-PLNE` s'écrit alors comme suit :

$$\min \Delta$$

S.C

$$\sum_{y \in \mathscr{Y}} c_{yj} = 1 \qquad \forall\ j \in \mathscr{M} \tag{3.10}$$

$$\sum_{j \in \mathscr{M}} c_{yj} = 1 \qquad \forall\ y \in \mathscr{Y} \tag{3.11}$$

$$\sum_{x \in \mathscr{X}} d_{xi} = 1 \qquad \forall\ i \in \mathscr{N} \tag{3.12}$$

$$\sum_{i \in \mathscr{N}} d_{xi} = 1 \qquad \forall\ x \in \mathscr{X} \tag{3.13}$$

$$u_j - t_i \geq \alpha - \Lambda * (3 - r_{xy} - c_{yj} - d_{xi}) \qquad \forall\ y \in \mathscr{Y},\ j \in \mathscr{M},\ x \in \mathscr{X},\ i \in \mathscr{N} \tag{3.14}$$

$$t_{i-1} + \alpha \leq t_i \qquad \forall\ i \in \mathscr{N} \setminus \{1\} \tag{3.15}$$

$$u_{j-1} + \beta \leq u_j \qquad \forall\ j \in \mathscr{M} \setminus \{1\} \tag{3.16}$$

$$\Delta \geq u_M + \beta \tag{3.17}$$

$$c_{yj} \in \{0,1\} \qquad \forall\ y \in \mathscr{Y},\ j \in \mathscr{M} \tag{3.18}$$

$$d_{xi} \in \{0,1\} \qquad \forall\ x \in \mathscr{X},\ i \in \mathscr{N} \tag{3.19}$$

$$u_j \geq 1, entier \qquad \forall\ j \in \mathscr{M} \tag{3.20}$$

$$t_i \geq 1, entier \qquad \forall\ i \in \mathscr{N} \tag{3.21}$$

Dans cette formulation, la fonction objectif est de minimiser le temps total de traitement Δ, où $\Delta = u_M + \beta$, tandis que les contraintes suivantes (3.10) — (3.21) sont satisfaites :

— Les contraintes (3.10) et (3.11) ont la même signification que les contraintes (3.1) et (3.2) dans la formulation `PSP-PLNE`.

— La contrainte (3.12) signifie que pour chaque étape de préchargement i, il existe une tuile d'entrée x qui sera préchargée à l'étape i.

— La contrainte (3.13) signifie que pour chaque tuile d'entrée x, il existe une étape de préchargement i à laquelle x sera préchargée.

— La contrainte (3.14) permet d'exprimer la relation entre les préchargements et les calculs. Elle garantit que si la tuile d'entrée x, préchargée à l'étape de préchargement i, est requise par la tuile de sortie y, calculée à l'étape de calcul j, alors cette tuile doit être présente dans le buffer durant ce calcul. Cela signifie que la date de début de ce calcul u_j doit être supérieure ou égale à une date où la tuile x est présente (par exemple la date $t_i + \alpha$). Autrement dit, on a :
$\forall y \in \mathscr{Y}, \forall j \in \mathscr{M}, \forall x \in \mathscr{X}, \forall i \in \mathscr{N}, r_{xy} = 1 \;\&\; c_{yj} = 1 \;\&\; d_{xi} = 1 \;\Rightarrow\; u_j \geq t_i + \alpha.$

— La contrainte (3.15) garantit que le préchargement à l'étape i ne commence que lorsque le préchargement à l'étape $i-1$ est fini.

— La contrainte (3.16) garantit que le calcul à l'étape j ne commence que lorsque le calcul à l'étape $j-1$ est fini.

— La contrainte (3.17) assure que la valeur de Δ est supérieure ou égale à la date de fin de la dernière tuile de sortie calculée, donnée par $u_{\mathrm{M}} + \beta$.

— Les contraintes (3.18) et (3.21) définissent le domaine des variables.

3.6.3 Formulations PLNE pour le problème B-C-MCTP

Rappelons que le problème B-C-MCTP est une variante mono-objectif du problème MCT-PSDPP, où le nombre de buffers Z est fixé en entrée. Après plusieurs tentatives pour sa modélisation, nous avons retenu les deux formulations, que nous appelons, respectivement `BCMCTP-PLNE1` et `BCMCTP-PLNE2`, présentées dans la suite. Dans ces formulations, nous choisissons de prendre comme valeur pour le nombre total de préchargements N la constante $\sum_{y \in \mathscr{Y}} |\mathscr{R}_y|$. En outre, nous définissons les ensembles suivants :

— $\mathscr{T} = \{1, \ldots, \mathrm{T}\}$, où T est une borne supérieure sur Δ : l'intervalle du temps pour effectuer toutes les étapes de préchargements et celles de calculs. Nous choisissons ici de prendre comme valeur pour T la constante $\alpha * \sum_{y \in \mathscr{Y}} |\mathscr{R}_y| + \beta * \mathrm{Y}$.

— $\mathscr{K} = \{1, \ldots, \alpha - 1\}$: l'intervalle de temps auquel une tuile d'entrée peut être préchargée.

— $\mathscr{S} = \{1, \ldots, \alpha\}$: l'intervalle de temps auquel une étape de préchargement d'une tuile d'entrée a été effectuée (le préchargement est terminé et la tuile est présente dans le buffer).

— $\mathscr{L} = \{1, \ldots, \beta - 1\}$: l'intervalle de temps auquel une tuile de sortie peut être calculée.

3.6.3.1 Formulation `BCMCTP-PLNE1`

La première formulation `BCMCTP-PLNE1` que nous proposons ici permet d'encoder les variables suivantes en se basant sur les dates de début pour les étapes de calcul et celles de préchargement.

⋆ $c_{yt} : \begin{cases} 1 & \text{si la tuile de sortie } y \text{ est en cours de calcul entre les instants } t \text{ et } t+1, \forall y \in \mathscr{Y}, \\ & \forall t \in \mathscr{T} \\ 0 & \text{sinon} \end{cases}$

⋆ $p_{xt} : \begin{cases} 1 & \text{si la tuile d'entrée } x \text{ est en cours de préchargement entre les instants } t \text{ et } t+1, \\ & \forall x \in \mathscr{X}, \forall t \in \mathscr{T} \\ 0 & \text{sinon} \end{cases}$

⋆ e_{xt} : $\begin{cases} 1 & \text{si la tuile d'entrée } x \text{ est présente dans un buffer entre les instants } t \text{ et } t+1, \\ & \forall x \in \mathscr{X}, \forall t \in \mathscr{T} \\ 0 & \text{sinon} \end{cases}$

⋆ Δ : le temps total de traitement.

La formulation `BCMCTP-PLNE1` est décrite comme suit :

$$\min \Delta$$

S.C

$$\sum_{y \in \mathscr{Y}} c_{yt} \leq 1 \qquad \forall\ t \in \mathscr{T} \qquad (3.22)$$

$$\sum_{t \in \mathscr{T}} c_{yt} = \beta \qquad \forall\ y \in \mathscr{Y} \qquad (3.23)$$

$$c_{yt} - c_{yt-1} \leq c_{yt+l} \qquad \forall y \in \mathscr{Y}, t \in \{2, \ldots, T-\beta\},\ l \in \mathscr{L} \qquad (3.24)$$

$$\sum_{x \in \mathscr{X}} p_{xt} \leq 1 \qquad \forall\ t \in \mathscr{T} \qquad (3.25)$$

$$p_{xt} - p_{xt-1} \leq p_{xt+k} \qquad \forall x \in \mathscr{X}, t \in \{\alpha+1, \ldots, T-\beta\},\ k \in \mathscr{K} \qquad (3.26)$$

$$e_{xt} - e_{xt-1} \leq p_{xt-s} \qquad \forall x \in \mathscr{X}, t \in \{\alpha+1, \ldots, T\},\ s \in \mathscr{S} \qquad (3.27)$$

$$\sum_{x \in \mathscr{X}} (e_{xt} + p_{xt}) \leq Z \qquad \forall t \in \mathscr{T} \qquad (3.28)$$

$$c_{yt} \leq e_{xt} \qquad \forall y \in \mathscr{Y}, t \in \mathscr{T},\ x \in \mathscr{R}_y \qquad (3.29)$$

$$e_{xt} = 0 \qquad \forall x \in \mathscr{X},\ t \in \mathscr{S} \qquad (3.30)$$

$$t * c_{yt} \leq \Delta \qquad \forall y \in \mathscr{Y},\ t \in \mathscr{T} \qquad (3.31)$$

$$c_{yt} \in \{0, 1\} \qquad \forall y \in \mathscr{Y},\ t \in \mathscr{T} \qquad (3.32)$$

$$p_{xt}, e_{xt} \in \{0, 1\} \qquad \forall x \in \mathscr{X},\ t \in \mathscr{T} \qquad (3.33)$$

Dans cette formulation, la fonction objectif est de minimiser le temps total de traitement Δ, tandis que les contraintes suivantes (3.22) — (3.33) sont satisfaites.

— La contrainte (3.22) signifie que pour chaque instant t, il existe au plus une tuile de sortie y qui sera calculée à cet instant.

— Les contraintes (3.23) — (3.24) garantissent que pour chaque tuile de sortie y et pour chaque instant t, il existe un intervalle de temps, allant de t à $t+\beta-1$, auquel la tuile y est calculée.

— La contrainte (3.25) signifie que pour chaque instant t, il existe au plus une tuile d'entrée x qui sera pré-chargée à cet instant.

— La contrainte (3.26) signifie que pour chaque tuile d'entrée x et pour chaque instant t, il existe un intervalle de temps, allant de t à $t+\alpha-1$, auquel la tuile x est pré-chargée.

— La contrainte (3.27) garantit que pour chaque instant t, s'il existe une tuile d'entrée x qui est présente à l'instant t mais pas à l'instant $t-1$, alors il existe un instant $t-s$ auquel cette tuile d'entrée x a été pré-chargée.
Autrement dit, on a : $\forall x \in \mathscr{X}, \forall t \in \{\alpha, \ldots, T\}, \forall s \in \{1, \ldots, \alpha\}, e_{xt} = 1\ \&\ e_{xt-1} = 0 \Rightarrow p_{xt-k} = 1$.

— La contrainte (3.28) garantit que le nombre de buffers n'est jamais dépassé.

— La contrainte (3.29) signifie que la tuile de sortie y peut être calculée à l'instant t si et seulement si toutes ses tuiles d'entrée requises $x, \forall x \in \mathscr{R}_y$ sont présentes dans les buffers.

— La contrainte (3.30) garantit qu'aucune tuile d'entrée x n'existe dans le buffer à l'instant $t = 0$.

— La contrainte (3.31) assure que le temps total de traitement Δ est égal à la date de fin du dernier calcul dans la séquence des calculs.

Le temps total de traitement Δ peut être exprimé en introduisant la variable de décision $\{0-1\}$ suivante :

⋆ $\delta_t : \begin{cases} 1 & \text{si le traitement (toutes les étapes de calcul et celles de préchargements) n'est pas encore fini} \\ 0 & \text{sinon (fini)} \end{cases}$

Dans ce cas, nous modifions légèrement la formulation `BCMCTP-PLNE1`, où la fonction objectif sera définie par

$$\min \sum_{t \in \mathcal{T}} \delta_t$$

et la contrainte (3.31) peut être réécrite en utilisant les deux inégalités (3.34) et (3.35) suivantes :

$$\sum_{y \in \mathcal{Y}} c_{yt} \leq \delta_t \quad \forall t \in \mathcal{T} \tag{3.34}$$

$$\delta_{t-1} \geq \delta_t \quad \forall t \in \{2, \ldots, \mathrm{T}\} \tag{3.35}$$

3.6.3.2 Formulation `BCMCTP-PLNE2`

Nous décrivons maintenant la deuxième formulation `BCMCTP-PLNE2` pour le problème B-C-MCTP. À l'inverse de la formulation précédente, nous choisissons ici d'encoder les variables de décision $\{0-1\}$ suivantes en se basant sur les dates de fin pour les étapes de calcul et celles de préchargement.

⋆ $f_{yt} : \begin{cases} 1 & \text{si le calcul de la tuile de sortie } y \text{ finit à l'instant } t, \forall y \in \mathcal{Y}, \forall t \in \mathcal{T} \\ 0 & \text{sinon} \end{cases}$

⋆ $q_{xt} : \begin{cases} 1 & \text{si le préchargement de la tuile d'entrée } x \text{ finit à l'instant } t, \forall x \in \mathcal{X}, \forall t \in \mathcal{T} \\ 0 & \text{sinon} \end{cases}$

⋆ $e_{xt} : \begin{cases} 1 & \text{si la tuile d'entrée } x \text{ est présente dans un buffer à l'instant } t, \forall x \in \mathcal{X}, \forall t \in \mathcal{T} \\ 0 & \text{sinon} \end{cases}$

⋆ $\delta_t : \begin{cases} 1 & \text{si le traitement (calculs et préchargements) n'est pas encore fini} \\ 0 & \text{sinon (fini)}, \forall t \in \mathcal{T} \end{cases}$

La formulation `BCMCTP-PLNE2` est décrite comme suit :

$$\min \sum_{t \in \mathcal{T}} \delta_t$$

S.C

$$\sum_{t\in\mathscr{T}} f_{yt} = 1 \qquad \forall\ y\in\mathscr{Y} \tag{3.36}$$

$$\sum_{s=t-\alpha+1}^{t}\sum_{x\in\mathscr{X}} q_{xs} \le 1 \qquad \forall\ t\in\{\alpha,\dots,\mathrm{T}\} \tag{3.37}$$

$$e_{xt} - e_{xt-1} \le q_{xt-1} \qquad \forall x\in\mathscr{X},\ t\in\{\alpha+1,\dots,\mathrm{T}\} \tag{3.38}$$

$$\sum_{x\in\mathscr{X}} e_{xt} + \sum_{x\in\mathscr{X}}\sum_{s=t}^{t+\alpha-1} q_{xs} \le \mathrm{Z} \qquad \forall t\in\{1,\dots,\mathrm{T}-\alpha\} \tag{3.39}$$

$$f_{yt} \le e_{xs} \qquad \forall y\in\mathscr{Y}, t\in\{\beta,\dots,\mathrm{T}\}, x\in\mathscr{R}_y,\ s\in\{t-\beta+1,\dots,t\} \tag{3.40}$$

$$e_{xs} = 0 \qquad \forall x\in\mathscr{X},\ s\in\mathscr{S} \tag{3.41}$$

$$f_{ys} = 0 \qquad \forall y\in\mathscr{Y},\ s\in\{1,\dots,\alpha*|\mathscr{R}_y|+\beta-1\} \tag{3.42}$$

$$\sum_{s=t-\beta+1}^{t}\sum_{y\in\mathscr{Y}} f_{ys} \le \delta_t \qquad \forall\ t\in\{\beta,\dots,\mathrm{T}\} \tag{3.43}$$

$$\delta_{t-1} \ge \delta_t \qquad \forall t\in\{2,\dots,\mathrm{T}\} \tag{3.44}$$

$$f_{yt}\in\{0,1\} \qquad \forall y\in\mathscr{Y}, t\in\mathscr{T} \tag{3.45}$$

$$q_{xt}, e_{xt}\in\{0,1\} \qquad \forall x\in\mathscr{X},\ t\in\mathscr{T} \tag{3.46}$$

$$\delta_t\in\{0,1\} \qquad \forall t\in\mathscr{T} \tag{3.47}$$

Dans cette formulation, la fonction objectif est de minimiser le temps total de traitement Δ, tandis que les contraintes suivantes (3.36) — (3.47) sont satisfaites.

— La contrainte (3.36) garantit que pour chaque tuile de sortie y, il existe un instant t auquel cette tuile est calculée.

— La contrainte (3.37) signifie que pour chaque instant t, il existe au plus un préchargement d'une tuile d'entrée qui sera fini à cet instant.

— La contrainte (3.38) garantit que pour chaque instant t, s'il existe une tuile d'entrée x qui est présente à l'instant t mais pas à l'instant $t-1$, alors il existe un instant $t-1$ auquel le préchargement de cette tuile d'entrée x a été fini.
Autrement dit, on a : $\forall x\in\mathscr{X}, t\in\{\alpha+1,\dots,\mathrm{T}\},\ e_{xt}=1\ \&\ e_{xt-1}=0\ \Rightarrow\ q_{xt-1}=1$.

— La contrainte (3.39) garantit que le nombre de buffers n'est jamais dépassé.

— La contrainte (3.40) signifie que le calcul de la tuile de sortie y sera fini à l'instant t si toutes ses tuiles d'entrée x, pour x dans $\mathscr{R}_y$, sont présentes dans les buffers à l'instant s.
Autrement dit, on a : $\forall y\in\mathscr{Y}, t\in\{\beta,\dots,\mathrm{T}\}, x\in\mathscr{R}_y,\ s\in\{t-\beta+1,\dots,t\},\ f_{yt}=1\ \Rightarrow\ e_{xs}=1$.

— Les contraintes (3.41) — (3.42) sont deux contraintes d'initialisation, où la (3.41) garantit qu'aucune tuile d'entrée x n'existe dans le buffer à l'instant s et (3.42) garantit qu'aucun calcul d'une tuile de sortie y n'est fini à l'instant s.

— Les contraintes (3.43) — (3.44) assure que le temps total de traitement Δ est égal à la date de fin du traitement.

Remarque 3.7. *La contrainte (3.40) peut être réécrite comme suit :*

$$\beta*|\mathscr{R}_y|*f_{yt} \le \sum_{x\in\mathscr{R}_y}\sum_{s=t-\beta+1}^{t} e_{xs} \quad \forall y\in\mathscr{Y},\ t\in\{\beta,\dots,\mathrm{T}\} \tag{3.48}$$

Cette contrainte garantit que pour chaque tuile de sortie y et pour chaque instant t, si le calcul de la tuile de sortie y finit à l'instant t, alors il existe un intervalle du temps $\{t-\beta+1,\dots,t\}$

auquel toutes les tuiles requises par y (données par l'ensemble $\mathscr{R}_y$) sont préchargées, c'est-à-dire la somme $sum^{t}_{s=t-\beta+1} e_{xs}$ doit être supérieure ou égale au temps nécessaire pour une étape de calcul (β) multiplié par le nombre de tuiles requise par la tuile y.

Par exemple, pour une instance quelconque avec Y = 10 *tuiles de sortie à calculer et* X = 9 *tuiles d'entrée à précharger, l'inégalité (3.40) compte* 8795 *contraintes alors que la (3.41) compte* 2174 *contraintes. On peut alors dire que cette inégalité permet de réduire le nombre des contraintes qui a un impact significatif sur la performance de résolution.*

Remarque 3.8. *Le nombre de buffers* Z *étant fixé comme une donnée d'entrée dans le contexte de deux variantes PSP et B-C-MCTP, les formulations* `BCMCTP-PLNE1` *et* `BCMCTP-PLNE2`*, peuvent être réutilisées comme deux formulations pour la variante PSP en introduisant une nouvelle fonction objectif définie par :*

$$\sum_{t\in\mathscr{T}}\sum_{x\in\mathscr{X}} q_{xt} \quad \text{, ou en utilisant } \sum_{t\in\mathscr{T}}\sum_{x\in\mathscr{X}} p_{xt} \tag{3.49}$$

Ces deux expressions comptent le nombre de total de préchargements N.

3.7 Conclusion

Comme nous l'avons vu tout au long de ce chapitre, le problème liée à l'optimisation du fonctionnement de MMOpt est un problème multi-objectif 3-PSDPP dont les trois critères à optimiser sont : le coût (surface du circuit produit), la consommation d'énergie et le temps (performance du TPU). Nous avons d'abord décrit d'une manière informelle ce problème. Ensuite, un modèle mathématique, décomposé en données d'entrées, variables à déterminer, contraintes à respecter et objectifs à minimiser, a été proposé. Cette phase de modélisation est fondamentale dans le processus de la RO, dans la mesure où elle conditionne les étapes suivantes. Enfin, une liste de variantes mono- et bi- objectif a été présentée et trois formulations par la PLNE, pour certaines entre elles, ont été introduites.

Chapitre 4

État de l'Art et Analyse des Modèles

Sommaire

4.1 Introduction

Après avoir présenté le modèle mathématique idéalisé de notre problème d'optimisation d'origine 3-PSDPP ainsi que la liste des différentes variantes mono- et bi-objectif dérivées (Chapitre 3), nous nous intéressons dans ce chapitre à l'analyse de complexité pour certaines d'entre elles, l'état de l'art ainsi que le calcul des bornes inférieures.

Dans un premier temps, nous présentons une revue de l'état de l'art sur les principaux travaux liés à cette thèse, y compris les travaux antérieurs proposés par Mancini et al. [89] ainsi qu'un bref aperçu sur des problèmes voisins trouvés dans la littérature de RO. Nous continuerons ensuite avec une étude comparative afin de positionner notre problème 3-PSDPP par rapport à la littérature. Puis, nous exposons les différentes bornes inférieures liées aux trois critères d'optimisation (surface, consommation d'énergie et performance). Finalement, nous présentons les principaux résultats pour établir la complexité (polynomiale et $\mathcal{NP}$-Complet) de certaines variantes du problème 3-PSDPP.

4.2 État de l'Art

Dans cette partie, nous présentons d'abord en détails l'approche de résolution proposée par Mancini et al. [89] pour résoudre la problématique d'optimisation liée au fonctionnement des circuits produits par l'outil MMOpt. Nous abordons par la suite les recherches qui ont été menées sur des problèmes voisins en nous penchant plus particulièrement sur les problèmes d'outillage et leurs applications.

4.2.1 Travaux antérieurs

Les premiers travaux concernant l'optimisation du fonctionnement des TPUs produits par l'outil MMOpt ont été menés par son concepteur Mancini et al. [89, 90]. Leur approche de résolution est la seule proposition générique qui permet de résoudre le problème d'origine 3-PSDPP dont l'étude fait l'objet de cette thèse. En effet, le processus d'optimisation proposé vise à générer un ordonnancement des calculs des tuiles de sortie ainsi que des chargements de données de façon à minimiser la quantité de buffers, la quantité de données chargées depuis la mémoire externe ainsi que le temps de calcul total.

Ce processus est découpé en cinq étapes. Les détails sur ces étapes ont été abordés dans une communication au sein du premier colloque d'informatique en parallélisme, architecture et système (ComPAS 2013) [90].

Suite à notre phase de modélisation du problème d'origine 3-PSDPP, nous présentons maintenant un bref aperçu sur cette approche, en quatre étapes simples seulement.

a) **Étape 1 - Calculs** : cette étape consiste essentiellement en la recherche d'un ordonnancement des tuiles de sortie, i.e. une séquence de calculs $(s_j)_{j\in\mathcal{M}} = \{s_1, s_2, \ldots, s_M\}$.

   Pour construire la séquence de calculs, les auteurs [89, 90] résolvent une instance du Problème du Voyageur de Commerce Asymétrique (PVCA), ou Asymetric Traveling Salesman Problem (ATSP) en anglais.

   En effet, le PVCA est une variante asymétrique du Problème du Voyageur de Commerce (PVC) classique [1], ou Traveling Salesman Problem (TSP) en anglais, qui est sans doute un des plus vieux problèmes d'optimisation combinatoire et certainement l'un des plus étudiés. Ces deux problèmes sont $\mathcal{NP}$-difficiles (voir Garey et Johnson [48]). En

1. **PVC** : étant donné un graphe complet non orienté à n sommets, chaque arête est munie d'un coût, trouver un cycle qui passe par tous les sommets une et une seule fois qui soit de longueur minimum.

outre, de nombreuses méthodes de résolution, exactes et heuristiques, ont été proposées, parmi lesquelles nous citons l'heuristique *Chained Lin-Kernighan* (voir Lin and Kernighan (1973) [84]). Une implémentation de cette heuristique a été présentée par Applegate et al. [11].

Leur démarche consiste à trouver un *circuit hamiltonien* C de valeur minimale dans le graphe orienté complet $\overrightarrow{G}_T = (V_T, A_T)$, valué par la fonction $\varphi(k, l)$, où :

— V_T : est l'ensemble des sommets du graphe $\overrightarrow{G}_T$. Il correspond uniquement à l'ensemble des tuiles de sortie à calculer $\mathscr{Y}$;

— A_T : est l'ensemble d'arcs du graphe $\overrightarrow{G}_T$. Il existe un arc a_{kl} entre les deux sommets V_{Tk} et V_{Tl}, où k et l sont deux tuiles de sortie à calculer successivement.

— La fonction $\varphi(k, l)$ donne le nombre de tuiles à précharger en plus pour pouvoir calculer la tuile l (donné par $|\mathscr{R}_l \backslash \mathscr{R}_k|$), lorsque les tuiles nécessaires pour k données par $\mathscr{R}_k$ — appelé aussi *empreinte* de la tuile de sortie k [90] — sont déjà préchargées.

La fonction φ est alors définie comme suit :

$$\begin{array}{rccc} \varphi : & A_T & \rightarrow & \mathbb{N} \\ & \overrightarrow{kl} & \mapsto & |\mathscr{R}_l \backslash \mathscr{R}_k| \end{array}$$

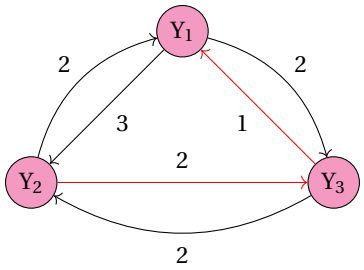
Le coût (poids) total du circuit C est alors défini par le coût cumulé de tous ses arcs : $\sum_{(k,l)\in C} \varphi(k, l)$. Ceci représente, dans le processus d'optimisation, le nombre total de préchargements. Pour tenir compte du coût de la première phase de préchargement (l'ensemble d'une ou plusieurs tuiles d'entrée requises par le premier calcul effectué), deux sommets de départ s et d'arrivée t peuvent être ajoutés pour compléter le graphe $\overrightarrow{G}_T$.

Pour voir concrètement en quoi consiste cette étape, nous considérons l'exemple 4.1 présenté ci-après.

Exemple 4.1. *Nous considérons les données d'entrée définies par la matrice r_{xy} (Figure 4.1a), la construction du graphe $\overrightarrow{G}_T(V_T, A_T)$ est alors donnée par la Figure 4.1b.*

$$r_{xy} = \begin{array}{c} \\ X_1 \\ X_2 \\ X_3 \\ X_4 \\ X_5 \\ X_6 \end{array} \begin{array}{c} \begin{array}{ccc} Y_1 & Y_2 & Y_3 \end{array} \\ \begin{pmatrix} 0 & 1 & 1 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 0 \end{pmatrix} \end{array}$$

(a) La matrice r_{xy}



(b) Le graphe $\overrightarrow{G}_T$

FIGURE 4.1 – Exemple illustrant l'étape 1 "Calculs"

Le circuit hamiltonien C correspondant est alors donné par la succession : $s \rightarrow Y_2 \rightarrow Y_3 \rightarrow Y_1 \rightarrow t$, avec un coût égal à 7. L'ordonnancement associé à la séquence de calculs $s_j, j \in \{1, \ldots, 3\}$ est alors défini par $\{s_1, s_2, s_3\} = \{Y_2, Y_3, Y_1\}$.

b) **Étape 2 - Pré-chargements** : cette étape consiste à construire la séquence de préchargements de tuiles de l'image d'entrée associée à l'ordonnancement de tuiles de sortie

produit à l'étape 1. Les échanges avec la mémoire centrale (N) sont alors minimisées en optimisant cet ordonnancement. En effet, en parallèle de chaque étape de calcul, on précharge les tuiles d'entrée supplémentaires nécessaires pour le prochain calcul. Autrement dit, une tuile d'entrée préchargée pour le calcul à l'étape j, peut être réutilisée uniquement pour le calcul à l'étape $j+1$ si elle est requise par ce dernier.

Par exemple, si l'on reprend l'exemple 4.1, la séquence de préchargements correspondante à la séquence de calculs $\{Y_2, Y_3, Y_1\}$ est définie par $\{X_1, X_2, X_5, X_6, X_3, X_4, X_5\}$ avec un nombre total de préchargements N égal à 7. En effet, pour calculer la tuile Y_1, les deux tuiles X_3 et X_4 sont déjà dans les buffers (préchargées pour calculer la tuile Y_3), et une seule tuile supplémentaire X_5 doit être préchargée.

c) **Étape 3 - Retarder des calculs** : cette étape cherche à réduire la taille de la mémoire interne (le nombre de buffers Z) en ré-ordonnançant le calcul des tuiles de sortie par l'insertion de calculs "fantômes" (aussi appelé "temps mort"). Il est alors possible de choisir Z qui est dans l'intervalle $[Z_{min}, Z_{max}]$, avec $Z_{min} = \max_{y \in \mathscr{Y}} |\mathscr{R}_y|$ (la plus grande empreinte) et Z_{max} correspond au nombre initial de buffers utilisé pour effectuer les N préchargements déterminés à l'étape 2.

Autrement dit, cette étape consiste à retarder certains calculs de tuiles de sortie pendant lesquels le préchargement d'une ou plusieurs tuiles d'entrée est impossible. L'insertion de "fantômes" se traduit alors par un accroissement du temps total de traitement Δ. De nombreux compromis entre la surface (nombre de buffers) et le temps sont ainsi rendus possibles et laissés au libre arbitre du concepteur. L'outil MMOpt expose l'ensemble des compromis possibles et calcule automatiquement un séquencement à partir du choix du concepteur.

Avec l'exemple 4.1, une nouvelle séquence de calculs s_j, $j \in \{1, \ldots, 5\}$, donnée $\{s_1, s_2, s_3\} = \{Y_2, \text{fantôme}, Y_3, \text{fantôme}, Y_1\}$, nous amène à un nombre de 4 buffers. En effet, il n'y a plus de préchargement pendant le calcul s_1. Idem, pendant le calcul s_2. La mémoire interne est alors réduite d'un facteur 1.5.

d) **Étape 4 - Destinations** : cette étape consiste simplement à décider dans quel buffer chaque tuile d'entrée préchargée sera placée, en utilisant le nombre de buffers Z, où $Z \in [Z_{min}, Z_{max}]$, tel que :

— Z_{max} correspond au nombre initial de buffers utilisé à l'étape 2 ;

— Z_{min} est le nouveau nombre de buffers déterminé à l'étape 3 ;

Dans l'exemple 4.1, Z_{max} égal à 6, alors que Z_{min} égal à 4.

Dans le but de simplifier l'exposé du principe de cette approche du point de vue du domaine de la RO, nous la décomposons en deux algorithmes dénommés M_1 et M_2. Ces deux approches visent à résoudre deux sous-problèmes bi-objectif dérivés du problème d'origine 3-PSDPP. En effet, M_1 consiste à minimiser à la fois le nombre total de préchargements N et le temps total de traitement Δ, alors que M_2 consiste à minimiser simultanément le nombre total de préchargements N et le nombre de buffers Z.

Comme le montre l'organigramme donné par la Figure 4.2, l'algorithme M_1 comporte trois étapes : *Calculs*, *Pré-chargements* et *Destinations*. A ces trois étapes, l'algorithme M_2 en rajoute une quatrième : *Retarder des calculs*.

Cette décomposition fait l'objet d'une étude comparative, dans le Chapitre 6, afin d'analyser la performance de nos approches proposées pour la résolution du problème 3-PSDPP par rapport à celles qui sont actuellement utilisées dans l'atelier MMOpt.

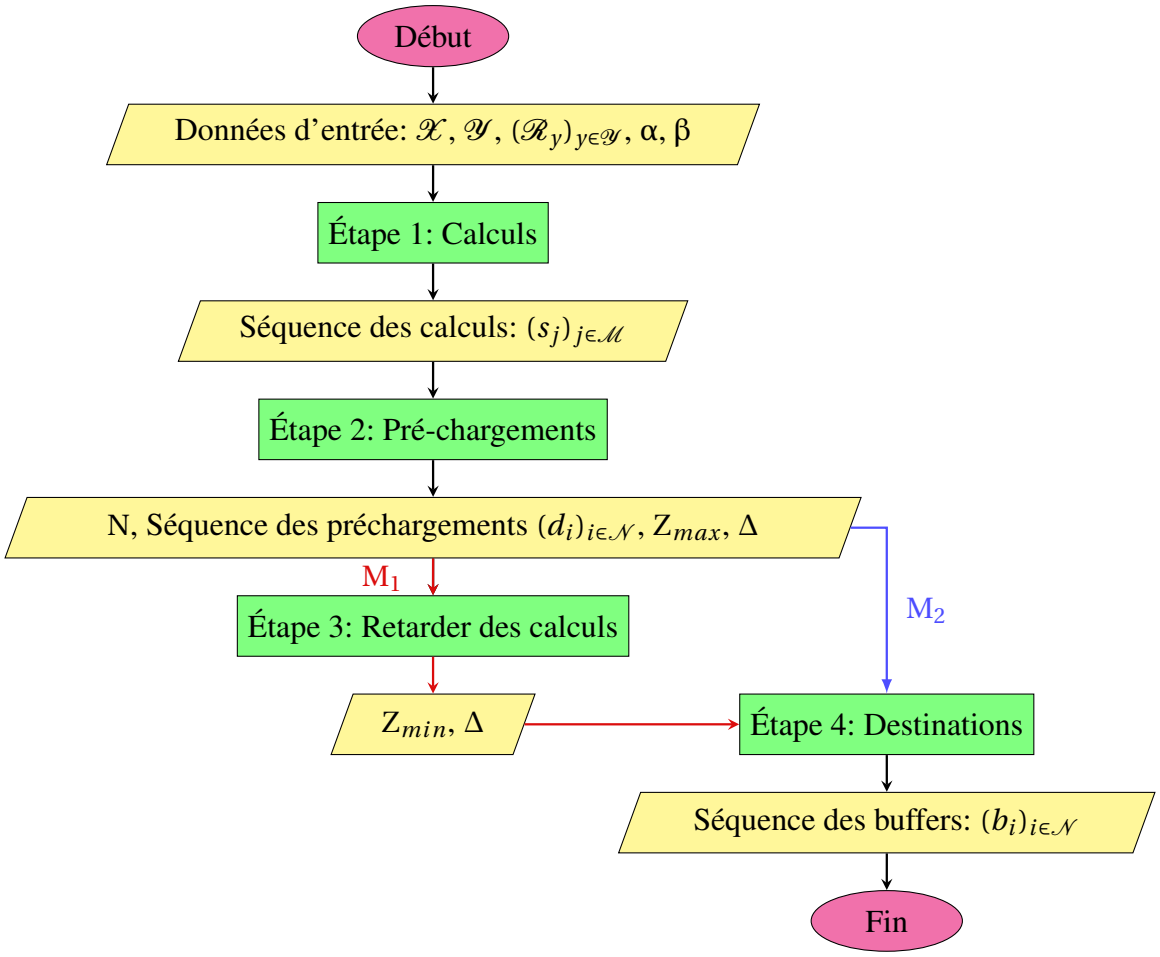


FIGURE 4.2 – Organigramme pour les algorithmes M_1 et M_2

4.2.2 Problèmes similaires trouvés dans la littérature

A notre connaissance, le problème d'ordonnancement multi-objectif 3-PSDPP n'a pas été étudié avant dans la littérature de RO. Au cours de l'étude que nous allons en faire, nous recensons une variété de problèmes voisins. Dans cette partie, nous introduisons en particulier les problèmes liés à la gestion d'outils dans un système de fabrication flexible, ou Flexible Manufacturing System (FMS) en anglais, avant de les illustrer par quelques applications réelles.

4.2.2.1 Définition des problèmes d'outillage

Au cours des dernières décennies, la gestion d'outillage est devenue un aspect et constituant essentiel des systèmes de contrôle et de gestion des ateliers flexibles. En effet, les problèmes d'outillage se posent le plus souvent dans les industries, où la multiplicité des outils de découpe rend nécessaire une gestion informatisée de ces ressources. Pour définir la gestion d'outils, on a souvent recours à la formule suivante :

Avoir le bon outil, au bon moment, au bon endroit, afin d'être en mesure de produire le type de pièce demandée dans la quantité demandée [99].

Dans un problème d'outillage, un ensemble de tâches (pièces) $J = \{1,\ldots,N\}$ doit être exécuté (usiné) sur une machine flexible (une seule tâche à la fois) en utilisant un jeu d'outils $T = \{1,\ldots,M\}$. Chaque tâche j nécessite un sous-ensemble d'outils T_j de T qui doit être chargé dans le magasin d'outils d'une capacité C (un ensemble de slots), tel que $\max_{j\in J}|T_j| \leq C$. La relation

entre les tâches et les outils est représentée par la matrice d'incidence Outils/Tâches A_{ij}, où les outils T présentent les M lignes i et les tâches J désignent les N colonnes j tel que :

$$A_{ij} = \begin{cases} 1, \text{si l'outil } i \text{ est requis pour exécuter la tâche } j, \\ 0, \text{sinon.} \end{cases}$$

Il est nécessaire que les outils correspondants soient placés dans le magasin d'outils avant que la phase d'exécution ne commence. Dans le cas où une nouvelle tâche sera exécutée et si le magasin est plein, des changements d'outils peuvent être nécessaires. Un changement d'outil consiste à enlever (démonter ou changer) un outil du magasin et insérer (introduire ou charger) un autre à la place.

En résumé, un problème d'outillage consiste à déterminer l'ordre de passage des tâches et le plan des chargements/changements (appelé aussi montages/démontages ou insertion/suppression) d'outils dans le magasin. C'est-à-dire, déterminer pour chaque tâche quels sont les outils qui doivent être chargés (introduits) dans le magasin, ainsi que les outils qui doivent être démontés (changés) pour leur faire place si le magasin est plein.

Il existe plusieurs variantes de problèmes d'outillage. Ces variantes imposent généralement des restrictions supplémentaires sur :

— Les caractéristiques des outils : taille *uniforme* (les outils sont de même taille) ou *non-uniforme* (les outils sont de taille différente) ;

— La gestion d'outils : *off-line* ou *on-line* ;
Dans le contexte d'une gestion *off-line*, les outils sont gérés en fonction d'une séquence de pièces déjà déterminée, alors que dans le cas d'une gestion *on-line*, les décisions sont prises au fur et à mesure de l'arrivée des informations, sans connaissance préalable de la séquence de pièces.

— Les critères à optimiser : *mono-* ou *bi-objectif*.

Nous allons donner brièvement un aperçu de certains problèmes liés à la gestion d'outillage qui ont été étudiés dans la littérature de la RO. Un critère d'optimalité souvent étudié est la minimisation du nombre total de changements d'outils, ou "Total Number of Tool Switches" en anglais. Cependant, quelques résultats concernant d'autres critères sont établis.

1. **ToSP** : est défini comme un problème d'ordonnancement sur une seule machine dont le critère d'optimisation est uniquement la minimisation du nombre total de changements d'outils, noté N_s. Ce problème considère la gestion *off-line* des changements d'outils de tailles uniformes.
   D'une manière générale, le problème ToSP présente deux aspects : d'une part l'ordonnancement des tâches, d'autre part la gestion des chargements/changements d'outils dans le magasin d'outils. Ces deux aspects sont naturellement liés. En effet, pour exécuter les tâches, on cherche à déterminer quel outil on doit changer à un instant où un nouvel outil est requis afin de minimiser le nombre de changements occasionnés entre chaque couple de tâches. Or comme chaque tâche n'utilise a priori qu'une partie des outils, le nombre de changements dépend des outils qui composent le reste du magasin.
   De nombreux travaux on été effectués sur l'étude de sa complexité ainsi que sa résolution (exacte et approchée). Le Tableau 4.1, donné ci-après, résume les différentes nomenclatures utilisées dans la littérature pour nommer ce problème.
   Comme le montre ce tableau, les quatre nomenclatures (JSP, ToSP, TMP et SSP) désignent le même problème d'origine lié à la gestion d'outillage, alors que la nomenclature TRP correspond à une variante de ce dernier. Nous reviendrons plus en détails sur la différence entre ces deux problèmes dans la prochaine section.

Nomenclature	Référence
JSP : Job Scheduling Problem	(Tang et Denardo, 1988)
TRP : Tool Replacement Problem	(Tang et Denardo, 1988)
ToSP : Tool Switching Problem	(Crama et al., 1994)
TMP : Tool Management Problem	(Privault et Finke, 1995)
SSP : job Sequencing and tool Switching Problem	(Laporte et al., 2003) (Catanzaro et al., 2015)

TABLEAU 4.1 – Nomenclatures proposées pour le problème ToSP

Le problème ToSP, dans le cas où la taille d'outils est non-uniforme, a aussi été étudié. Peu de travaux ont été effectués sur l'analyse de sa complexité ainsi que sa résolution. Nous citons uniquement : Matzliach et al. (2000) [93], Tzur et al. (2004) [114] et Crama et al. (2007) [30].

2. **Minimum Tool Switching Instants Problem (MTSIP)** : est un problème de gestion *off-line* des changements d'outils avec tailles uniformes, où l'on cherche à minimiser le nombre total d'instants de changements, ou "Number of Instants of Switches" en anglais. En effet, ce nombre correspond au nombre de fois où la machine doit être arrêtée pour procéder au changement d'outils, sans avoir à prendre en compte les changements simultanés (coût du changement fixe) durant la période d'arrêt. Tang et Denardo (1988) [111] ont été les premiers à étudier cette variante. Konak (2007) [81] a aussi établi sa résolution.
3. **The Modular Tool Switching Problem (MTSP)** : est une variante du problème classique ToSP où la gestion des changements d'outils est *on-line*. L'étude de ce problème fait l'objet d'une partie de la thèse de C. Privault (1994) [98]. Matzliach et al. (1998) [92] ont également proposé trois heuristiques pour sa résolution dans le cas où la taille d'outils est non-uniforme. Une analyse de sa complexité a aussi été établie par Raduly-Baka et al. (2015) [101].
4. **Multi-Objective Tool Selection Problem (MO-TSP)** : est une variante du problème classique ToSP, où les deux critères d'optimisation "Number of Switches" et "Number of Instants of Switches" doivent être simultanément minimisés. Parmi les études qui ont été effectuées, nous citons celle de Keung et al. (2001) [76]. Les auteurs ont développé un modèle du problème ainsi qu'un algorithme génétique pour sa résolution.

Nous nous focalisons, dans le reste de cette partie, sur l'étude du problème ToSP. Par la suite, nous dressons un panorama des principaux résultats concernant sa complexité et sa résolution ainsi qu'une variété d'applications dérivées de l'étude de ce problème.

4.2.2.2 Principaux résultats de complexité

Nous allons maintenant illustrer les principaux résultats connus à ce jour concernant l'analyse de la complexité du problème ToSP, où l'on cherche à minimiser le nombre total de changements d'outils N_s. Comme le montre le Tableau 4.2, deux variantes ont été étudiées : TP et ToSP.

Dans le cas du problème TP, la séquence des tâches est fixée comme donnée d'entrée. Cette variante est résolue de façon optimale par l'algorithme KTNS qui a été proposée par Tang et Denardo [110]. Cet algorithme est polynomial, avec une complexité en $O(C * M * N)$ [98]. Son principe consiste à respecter les deux règles suivantes :

i) N'insérer d'outils que s'il est requis par la tâche à usiner.

ii) Si le magasin est plein, et qu'un outil doit être éliminé, choisir l'outil qui sera requis de nouveau au plus tard dans la séquence.

Tang et Denardo [110] ont également donné une preuve ad-hoc de son optimalité. Crama et al. (1994)[29] ont ensuite fourni une preuve alternative de cette optimalité en utilisant des matrices d'intervalles, ce qui a permis une généralisation lorsqu'un coût d'installation ("setup cost") arbitraire s_i est donnée pour chaque outil $i \in T$. Ces résultats ont été généralisés dans le cas des coûts de changement ("changeover costs") de la forme d_{ik} lorsque l'outil i est inséré directement après avoir retiré l'outil k par Privault et al. [100].

En revanche, lorsqu'on doit également déterminer l'ordre d'usinage des pièces, le problème ToSP est $\mathcal{NP}$-difficile. Pour le vérifier, C. Privault [98] (Chapitre 1) a établi une *réduction polynomiale* du problème de l'existence d'une *chaîne hamiltonienne* dans un graphe cubique — qui est $\mathcal{NP}$-Complet [48] — vers le problème ToSP pour $C \geq 3$. En outre, Crama et al. [29] ont démontré la $\mathcal{NP}$-difficulté de ToSP en faisant une *réduction polynomiale* du problème de l'existence d'une *chaîne hamiltonienne* dans un Line-graph — qui est $\mathcal{NP}$-Complet [20] — au problème ToSP pour une capacité $C \geq 2$.

	Séquence des tâches	
	fixée en entrée	**à déterminer**
Problème	TP	ToSP
Complexité	Polynomiale ($\in \mathcal{P}$)	$\mathcal{NP}$-difficile
Référence	Tang et Denardo (1988)	Crama et al., (1994) : $C \geq 2$ Privault et al., (1994) : $C \geq 3$
Preuve	Algorithme KTNS	Réduction polynomiale

TABLEAU 4.2 – Variantes & principaux résultats de complexité

4.2.2.3 Aperçu sur les méthodes de résolution

Le problème ToSP étant $\mathcal{NP}$-difficile, plusieurs études ont été menées dans la littérature concernant des techniques d'optimisation pour sa résolution. La plupart de ces travaux abordent la résolution par des méthodes heuristiques et méta-heuristiques. En revanche, Il y a très peu de travaux qui s'intéressent à l'optimisation exacte de ce problème.

Tang et Denardo (1988) [110], puis Bard (1988) [14] sont les premiers à avoir étudié le problème ToSP en proposant une approche exacte en donnant un modèle en PLNE. Mais, cette approche s'est révélée décevante à cause de la taille du problème et du temps d'exécution.

Crama et al. (1994) [29] ont ensuite abordé la résolution approchée de ce problème. Ils ont également proposé une liste de six heuristiques regroupées dans deux catégories principales : stratégies d'amélioration et stratégies de construction. Parmi celles-ci nous citons les heuristiques connues pour la résolution du problème TSP telles que : *Shortest Edge*, *Nearest Neighbour*, *Farthest Insertion Edge* et *Branch and Bound*. Pour chacune de ces méthodes, ils utilisent l'algorithme KTNS pour évaluer la séquence obtenue. Nous pouvons aussi citer les heuristiques *Block Minimization*, *Interval* et les méthodes *Gloutonnes* ainsi que *2-OPT* et *Load and Optimize*.

L'étude de ce problème a aussi fait l'objet de la thèse de C. Privault (1994) [98]. En effet, elle a proposé plusieurs modèles ainsi que des méthodes de résolution de type heuristique telles que : *Stratégie de Regroupement*, *Meilleure Insertion* et *Next Best* [100]. Elle a également étudié

la variante *on-line* de ce problème. Djellab et al. (2000) [36] ont aussi proposé une nouvelle heuristique basée sur une représentation par un *hypergraphe*.

Résoudre les grandes instances du problème ToSP à l'optimalité reste encore un défi : Laporte et al. (2003) [82] ont décrit une nouvelle formulation en PLNE inspirée du problème de TSP. Ils ont développé une approche de B&B pour résoudre des instances avec jusqu'à 25 tâches et 25 outils.

Denizel (2003) [33] a ensuite développé une approche de *décomposition lagrangienne*. Zhou et al. (2005) [119] ont aussi introduit un algorithme *beam-search-based*. Amaya et al.(2008) [8] ont également proposé un *algorithme mémétique* et décrit plusieurs modèles à la base de *coopération hybride* [9]. Ghiani et al. (2010) [49] ont ainsi proposé une nouvelle approche de solution exacte basée sur la modélisation du problème comme un *TSP non linéaire*.

Plus récemment, Catanzaro et al. [21] ont proposé des nouvelles formulations par la PLNE. Ils ont réussi à résoudre des instances générées aléatoirement, proposées dans [82], avec jusqu'à 40 tâches et 60 outils. Les résultats de [21] démontrent la pertinence des formulations considérées dans [21] par rapport à celles de [110] et [82].

Les différents travaux que nous avons cité montrent l'intérêt qui motive l'étude de ce problème d'ordonnancement : les principaux résultats que nous avons recensé concernant en particulier sa résolution exacte permettent aussi d'en apprécier la difficulté.

4.2.2.4 Applications et cas particuliers

Comme nous l'avons vu ci-dessus, le problème ToSP a fait l'objet de nombreux travaux de recherche, car il possède un champ d'applications réelles particulièrement vaste. Parmi ceux-ci, nous citons les problèmes suivants :

1. **Les problèmes de « K-serveurs »**, où ils sont en étroite relation avec la gestion *on-line* des mémoires en informatique :

   Le principe peut être résumé comme suit : sur un réseau de n clients potentiels, on dispose de k serveurs mobiles, avec lesquels on doit répondre *on-line* aux demandes unitaires et successives des clients, tout en optimisant les déplacements des serveurs. Plus formellement, le problème des K-serveurs consiste à planifier et à décider du déplacement de k serveurs identiques sur les n sommets (clients) d'un graphe (complet ou non), suivant une séquence d'appels de ces sommets. Un appel est représenté par le numéro d'un sommet et signifie que l'on doit déplacer un serveur sur ce sommet. Les arêtes du graphe ont une pondération d. Le coût total de réponse à une séquence d'appels est alors donné par la somme des poids des arêtes parcourues par l'ensemble des serveurs.

   Ce type de problème a été introduit par Manasse et al. (1988) [87]. Il a ensuite été considéré dans plusieurs travaux de recherche. Parmi ceux-ci, nous citons Chrobak et al. (1991) [24] et en particulier la thèse de C. Privault (1994) [98], où elle a appliqué la modélisation sous forme de problème de K-serveurs dans le contexte du problème ToSP avec une gestion *on-line* des changements d'outils.

2. **Les problèmes de « Web Caching »** :

   Un *cache web* est une mémoire locale dans laquelle une quantité limitée de données peut être stockée. Lorsque les utilisateurs veulent accéder à un document qui est stocké dans le *cache web*, il n'est pas nécessaire de télécharger à partir du Web. Cela peut conduire à des temps de réponse plus rapides et une congestion de réseau plus faible. Le problème du "Web Caching" consiste alors à déterminer à tout moment quels documents doivent être stockés dans le *cache web* suivant une séquence des demandes des utilisateurs.

   Dans le modèle uniforme, appelé *problème de pagination*, on suppose que chaque document a la même taille : ce problème peut être résolu en temps polynomial [16]. En

revanche, Crama et al.(2007) [30] ont établi une étude comparative entre le problème général du "Web Caching" et la variante polynomiale TP du problème ToSP, où les outils ont une taille *non-uniforme* et la gestion des changements d'outils est de type *on-line*. Pour obtenir plus de détails sur ce problème, nous invitons le lecteur intéressé à se rapporter à [71], [5], [44] et [108].

3. **Autres problèmes d'ordonnancement liés aux FMSs** :

   De nombreux problèmes avec des contraintes liées à l'outillage sur une seule machine, où certains critères d'optimisation tels que le "makespan" et le "total tardiness of jobs" doivent être minimisés, ont été considérés dans plusieurs travaux de recherche. Parmi ceux-ci, nous citons : Hertz et al. (1996) [68], Akturk et al. (2003, 2004, 2007) [1], [2], [3], Chen (2008) [22] et Low et al. (2010) [85]. Ces travaux ont principalement concerné les résultats de complexité, l'étude de cas particuliers ainsi que la résolution (exacte et approchée).

   Toutefois, dans une étude récente, Beezao et al. (2016) [15] s'intéressaient à la modélisation et la résolution du problème Identical Parallel Machines Problem with Tooling Constraints (IPMTC). En effet, ce problème consiste à exécuter un ensemble de tâches sur un ensemble de machines identiques parallèles, en respectant les contraintes liées aux outils requis pour effectuer chaque étape d'exécution, afin de minimiser le "makespan" (le temps total pour traiter l'ensemble des tâches en utilisant toutes les machines disponibles).

4. **Les problèmes d'optimisation liés à la plateforme de Cross-Docking dans le domaine de la logistique** :

   L'étude de ce type de problèmes a fait l'objet de la thèse de R. Larbi (2008) [83], intitulée « Optimisation des séquences d'opérations dans une plateforme de cross-docking », où elle a appliqué des méthodes de résolution du problème classique ToSP, telle que l'algorithme KTNS.

4.3 Où se situe le problème 3-PSDPP dans la littérature ?

La détermination de position de notre problème d'origine 3-PSDPP, par rapport à celles qui existent dans la littérature de la RO, peut faciliter la phase d'étude de complexité de certaines variantes dérivées, comme on le verra dans la Section 4.5. Ce positionnement se fait via ses deux variantes mono-objectif : PSP et DPP.

Notons que dans le contexte de l'atelier MMOpt, l'ensemble des tuiles de l'image d'entrée requises pour le calcul de l'image de sortie est fixé à l'avance (gestion off-line des tuiles). Rappelons aussi que le problème PSP consiste à minimiser le nombre total de préchargements N, où le nombre de buffers Z est fixé en entrée et la séquence de calculs $(s_j)_{j\in\mathcal{M}}$ est à déterminer. En revanche, cette séquence est donnée comme entrée pour la variante DPP.

Dans cette partie, nous nous intéressons particulièrement à positionner ces deux variantes par rapport aux problèmes ToSP et TP, qui ont été décrits dans la Section 4.2.2.

Comme le montrent les deux Tableaux 4.3 et 4.4, donnés ci-après, plusieurs similarités ainsi que quelques différences sont exposées. En effet, une tuile d'entrée $x \in \mathcal{X}$ à précharger correspond à un outil à charger du magasin d'outils, de même une tuile de sortie $y \in \mathcal{Y}$ correspond à une pièce à usiner. La configuration Tuile d'entrée/Tuiles de sortie correspond à la matrice d'incidence Outils/Tâches. Les séquencements des calculs et des préchargements peuvent être vus comme les séquencements d'usinages et de chargements/changements d'outils. Dans le problème PSP, on cherche à minimiser le nombre total de préchargements N, ce qui revient

à minimiser le nombre total de changements d'outils en ajoutant le nombre de chargements effectués pour usiner la première pièce.

PSP/DPP	**ToSP/TP**
Tuiles d'entrée $\mathscr{X}$	Outils
Nombre de buffers $\mathscr{Z}$	Les slots du magasin d'outils
Tuiles de sortie $\mathscr{Y}$	Tâches
Matrice $r_{xy}, \forall (x,y) \in (\mathscr{X},\mathscr{Y})$	Matrice Outils/Tâches
Séquencement des calculs	Séquencement d'usinage des tâches
Séquencement des préchargements	Séquencement des chargements/changements d'outils
Nombre total de préchargements	Nombre total de chargements/changements d'outils

TABLEAU 4.3 – Analogie entre PSP/DPP & ToSP/TP

En revanche, le problème PSP est défini comme un problème d'ordonnancement sur deux machines en parallèle, où les préchargements et les calculs peuvent être effectués simultanément. Deux séquencements (préchargements et calculs) sont alors déterminés, où la gestion des buffers — c'est-à-dire vers quel buffer une étape de préchargement d'une tuile d'entrée est effectuée — et la spécification des dates de début pour chaque étape de préchargement ou calcul sont prises en compte.

Au contraire, le problème ToSP est un problème d'ordonnancement sur une seule machine, où les chargements/changements d'outils et les usinages des pièces s'effectuent en série. En effet, on obtient un seul séquencement qui couvre toutes ces étapes sans à savoir à prendre en considération les dates de début de chaque étape ainsi que l'emplacement de chaque outil dans le magasin d'outils avec une capacité limitée.

PSP/DPP	**ToSP/TP**
En parallèle (2 machines)	En série (1 seule machine)
2 séquencements	1 seul séquencement
Calcul des dates de début d'une étape (calcul et préchargement)	Dates se déduisent des séquencements

TABLEAU 4.4 – Différences entre PSP/DPP & ToSP/TP

4.4 Calcul des bornes

Nous donnons dans cette section une liste de différentes bornes sur les trois critères d'optimisation du problème d'origine 3-PSDPP. L'intérêt qui réside derrière le calcul de ces bornes consiste à considérer ces derniers comme un facteur de performance pour valider l'efficacité des approches de résolution proposées.

Nous notons :

— Ω : l'ensemble des tuiles d'entrée qui ne sont jamais requises pour le calcul d'une tuile de sortie. C'est-à-dire une ligne entièrement constituée de "0" dans la matrice d'incidence Tuile d'entrée/Tuile de sortie $r_{xy}, \forall (x,y) \in (\mathscr{X},\mathscr{Y})$;

— $\mathscr{X}' = \mathscr{X} \setminus \Omega$: l'ensemble des tuiles d'entrée qui sont requises au moins une fois pour le calcul d'une tuile de sortie ;

— X' : le nombre de tuiles d'entrée de l'ensemble $\mathscr{X}'$: $X' = |\mathscr{X}'| = X - |\Omega|$.

La liste des différentes des *bornes inférieures*, ou Lower Bound (LB) en anglais, est donnée par les trois propositions 4.1, 4.2 et 4.3 suivie d'une preuve mathématique 4.1 afin d'établir la véracité de formulations citées.

Proposition 4.1

$\max_{y \in \mathscr{Y}} |\mathscr{R}_y|$ est une borne inférieure sur le nombre de buffers Z pour le problème 3-PSDPP.

Proposition 4.2

X' est une borne inférieure sur le nombre total de préchargements N pour le problème 3-PSDPP.

Proposition 4.3

$\alpha * X' + \beta$, $\alpha * \min_{y \in \mathscr{Y}} |\mathscr{R}_y| + \beta * Y$ et $\max(\alpha * X' + \beta, \alpha * \min_{y \in \mathscr{Y}} |\mathscr{R}_y| + \beta * Y)$ sont trois bornes inférieures sur le temps total de traitement Δ pour le problème 3-PSDPP.

Preuve 4.1

Étant données une instance I du problème 3-PSDPP, où $I = (\mathscr{X}, \mathscr{Y}, \mathscr{R}_y, \alpha, \beta)$ et une solution réalisable S, définie par le quintuplet $\left(N, (p_i)_{i \in \mathscr{N}}, Z, (c_j)_{j \in \mathscr{M}}, \Delta\right)$, pour I, nous observons que toutes les tuiles d'entrée de l'ensemble $\mathscr{X}'$, doivent être préchargées au moins une fois dans un certain buffer. Ainsi, le nombre total de préchargements ne peut pas être inférieur à X'. Alors, X' est une borne inférieure sur le nombre total de préchargements N, et on la note par lb_N.

De plus, lorsqu'une tuile de sortie est calculée, toutes les tuiles d'entrée requises doivent être présentes dans les buffers. Par conséquent, le nombre maximum de tuiles d'entrée requises pour le calcul d'une tuile de sortie, donné par $\max_{y \in \mathscr{Y}} |\mathscr{R}_y|$, est une borne inférieure sur le nombre de buffers Z et on la note par lb_Z.

Enfin, on note par lb_1 le temps total de préchargement nécessaire pour toutes les tuiles d'entrée préchargées ($\alpha * X'$) plus le temps du calcul de la dernière tuile de sortie (β), et par lb_2 le temps du calcul nécessaire pour toutes les tuiles de sortie ($\beta * Y$) plus le temps de préchargement pour les premières tuiles d'entrée ($\alpha * \min_{y \in \mathscr{Y}} |\mathscr{R}_y|$). Par conséquent, le temps total de traitement Δ est borné par le maximum entre lb_1 et lb_2 : $lb_\Delta = \max(lb_1, lb_2)$.

□

4.5 Analyse de Complexité

Cette section est maintenant consacrée aux principaux résultats de complexité de certaines variantes du problème d'origine 3-PSDPP.

4.5.1 Problèmes polynomiaux

Nous nous intéressons ici à la présentation de trois algorithmes développés pour prouver la polynomialité pour chacun des problèmes suivants : MB-PSDPP, MP-PSDPP et DPP.

Nous considérons tout d'abord le problème MB-PSDPP. Il présente la première variante principale dérivée du problème 3-PSDPP, où seul le nombre de buffers Z doit être minimisé. Une instance de ce problème est définie par $I_{MB} = (\mathscr{X}, \mathscr{Y}, \mathscr{R}_y, \alpha, \beta)$.

Théorème 5.1

Le problème MB-PSDPP peut être résolu en temps polynomial.

Preuve 5.1

Pour prouver la polynomialité de ce problème, nous donnons un algorithme optimal et polynomial, où le nombre de buffers Z^* fourni vaut la valeur de sa borne inférieure lb_Z.

Nous fixons le nombre de buffers Z à la valeur $\max_{y \in \mathscr{Y}} |\mathscr{R}_y|$. Pour chaque calcul, nous préchargeons la liste de ses tuiles d'entrée requises dans ce Z buffers. On itère ce processus pour tous les autres calculs. Dans le planning obtenu, les préchargements et les calculs ne peuvent pas être effectuées simultanément (pas de chevauchement entre calculs et préchargements).

□

La deuxième variante principale dérivée du problème 3-PSDPP est celle de MP-PSDPP, où seul le nombre total de préchargements N doit être minimisé. Une instance de ce problème est définie par $I_{MP} = (\mathscr{X}, \mathscr{Y}, \mathscr{R}_y, \alpha, \beta)$.

Théorème 5.2

Le problème MP-PSDPP peut être résolu en temps polynomial.

Preuve 5.2

Pour prouver la polynomialité de cette variante, nous donnons un algorithme optimal et polynomial, où le nombre de préchargements N^* obtenu vaut la valeur de sa borne inférieure lb_N.

Dans cet algorithme, nous chargeons d'abord successivement l'ensemble des tuiles de $\mathscr{X}'$. Ensuite, lorsque les étapes de préchargements sont terminées, toutes les tuiles de sortie sont calculées successivement.

□

Nous considérons maintenant le problème DPP. Il présente un cas particulier du problème 3-PSDPP, où le nombre de buffers Z et la séquence de calculs $(s_j)_{j \in \mathscr{M}}$ sont fixés comme données d'entrée. L'objectif est de minimiser le nombre total de préchargements N. Une instance de ce problème est donnée par $I_{DPP} = (\mathscr{X}, \mathscr{Y}, \mathscr{R}_y, Z, (s_j)_{j \in \mathscr{M}}, \alpha, \beta)$.

Théorème 5.3

Le problème DPP peut être résolu en temps polynomial.

Preuve 5.3

Afin de prouver la polynomialité de cette variante, nous adaptons l'algorithme KTNS, proposé par Tang et Denardo [110] pour résoudre la variante polynomiale TP du problème ToSP. Nous appelons cette adaptation KAD.

Étant donné la similitude existante entre les deux problèmes DPP et TP (cf. Section 4.3), le principe de l'algorithme KAD est exactement le même que celui de l'algorithme KTNS. En effet, l'idée générale de cette adaptation peut être résumée ainsi.

1. **Traduction des instances** :
   À partir d'une instance du problème TP, nous construisons une instance du problème DPP. En complément de ces données, nous considérons également les deux durées α (pour une étape de préchargement) et β (pour une étape de calcul).
2. **Appliquer l'algorithme KTNS** :
   Cette étape consiste essentiellement en la détermination du nombre total de préchargements N^*.
3. **Déterminer les sorties de DPP** :
   Dans le planning fourni par l'algorithme KTNS, nous déterminons les séquencements des préchargements $(d_i)_{i\in\mathcal{N}}, (b_i)_{i\in\mathcal{N}} et (t_i)_{i\in\mathcal{N}}$, la séquence des dates de début $(u_j)_{j\in\mathcal{M}}$ associé à la séquence des calculs ainsi que la valeur du temps total de traitement Δ.

□

Remarque 4.1. *Une description détaillée de l'algorithme KAD sera décrite dans le Chapitre 5. En outre, cet algorithme est une phase intermédiaire d'autres méthodes de résolution.*

4.5.2 Problèmes NP-Complets

Étant défini un problème de décision P, nous rappelons que pour montrer que ce problème est $\mathcal{NP}$-Complet, il suffit de :

— Montrer qu'il est dans $\mathcal{NP}$;

— Choisir un problème, P_2, $\mathcal{NP}$-Complet approprié (pour un catalogue des problèmes $\mathcal{NP}$-Complets, voir le livre de M.R. Garey et D.S. Johnson [48]) ;

— Construire une *transformation polynomiale* R_p de P_2 vers P et prouver qu'elle est bien une *réduction polynomiale*. En effet, on dit que P_2 se réduit polynomialement à P (et on note $P_2 \propto P$) s'il existe un algorithme de résolution de P_2, qui fait appel à un algorithme de résolution de P, et qui est polynomial lorsque la résolution de P est comptabilisée comme une opération élémentaire.

Dans cette partie, nous prouvons la $\mathcal{NP}$-Complétude pour chacun des problèmes suivants : PSP, MCT-PSDPP et B-C-MCTP.

4.5.2.1 La NP-Complétude du problème PSP

Le problème PSP est parmi les variantes mono-objectif du problème 3-PSDPP, où le nombre de buffers Z est fixé comme une donnée d'entrée et le nombre total de préchargements N doit être minimisé. En effet, une instance de ce problème est définie par $I_{PSP} = (\mathscr{X}, \mathscr{Y}, \mathscr{R}_y, Z, \alpha, \beta)$.

Nous allons montrer que le problème de décision correspondant au problème PSP — défini par *existe-t-il un ordonnancement tel que* $N^* \leq k$ *?*, où k un entier positif — est NP-Complet.

Théorème 5.4

Le problème PSP est $\mathscr{NP}$-Complet.

Preuve 5.4

Notons tout d'abord que PSP est bien dans $\mathscr{NP}$, dû à l'existence d'un algorithme de vérification de solution dont le temps est $O(X * Y * Z)$, où $N^* \leq k$. Nous allons montrer que ce problème est équivalent au problème ToSP, variante $\mathscr{NP}$-Complet, en donnant deux réductions polynomiales :

a) ToSP $\propto$ PSP

b) PSP $\propto$ ToSP

La description du problème ToSP a été détaillée dans la Section 4.2.2. Reprenons alors l'analogie qui a été identifiée entre les deux problèmes, il est facile qu'à partir d'une instance du problème ToSP, notée I_{ToSP}, on peut construire aisément une instance du problème PSP, notée I_{PSP}, et inversement.

En effet, si on sait résoudre l'instance I_{ToSP}, et soit S_{ToSP} la séquence qui minimise le nombre de changements total N_s, alors S_{ToSP} détermine une séquence des calculs pour le problème PSP, une solution du *OUI* pour I_{PSP}, avec un nombre de préchargements N^* égal $N_s + \pi$, où π est le nombre de chargements pour le premier usinage.

Idem, si on sait résoudre l'instance I_{PSP}, et soit S_{PSP} la séquence des calculs qui minimise le nombre total de préchargements N, il est facile de vérifier que S_{PSP} correspond à un séquencement d'usinage, une solution du *OUI* pour I_{ToSP}, avec un nombre de chargements $\hat{N}_s$ égal $N - \pi$, où π est le nombre de préchargements pour le premier calcul.

En d'autres termes, établir une *réduction polynomiale* dans les deux sens nous permet alors de prouver l'équivalence entre ces deux problèmes.

□

Cette preuve d'équivalence permet de conclure que le problème PSP est NP-Complet. Rappelons que le problème DPP est une variante particulière du problème PSP, car la séquence des calculs $(s_j)_{j \in \mathscr{M}}$ est fixée en tant qu'une donnée d'entrée, cette réduction est aussi valide pour prouver la polynomialité du problème DPP qui est équivalent au problème TP.

4.5.2.2 La NP-Complétude du problème MCT-PSDPP

Le problème MCT-PSDPP représente l'un des trois principaux problèmes mono-objectif dérivés du problème 3-PSDPP, où l'on cherche à minimiser uniquement le temps total de traitement Δ.

Considérons une instance I_{MCT}, où $I_{MCT} = (\mathscr{X}, \mathscr{Y}, \mathscr{R}_y, \alpha, \beta)$, le problème de décision correspondant est défini par *existe-t-il un ordonnancement tel que* $\Delta^* \leq q$ *?*, où q est un entier positif.

La question qui se pose maintenant est : **ce problème de décision est-t-il $\mathcal{NP}$-Complet ou peut-être résolu en temps polynomial ?**

Afin d'analyser la complexité de ce problème, nous nous focalisons ici sur l'étude des différents cas en fonction du choix de deux paramètres α et β, qui représentent les durées nécessaires pour les différentes étapes des calculs et celles des préchargements. Nous considérons alors les deux familles suivantes :

1. **MCT-PSDPP, où $\alpha \geq \beta$:**
   Soit $\epsilon_y, \forall y \in \mathcal{Y}$ le nombre de tuiles d'entrée requises par chaque tuile de sortie. Ce paramètre, qui est donné par le cardinal de l'ensemble $|\mathcal{R}_y|, \forall y \in \mathcal{Y}$, est constant pour toutes les tuiles de sortie $y, \forall y \in \mathcal{Y}$. En effet, de nouvelles classes d'instances, notées ϵ_y-MCT, peuvent être ainsi identifiées. Pour simplifier l'étude de leurs complexités, nous considérons dans cette partie le cas où $\alpha = \beta$ et nous utilisons 1 comme valeur de ce deux paramètres ($\alpha = \beta = 1$).

   — **Si** $\epsilon_y = 1, \forall y \in \mathcal{Y}$:
   Le problème 1-MCT est trivial et solvable en temps polynomial avec un Δ égale à $\alpha + \beta * Y$ qui est une borne inférieure sur le temps total de traitement Δ (cf. Proposition 4.3).

   — **Si** $\epsilon_y = 2, \forall y \in \mathcal{Y}$:
   Les données du problème 2-MCT peuvent être représentées par un graphe non-orienté $G = (V, E)$, où $V = \{x / x \in \mathcal{X}\}$ (les tuiles d'entrée) et $E = \{e_y, y \in \mathcal{Y} / e_y = (\mathcal{R}_y)_{y \in \mathcal{Y}}\}$ (les tuiles de sortie) . On cherche à trouver un ordonnancement avec un temps total de traitement Δ qui atteint la borne inférieure lb_Δ, c'est-à-dire un ordonnancement sans temps morts entre les étapes de calculs. En effet, pour chaque nouvelle tuile calculée, il est obligatoire que ses tuiles requises soient déjà préchargées et présentées dans les buffers à la fin du calcul précédent. Dans ce cas, on cherche à couvrir tous les sommets du graphe G. Ceci revient à trouver un *couplage parfait*[2] dans G. Ce problème est connu et très étudié dans la littérature. En outre, si le graphe G est pondéré (au sens du poids des arêtes, donné dans notre cas par la valeur 1 ou 0), et s'il existe un tel couplage, alors on sait trouver un couplage parfait de poids minimum avec un algorithme d'Edmonds. En revanche, si le graphe G n'admet pas de couplage parfait alors, on cherche un couplage maximum de poids minimum (cf. Chapitre 2-Section 2.4.4). Nous pouvons ainsi conclure que le problème 2-MCT est solvable en temps polynomial.

   — **Si** $\epsilon_y = 3, \forall y \in \mathcal{Y}$:
   On cherche à trouver un ordonnancement avec un temps total de traitement Δ optimal, c'est-à-dire un ordonnancement sans temps morts entre les étapes de calculs. Après plusieurs comparaisons et tentatives de re-modélisation, il a été difficile pour nous de trouver un problème $\mathcal{NP}$-Complet connu et étudié dans la littérature de RO dans le but de ramener notre problème 3-MCT à celui-ci afin de prouver sa $\mathcal{NP}$-Complétude.

   — **Si** $\epsilon_y = X, \forall y \in \mathcal{Y}$, avec X est le nombre de tuiles d'entrée :
   Le problème X-MCT est trivial et solvable en temps polynomial avec un Δ optimal égal à $\alpha * X + \beta * Y$. En effet, pour effectuer le premier calcul, il est nécessaire de précharger les X tuiles d'entrée requises.

2. Couplage parfait : est un couplage (un sous-ensemble M d'arêtes d'un graphe G tel que deux arêtes de M ne soient pas adjacentes) couvrant tous les sommets, c'est-à-dire tout sommet est l'extrémité d'une des arêtes du couplage.

Donc, nous pouvons conclure que pour tout $3 \leq \epsilon_y < X$ la complexité de cette famille d'instances ϵ_y-MCT, où $\alpha = \beta$ reste inconnue. Sur la base du même raisonnement, nous pouvons également montrer que la complexité reste inconnue même dans le cas où $\alpha > \beta$.

2. **MCT-PSDPP, où $\beta > \alpha$:**
   Soit A le nombre maximal de tuiles d'entrée requises et donné par $A = \max_{y \in \mathcal{Y}}(\epsilon_y)$, on a alors

   — **Si** $\beta \geq \alpha A$: le problème est trivial et solvable en temps polynomial avec un Δ égale à $\alpha * \min_{y \in \mathcal{Y}} |\mathcal{R}_y| + \beta * Y$ qui est une borne inférieure sur le temps total de traitement Δ (cf. Proposition 4.3).

   — **Si** $\alpha < \beta < \alpha A$: la complexité de cette famille d'instances reste inconnue.

Vue l'existence de plusieurs cas polynomiaux, la complexité finale du problème général MCT-PSDPP présente un challenge à résoudre. Pour aborder cette question, nous procédons en deux étapes suivantes :

— Premièrement, nous montrons qu'une version généralisée du problème MCT-PSDPP, noté MCT-PSDPP$_1$, est $\mathcal{NP}$-Difficile. Dans cette version, la durée β est unitaire pour toutes les étapes de calculs ($\beta = 1$) et la durée α a une certaine valeur k dans $\mathbb{N}$ pour toutes les étapes de préchargements ($\alpha > 1$).

— Deuxièmement, nous montrons que le problème MCT-PSDPP(avec $\alpha = \beta = 1$ unité de temps) est $\mathcal{NP}$-Difficile avec une réduction de MCT-PSDPP$_1$.

Théorème 5.5

Le problème MCT-PSDPP$_1$ est $\mathcal{NP}$-Difficile.

Preuve 5.5

Le problème MCT-PSDPP$_1$ est dans $\in \mathcal{NP}$ car la donnée d'un ordonnancement des tuiles de sortie est un certificat valide vérifiable aisément en temps polynomial : $O(X * Y)$.

Nous allons maintenant montrer que le problème de décision correspondant à MCT-PSDPP$_1$ est $\mathcal{NP}$-Difficile. On va démontrer ce fait en réduisant le problème $\mathcal{NP}$-Difficile *Weak k-visit* [13] à ce problème. Cette *réduction polynomiale* est notée R_p : *Weak k-visit* $\propto$ MCT-PSDPP$_1$.

Le problème *Weak k-visit* est défini comme suit :

◊ **Données** : un graphe $G = (V, E)$, un entier $k \geq 1$ et un sous-ensemble de k sommets : $\sigma(v_i), \forall i \in \{1, \ldots, k\}$

◊ **Solution** : une permutation de sommets $\sigma : V \rightarrow \{1, \ldots, n\}$, tel que $\forall i, \in \{k, \ldots, |V|\}, |E_i(\sigma)| \geq k(i-k)$, avec $E_i(\sigma)$ est l'ensemble des arêtes induites par les i premiers sommets de la permutation σ.

L'objectif est de trouver un ordre sur les sommets σ, où les premiers sommets sont libres. Ensuite, pour chaque sommet ajouté, cela coûte k arêtes. À chaque étape, il faut dépasser les arêtes du graphe induit. La question est alors : *est-il possible, pour un graphe donné, de trouver un tel ordre ?*

L'étape suivante est de construire la réduction R_p. Étant donné une instance du problème MCT-PSDPP$_1$ (un graphe $G = (V, E)$ et une durée $\alpha = k$), il est possible de montrer,

par contradiction, que si et seulement si $\Delta^* \leq \alpha k + |E|$, il existe une solution pour *Weak k-visit*. □

Finalement, nous pouvons prouver que le problème MCT-PSDPP est $\mathcal{NP}$-Difficile, en construisant une instance, où les sommets (les étapes de préchargements) d'une durée $\alpha = k$ dans la version généralisée MCT-PSDPP$_1$ sont remplacé par k sommets d'une durée $\alpha = 1$. De plus, puisqu'on a $\alpha = k < E|$, alors cette réduction est en $O(|E|^2)$.

4.5.2.3 La NP-Complétude du problème B-C-MCTP

Le problème B-C-MCTP est une variante mono-objectif du problème 3-PSDPP, où le nombre de buffers Z est fixé comme une donnée d'entrée. L'objectif est de trouver un ordonnancement qui minimise le nombre total de traitement Δ. Une instance de ce problème est spécifiée par $I_{BCMCT} = (\mathcal{X}, \mathcal{Y}, \mathcal{R}_y, Z, \alpha, \beta)$.

Pour étudier la complexité de ce problème, nous examinons celle d'un cas particulier, noté B-C-MCTP$_1$, où la durée β est unitaire pour toutes les étapes de calculs ($\beta = 1$) et $\alpha = \beta * Y$. Le problème de décision correspondant est défini par : *Existe-t-il un ordonnancement tel que* $\Delta^* \leq s$ *?*, où s est un entier positif.

Pour ce faire, nous avons besoin d'un résultat préliminaire : Lemme 5.1 ci-après.

Lemma 5.1

Pour toute instance I du problème B-C-MCTP$_1$, où $\beta = 1$ et $\alpha = \beta * Y$, le temps total de traitement obtenu est donné par la formule $\Delta^* = \alpha * N + 1$.

Preuve 5.6

Étant donnée une instance I_1 du problème B-C-MCTP$_1$, où $\beta = 1$ et $\alpha = \beta * Y$, nous observons que le temps nécessaire pour précharger l'ensemble $\mathcal{R}_y$ afin d'effectuer le calcul de la tuile de sortie y est au plus égal à $\alpha * |\mathcal{R}_y|$. En effet, cette valeur est plus grand que la durée β nécessaire pour effectuer un seul calcul. Or, on sait que les préchargements peuvent être effectués en parallèle des calculs. Cela signifie que pour effectuer les Y calculs, on a alors besoin de $(Y-1) * (\alpha * |\mathcal{R}_y|) + \beta$ (une valeur supérieure sur le temps total de traitement).

En outre, une tuile d'entrée peut être requise par un ou plusieurs calculs. Elle sera ainsi réutilisée sans avoir besoin de la précharger de nouveau. Dans ce cas, le temps total de traitement peut être exprimé par la somme de $\alpha * N + 1$, où N désigne le nombre total de préchargements (une variable à déterminer). □

Le Lemme 5.1 prouve que, avec $\beta = 1$ et $\alpha = \beta * Y$, le temps total de traitement Δ pour le problème B-C-MCTP$_1$ peut être formulé comme $\alpha N + \beta$ (ou également $Y * N + 1$). On peut alors dire que la minimisation de Δ revient à minimiser le nombre total de préchargements N.

Théorème 5.6

Le problème B-C-MCTP$_1$ est $\mathcal{NP}$-Complet pour $Z \geq 2$.

Pour simplifier, nous présentons ici les détails de la preuve de $\mathcal{NP}$-Complétude du problème B-C-MCTP seulement dans le cas où $Z = 2$.

Preuve 5.7

Le problème B-C-MCTP$_1$ est dans $\in \mathcal{NP}$ car la donnée d'un ordonnancement des tuiles de sortie est un certificat valide vérifiable aisément en temps polynomial : O(X ∗ Y ∗ Z).

Nous allons maintenant montrer que le problème de décision correspondant est $\mathcal{NP}$-Complet. On va démontrer ce fait en réduisant le problème $\mathcal{NP}$-Complet Edge Hamiltonien Path (EHP) [20] à ce problème. Cette *réduction polynomiale* est notée R_p : EHP $\propto$ B-C-MCTP$_1$.

Le problème EHP est défini comme suit : soit G = (V,E) un graphe quelconque non-orienté tel que V = $\{v_1, v_2, \ldots, v_n\}$ désigne un ensemble fini des sommets et E = $\{e_1, e_2, \ldots, e_m\}$ est un ensemble d'arêtes. On note H = (E,I) son *Line-graphe* L(G), tels que chaque sommet E de H est une arête de G et chaque paire $\{e, f\}$, avec $e, f \in$ E, est une arête de H si, et seulement si les arêtes e et f partagent un sommet commun dans G. Le problème associé de décision est le suivant : Existe-t-il alors une *chaîne hamiltonienne* dans le Line-graphe H de G ?

L'étape suivante est de construire la réduction R_p. C'est-à-dire, étant donné une instance du problème EHP, notée I_{EHP}, construire en temps polynomial une instance du problème B-C-MCTP$_1$, notée I_{BCMCT1}, de telle sorte que l'existence d'un ordonnancement valide tel que $\Delta^* \leq s$ soit équivalente à l'existence d'une *chaîne hamiltonienne* dans le Line-graphe H de G.

Cette réduction peut être résumée dans les deux étapes suivantes :

1. **Construction d'instances** : $I_{EHP} \rightarrow I_{BCMCT1}$
   Soient le graphe G = (V,E) et son Line-graphe H, nous construisons alors l'instance I_{BCMCT1}, où : chaque tuile d'entrée $x, x \in \{1, \ldots, X\}$ correspond à un sommet $v_x, x \in \{1, \ldots, n\}$ dans V, une tuile de sortie $y, y \in \{1, \ldots, Y\}$ correspond à une arête $e_y, y \in \{1, \ldots, m\}$ dans E, la matrice r_{xy} possède n lignes associées aux sommets de G et m colonnes associées aux arêtes de G, telles que $r_{xy} = 1$ si, et seulement si l'arête $e_y, y \in \{1, \ldots, m\}$ contient le sommet $v_x, x \in \{1, \ldots, n\}$ et $s = \alpha * N + 1$, où N = m+1 et m désigne le nombre d'arêtes dans G ou aussi le nombre des sommets dans H.

2. I_{EHP} **OUI** pour EHP $\Leftrightarrow$ I_{BCMCT1} **OUI** pour B-C-MCTP$_1$
   Supposons maintenant que le problème EHP possède une solution, notée S_{EHP}. La solution S_{EHP} définit une *chaîne hamiltonienne* dans le *Line-graphe* H de G, c'est-à-dire une séquence $v_1, v_2, \ldots, v_k$, telle que chaque paire (v_i, v_{i+1}) est une arête de H, $i = 1, \ldots, k-1$ ($\simeq$ succession d'arêtes e_i, $i = 1, \ldots, m$ de G, où chaque sommet v_i de H correspond à une arête e_i de G).

   Alors, nous devrions montrer que notre problème B-C-MCTP$_1$ possède une solution, notée S_{BCMCT1}. On a à définir maintenant S_{BCMCT1} comme suit : S_{BCMCT1} = $(N, (d_i)_{i\in\mathcal{N}}, (b_i)_{i\in\mathcal{N}}, (t_i)_{i\in\mathcal{N}}, (s_j)_{j\in\mathcal{M}}, (u_j)_{j\in\mathcal{M}}, \Delta)$, où :

   - ⋆ $(s_j)_{j\in\mathcal{M}}$: la séquence de tuiles de sortie à calculer ($\simeq$ succession de m tuiles de sortie $y, y \in \{1, \ldots, m\}$ à calculer).
   - ⋆ N : le nombre total de tuiles d'entrée $x, x \in \{1, \ldots, n\}$ préchargées pour effectuer les calculs de $(s_j)_{j\in\mathcal{M}}$.
   - ⋆ $(d_i)_{i\in\mathcal{N}}$: la séquence de tuiles d'entrée à précharger dans les buffers ($\simeq$ succession de n tuiles d'entrée).

- ⋆ $(b_i)_{i\in\mathcal{N}}$: la séquence des destinations (≃ succession de Z buffers $z, z \in \{1,\ldots,Z\}$).
- ⋆ $(t_i)_{i\in\mathcal{N}}$: la séquence qui définit les dates de début des préchargements (≃ succession de N dates de début $t_i, t_i \in \mathbb{N}^*$).
- ⋆ $(u_j)_{j\in\mathcal{M}}$: la séquence qui définit les dates de début des calculs (≃ succession de m dates de début $u_j, u_j \in \mathbb{N}^*$).
- ⋆ et Δ le temps total de traitement pour effectuer les différentes étapes de préchargements et celles de calculs.

On a chaque tuile de sortie $y, y \in \{1,\ldots,m\}$ de S_{BCMCT1} correspond à une arête dans G (un sommet dans H), nous constatons alors que le nombre de tuiles d'entrée $x, x \in \{1,\ldots,n\}$ préchargées entre deux calculs successifs y_1 et y_2 dans $(s_j)_{j\mathcal{M}}$ (correspondants aux arêtes e_{y_1} et e_{y_2} de G) est égal à 1. Donc, N égal $2+1*(m-1)$. C'est à dire, $N = m+1$. Cela signifie que $\Delta \leq \alpha * N + 1 = s$.

On peut donc conclure que si le problème EHP a une solution, alors notre problème B-C-MCTP$_1$ aussi en possède une.

Réciproquement, supposons que l'on dispose d'une solution S_{BCMCT1} du problème B-C-MCTP$_1$ — i.e. un ordonnancement des calculs $(s_j)_{j\in\mathcal{M}}$ avec $\Delta \leq \alpha * N + 1$, où $N = m+1$ — alors, nous devrions montrer que $(s_j)_{j\in\mathcal{M}}$ détermine une *chaîne hamiltonienne* dans H.

Pour montrer cette implication, on définit par :

- ⋆ e_{y_i}, $i = 1,\ldots,Y$: une succession d'arêtes de G dans H ;
- ⋆ $a_y, y \in \{1,\ldots,Y-1\}$: le nombre de sommets communs entre chaque paire d'arêtes successives e_{y_1} et e_{y_2} (correspondantes aux tuiles de sortie y_1 et y_2 dans $(s_j)_{j\mathcal{M}}$).

Cela signifie que a_y est :

- ⋆ égal 1 si y_1 et y_2 partagent une tuile d'entrée à précharger en commun ;
- ⋆ égal 2 sinon.

Alors, on a N égal $2+1*(Y-1-\epsilon)+2*\epsilon$, tels que :

- ⋆ ϵ : est le nombre de fois où $a_y = 2, \forall y \in \{1,\ldots,Y-1\}$;
- ⋆ $(Y-1-\epsilon)$ le nombre de fois où $a_y = 1, \forall y \in \{1,\ldots,Y-1\}$.

C'est-à-dire, N égale $2+Y-1+\epsilon = Y+1+\epsilon$, où $\epsilon \geq 0$. Or, si on considère $\epsilon = 0$. On déduit alors que $N = Y+1$. Cela signifie que $\Delta = \alpha * N + 1 = \alpha * (Y+1) + 1$.

On peut donc conclure que si notre problème B-C-MCTP$_1$ a une solution, alors le problème EHP aussi en possède une.

Le problème EHP est par conséquent équivalent à notre problème B-C-MCTP$_1$. Nous pouvons donc conclure que le problème B-C-MCTP$_1$ est $\mathcal{NP}$-Complet pour $Z = 2$.

On peut généraliser cette réduction, pour tout nombre de buffers Z supérieur ou égal à 2, en complétant la matrice r_{xy} avec $Z-2$ lignes entièrement constituées de "1". La matrice r_{xy} a maintenant $X+Z-2$ lignes et $r_{xy} = 1$ si, et seulement si $x \geq n+1$. En d'autres

termes, le nombre de tuiles d'entrée requises par chaque tuile de sortie est toujours égal au nombre de buffers Z ($Z \geq 2$), et de même le nombre de préchargements entre deux tuiles de sortie est toujours égal à 1 ou 2.

□

Nous pouvons donc conclure que le problème général B-C-MCTP est $\mathcal{NP}$-Complet.

4.6 Conclusion

Le problème d'origine 3-PSDPP étant multi-objectif, plusieurs variantes mono-objectif ont été considérées. Dans ce chapitre, nous avons d'abord donné un bref aperçu sur l'approche d'optimisation générique proposée par Mancini et al. [90]. Une revue sur l'état de l'art des problèmes voisins de la littérature de la RO est présentée par la suite. Cette étude nous a amené à positionner notre problème d'origine 3-PSDPP. Ensuite, nous avons caractérisé les différentes bornes liées aux trois critères d'optimisation (Z, N, Δ). Enfin, nous avons fait une analyse de la complexité de certaines variantes du problème 3-PSDPP. Pour l'établir, nous avons proposé trois algorithmes polynomiaux afin de prouver la polynomialité de trois variantes d'entre elles. Nous avons également établi la $\mathcal{NP}$-Complétude — en effectuant une réduction polynomiale à partir des problèmes $\mathcal{NP}$-Complets connus dans la littérature de la MMOpt — d'autres variantes.

L'objectif principal de ce chapitre est de donner les clés pour la compréhension des chapitres 5 et 6 présentant nos différentes approches de résolution proposées (exactes et approchées).

Troisième partie

Approches de Résolution et Validation Numérique

« *Il faut tenir à une résolution parce qu'elle est bonne, et non parce qu'on l'a prise.* »

La Rochefoucauld, (1613-1680)

Chapitre 5

Approches de Résolution Proposées

Sommaire

5.1 Introduction

La diversité des questions de décision liées à la gestion des préchargements (Tuiles, Buffers) pour le calcul d'un noyau non-linéaire met en évidence la particularité du problème d'origine 3-PSDPP que nous avons présenté dans le Chapitre 3. Comme il est difficile de résoudre le problème globalement, nous nous focaliserons dans ce chapitre sur la résolution, exacte et approchée, de certaines variantes dérivées (mono- et bi-objectif).

Nous décrivons dans un premier temps les principes généraux pour aborder la phase de résolution qui fait l'objet essentiel de ce cinquième chapitre. Nous présentons ensuite le contexte d'optimisation qui décrit la liste des paramètres complémentaires au modèle d'origine afin de les prendre en considération. Puis, nous détaillons la phase de résolution exacte par la PLNE pour les trois sous-problèmes mono-objectif $\mathscr{NP}$-Complets : PSP, MCT-PSDPP et B-C-MCTP. Enfin, nous abordons dans la dernière partie la résolution approchée du problème d'origine 3-PSDPP ainsi que certaines de ses variantes mono- et bi-objectif. En effet, nous proposons plusieurs méthodes de type *heuristique* pour apporter des solutions de bonne qualité dans un temps raisonnable.

5.2 Schéma général de résolution

Comme nous l'avons signalé dans le Chapitre 3, nous sommes face à un problème d'optimisation multi-objectif, 3-PSDPP, avec trois objectifs contradictoires tels que le faible coût, la faible consommation d'énergie et la haute performance. Il est alors difficile de trouver une solution unique et optimale pour le résoudre. Par conséquent, une des grandes questions liées à sa résolution est : existe-il des algorithmes (exacts ou approchés) en temps polynomial pour trouver de bonnes solutions à ses variantes dérivées ?

Ce cinquième chapitre apporte des réponses à cette question. Nous nous concentrons alors, tout au long de ce chapitre, sur le développement de diverses méthodes pour résoudre certains problèmes (mono- et bi-objectif) ainsi que le problème d'origine 3-PSDPP. Prenant en compte la grande taille d'instances propre à l'application MMOpt ainsi que la vitesse de résolution (temps d'exécution), ces approches nous permettent de fournir des solutions utiles à l'utilisateur de l'outil MMOpt. Il est bien évident qu'une telle solution sera de bonne qualité.

D'une manière générale, résoudre un problème d'optimisation peut se faire de deux façons :

1. **Méthodes exactes** : dans ce cas, on cherche une meilleure solution possible et on prouve qu'elle est optimale. Cependant, l'utilisation de méthodes exactes n'est pas toujours possible pour un problème donné, par exemple à cause du temps de calcul trop important si le problème est compliqué à résoudre, c'est-à-dire dans le cas où les instances sont de grandes tailles.

2. **Méthodes approchées** : dans ce cas, on cherche à obtenir une "bonne" solution, de valeur proche de la solution optimale en un temps polynomial, sans aucune garantie qu'elle soit la meilleure.

Dans le contexte du problème 3-PSDPP, plusieurs variantes mono-objectif dérivées sont $\mathscr{NP}$-Complètes (cf. Chapitre 4-Section 4.5.2). Nous nous focalisons alors sur leur résolution en utilisant les deux approches citées ci-avant.

5.3 Contexte d'optimisation pour l'application MMOpt

L'objectif de cette section est d'illustrer en termes concrets les différents éléments complémentaires au modèle du problème d'origine 3-PSDPP (cf. Chapitre 3-Section 3.4). L'ensemble de ces paramètres peut être résumé ainsi :

— **La taille d'une instance** : dans le contexte de l'outil MMOpt, les images d'entrée et de sortie traitées contiennent jusqu'à quelques milliers de tuiles d'entrée ainsi que de sortie. En effet, la recherche des méthodes d'optimisation performantes est alors essentielle.

— **La quantité de mémoire interne** : cette métrique est un critère d'optimisation essentiel mais il est difficilement modélisable. Il est alors nécessaire de garder un bon contrôle sur ce paramètre. En effet, face à la problématique du "Memory Wall", la quantité de buffers Z, considérée par le concepteur de l'outil MMOpt, doit être limitée par rapport au nombre de tuiles de l'image d'entrée à précharger. En d'autres termes, dans le contexte d'optimisation du fonctionnement de MMOpt, les solutions fournies pour sa résolution ne seront pas utilisables que si la valeur de Z est assez petite : $Z \leq \frac{1}{6}X$.

— **Le temps disponible pour la phase d'optimisation** : face à un problème d'optimisation multi-objectif, avec trois critères contradictoires, l'utilisateur de l'outil MMOpt cherche un algorithme suffisamment rapide et efficace pour lui fournir une solution optimisant à la fois les trois critères.

  Par conséquent, il est intéressant de poser la question sur le fait d'avoir des approches de résolution performantes tout en garantissant la qualité des solutions fournies, pour que l'utilisateur puisse ainsi choisir sa solution ou décider du meilleur compromis. Autrement dit, le temps d'exécution de chaque module d'optimisation ne doit pas dépasser quelques secondes. En particulier, la résolution du problème 3-PSDPP revient à résoudre le problème bi-objectif 2-PSDPP tout en faisant varier plusieurs fois le nombre de buffers Z.

Les différents paramètres cités ci-avant doivent être pris en considération lorsque nous pensons à la façon d'optimiser le fonctionnement de l'outil MMOpt.

5.4 Résolution exacte

Dans le contexte d'une résolution exacte, nous utilisons la PLNE afin d'obtenir l'optimalité dans le cas où les instances sont de petites tailles. En revanche, la proposition d'une PLNE est clairement hors de portée pour résoudre les instances de grandes tailles fournies par Mancini et al. (2012) [89]. Des techniques visant à réduire ou simplifier l'espace de recherche de solutions peuvent alors êtres associées à ces algorithmes afin de les renforcer. Dans ce chapitre, nous nous intéressons en particulier au développement des algorithmes de type B&B pour les différentes PLNE décrites au Chapitre 3.

Dans cette section, nous considérons les trois sous-problèmes $\mathcal{NP}$-Complets suivants : PSP, MCT-PSDPP et B-C-MCTP. Nous y décrivons en particulier les différentes idées établies pour effectuer des pré et/ou post-traitements afin d'enrichir les PLNE proposées. Nous reviendrons en détails sur la phase d'évaluation numérique dans le Chapitre 6.

Dans le contexte des instances de petite taille, ces formulations de base peuvent être aisément résolues en utilisant un solveur de PLNE tel que Gurobi. Mais la complexité de ces problèmes ne permet pas de résoudre que les grandes instances de manière optimale et dans un délai raisonnable.

Le Tableau 5.1 donne une comparaison des différentes formulations proposées, en termes de variables et de contraintes. Nous rappelons que :

— X est le nombre de tuiles d'entrée ;

— Y est le nombre de tuiles de sortie à calculer. Il désigne également le nombre d'étapes de calculs ;

— M désigne le nombre d'étapes de calculs, où M = Y ;

— N désigne le nombre d'étapes de préchargements ;

— T désigne le nombre d'instants distincts dans le temps pour effectuer toutes les étapes de préchargements et de calculs.

		PSP−PLNE	MCT−PLNE	BCMCTP−PLNE2
No. Variables	**binaires**	$2XY + Y^2$	$X^2 + Y^2$	$YT + 2XT + T$
	entières	0	$2X + Y$	0
	réelles	0	0	0
No. Contraintes		$3Y+Y^2X+XY-X$	$2Y+4X+YX^3-1$	$Y*(1+\lvert\mathscr{R}_y\rvert*\beta+\beta+\alpha)+X*(\alpha+T)+5T$
Contrainte "Big M"		Non	Oui	Non

TABLEAU 5.1 – Comparaison des nos formulations en PLNE

Nous comparons ici le nombre de variables (binaires, entières ou réelles) utilisées ainsi que le nombre de contraintes générées, en particulier l'utilisation de "Big M", par chacune des trois formulations : PSP−PLNE, MCT−PLNE et BCMCTP−PLNE2. L'importance de cette analyse réside dans le fait qu'un nombre de contraintes et de variables réduit permet de résoudre des instances de grande taille.

Comme illustré dans le Tableau 5.1, la formulation PSP−PLNE a le plus petit nombre de variables et de contraintes. En outre, elle ne contient pas de contraintes avec un "Big M". En revanche, la formulation MCTP−PLNE comporte plus de variables binaires et entières ainsi que de contraintes que PSP−PLNE. L'inconvénient le plus important de cette formulation est qu'elle utilise un grand nombre de M, défini par la constante Λ dans la contrainte principale (3.13) du modèle. Finalement, la formulation BCMCTP−PLNE2 a le plus grand nombre de variables, mais il a aussi un avantage remarquable car elle ne contient pas les contraintes de "Big M" qui sont bien connues pour affaiblir la relaxation linéaire et pour diminuer la performance des modèles de PLNE.

D'une manière générale, les différentes formulations proposées sont équivalentes en terme de complexité.

Pour améliorer la performance de ces modèles et minimiser le temps de résolution, nous nous intéressons à enrichir chacun d'entre eux par l'ajout de différentes coupes et/ou des contraintes supplémentaires afin de réduire l'espace de recherche dans le but d'accélérer le processus de résolution. En effet, ces contraintes traduisent respectivement :

— La réduction de taille du problème ;

— Les contraintes de précédence liées à la relation entre les étapes de calculs (tuile de sortie) et celles de préchargements (tuiles d'entrée) ;

— Le nombre limité de buffers.

Nous décrivons maintenant la liste de différentes idées pour des :

1. **Pré-traitements** :

a) `PSP-PLNE` :

Dans le contexte de cette formulation en PLNE, nous cherchons à réduire la taille de l'instance utilisée. Ceci revient à réduire le nombre de tuiles de sortie à calculer. En effet, si la tuile y_2 nécessite le même ou un sous-ensemble de tuiles d'entrée requises par la tuile de sortie y_1, alors calculer y_2 immédiatement après y_1 n'entraîne aucun nouveau préchargement.

Plus formellement, soit $\Omega(y) = \{k : r_{xk} \leq r_{xy}, \forall x \in \mathscr{X}\}$ l'ensemble des tuiles de sortie, où leur calcul immédiatement après la tuile de sortie y pourrait être effectué sans avoir à précharger des tuiles supplémentaires. Il est alors possible de traiter $y \cup \Omega(y)$ comme une seule tuile de sortie à calculer et supprimer l'ensemble $\Omega(y)$ de la formulation, avec une durée de calcul égale à $\beta * (1 + |\Omega(y)|)$.

b) `MCTP-PLNE` :

Dans le contexte de cette formulation en PLNE, nous cherchons aussi à réduire la taille de l'instance utilisée. L'idée ici est inspirée de celle utilisée pour le prétraitement précédent (pour `PSP-PLNE`). En effet, nous réduisons l'ensemble des tuiles de sortie $\mathscr{Y}$ vers un nouvel ensemble, noté $\mathscr{Y}'$ et donné par $\mathscr{Y}' = \mathscr{Y} \setminus \bigcup_{y \in \mathscr{Y}} \Omega(y)$, c'est-à-dire, on a $\mathscr{Y}' \cap \bigcup_{y \in \mathscr{Y}} \Omega(y) = \emptyset$.

Notons maintenant que chaque tuile d'entrée requise est préchargée une et une seule fois dans son propre buffer, c'est-à-dire N = Z, nous cherchons d'abord à déterminer le temps total de traitement, noté $\Delta_{\mathscr{Y}'}$, en résolvant la PLNE `MCTP-PLNE` en utilisant l'ensemble réduit $\mathscr{Y}'$. Ensuite, nous calculons le temps total de traitement Δ en fonction de $\Delta_{\mathscr{Y}'}$ et le temps nécessaire pour calculer toutes les tuiles dans l'ensemble $\bigcup_{y \in \mathscr{Y}} \Omega(y)$. Plus formellement, Δ vaut $\Delta_{\mathscr{Y}'} + \beta * \left| \bigcup_{y \in \mathscr{Y}} \Omega(y) \right|$.

En outre, il est possible d'ajouter une contrainte pour la rupture de symétrie. Notre idée est inspirée du fait qu'on a N = X′ = Z.

2. **Post-traitements** :

Comme nous l'avons signalé dans le Chapitre 3, la formulation `PSP-PLNE` est seulement utilisée pour déterminer le nombre de préchargements N optimal. Or, dans le modèle initial du problème PSP, la gestion des buffers ainsi que des dates de début pour les différentes phases des préchargements et celles des calculs doit être prise en considération lors de la détermination des différentes variables de sortie. Il est alors nécessaire d'ajouter une routine qui détermine les séquences associées. En effet, nous construisons la matrice d'incidence Tuiles d'entrée/Étapes de calculs X × M (matrice booléenne 0 − 1), notée $r_{xj}, \forall x \in \mathscr{X}$ et $j \in \mathscr{M}$, où les tuiles d'entrée présentent les X lignes et les étapes de calculs définissent les Y colonnes.

À partir de cette matrice, nous déterminons la liste des blocs de "1", où un bloc de "1" est défini comme une succession de "1" sur une même ligne encadrée par un "0", à gauche (en colonne j par exemple) et à droite (colonne $j + k$). Concrètement, il s'agit d'une tuile d'entrée qui est préchargée dans le buffer comme une tuile requise pour le premier calcul, puis elle est réutilisée de nouveau par les calculs suivants ou bien si, tout simplement, elle n'est pas écrasée du buffer par un nouveau préchargement.

Une fois la liste des blocs de "1" est déterminée, nous pouvons ainsi définir la séquence de destinations ainsi que celle de dates de début et de fin pour les différentes étapes des préchargements et des calculs (pas de recouvrement entre les calculs et les préchargements).

L'autre manière est d'ajouter les variables binaires décrivant les différentes données de sortie manquantes qui permettent la gestion des buffers ainsi que celle des dates de début et fin des différentes étapes de calculs et idem de celles des préchargements).

Idem pour la formulation `BCMCTP-PLNE`, nous réutilisons le même principe en construisant cette fois-ci la matrice d'incidence Tuiles d'entrée/Instances de préchargements X × T (matrice booléenne 0 − 1), notée $r_{xt}, \forall x \in \mathscr{X}$ et $t \in \mathscr{T}$, où les tuiles d'entrée présentent les X lignes et les instants de préchargements définissent les Y colonnes. En outre, l'idée d'ajouter les variables binaires nécessaires pour les sorties manquantes peut aussi être appliquée dans le contexte de formulation `BCMCTP-PLNE`.

Dans cette thèse, la majorité des problèmes abordés sont $\mathscr{NP}$-Complets et les instances considérées sont de grande taille. Il est alors intéressant de noter que bien que la PLNE soit extrêmement simple à exprimer comme à mettre en œuvre, fournir une solution optimale dans le contexte des grandes instances est par contre assez long et étonnamment complexe. Cette réflexion nous a conduit à rechercher une autre méthode de résolution souple et efficace pour trouver, en un temps raisonnable, des solutions pour les variantes dérivées de notre problème d'origine 3-PSDPP qui soient de très "bonne qualité", même pour des instances de grande taille.

5.5 Résolution approchée

Dans cette section, nous décrivons les différentes méthodes que nous avons élaborées pour résoudre de manière approchée certaines variantes mono- et bi-objectif ainsi que le problème d'origine 3-PSDPP.

5.5.1 Catégories des heuristiques

Dans le cas où on cherche des solutions approchées, les méthodes développées peuvent être réparties en trois familles comme suit :

1. **Heuristiques constructives** : sont des méthodes par construction progressive où à chaque étape, une solution partielle est complétée. Globalement, la plupart de ces méthodes sont des *algorithmes gloutons*, où leur principe est de faire une succession de choix optimaux localement. Ce principe doit être adapté en fonction de la structure du problème posé. En outre, ces algorithmes permettent de fournir rapidement des solutions initiales et de "bonne" qualité (comme nous le verrons dans le Chapitre 6).

   Dans ce contexte, nous proposons deux algorithmes dénommés KAP et SPbP.

2. **Heuristiques basées sur la stratégie de "Partitionnement"** : ces méthodes ont été proposées dans le contexte où on minimise uniquement le temps total de traitement Δ. Leur principe de base consiste à établir dans un premier temps un partitionnement réalisable, un regroupement des tuiles de l'image de sortie à calculer en groupe de tuiles homogènes selon une liste de priorité en fonction de plusieurs critères, tels que le nombre de tuiles requises, le nombre de tuiles partagées en commun entre un ou plusieurs calculs et la quantité de buffers. Dans un second temps, on détermine la séquence de calculs finale.

   À ce titre, nous proposons les deux algorithmes dénommés CGM et CCM.

3. **Heuristiques basées sur la stratégie du "Meilleur Préchargement"** : nous l'appelons aussi "Best Tile". Dans cette méthode, on prend en compte la fréquence d'utilisation des tuiles d'entrée. Autrement dit, on calcule pour chaque tuile d'entrée requise son nombre d'occurrences dans la configuration Tuiles d'entrée/Tuiles de sortie $\mathscr{R}_y, \forall y \in \mathscr{Y}$.

   Dans ce contexte, nous proposons l'algorithme ECM.

En outre, pour résoudre le problème d'origine 3-PSDPP, nous proposons une liste de quatre méthodes heuristiques issues d'une amélioration des méthodes de résolution proposées dans le contexte de ses variantes mono-objectif $\mathcal{NP}$-Complètes.

5.5.2 Résolution de variantes mono-objectif

Cette section est consacrée à la résolution des sous-problèmes PSP, MCT-PSDPP et B-C-MCTP. Compte tenu de leur complexité $\mathcal{NP}$-Complète et du temps imparti à leur résolution exacte en pratique, les algorithmes proposés par la suite font appel à des méthodes de type celles citées dans la section précédente (cf. Section 5.5.1).

5.5.2.1 Solutions pour minimiser le nombre total de préchargements N

Dans cette partie, nous considérons le problème PSP, où le nombre de buffers Z est fixé comme une donnée d'entrée et seul le nombre total de préchargements N doit être minimisé. Comme nous l'avons signalé dans la Section 4.5.2.1 du Chapitre 4, le problème PSP est $\mathcal{NP}$-Complet. C'est pourquoi, nous proposons une méthode de type *heuristique constructive* pour sa résolution, que nous appelons KTNS Adapted to PSP (KAP).

L'algorithme KAP consiste alors en deux étapes principales décrites comme suit :

a) **Étape 1 - Determine a Computation Sequence** :

Cette étape est celle de l'algorithme M_1 et M_2 (cf. Section 4.2.1 du Chapitre 4). En effet, elle consiste essentiellement en la recherche d'un ordonnancement des tuiles de sortie, i.e. une séquence de calculs $(s_j)_{j\in\mathcal{M}} = \{s_1, s_2, \ldots, s_M\}$.

Pour déterminer cet ordonnancement, nous construisons un graphe complet orienté à Y sommets, noté $\overrightarrow{G}_T = (V_T, A_T, \varphi)$, où V_T est l'ensemble des sommets représentés par les tuiles de sortie $\mathcal{Y}$, et $\varphi(k,l)$ désigne la pondération des arcs (k,l). Cette pondération est donnée par $|\mathcal{R}_l \backslash \mathcal{R}_k|$, où l'ensemble $\mathcal{R}_l$ (respectivement $\mathcal{R}_k$) désigne les tuiles d'entrée requises par la tuile de sortie l (respectivement par la tuile de sortie k). Autrement dit, $\varphi(k,l)$ définit le nombre de tuiles à précharger en plus pour l'enchainement du préchargement de la tuile de sortie k à celui de la tuile de sortie l.

Une plus courte *chaîne hamiltonienne* sur le graphe $\overrightarrow{G}_T$, notée C, détermine un ordre du calcul des tuiles de sortie dont le coût total, qui correspond au poids cumulé de tous ses arcs et donné par $\sum_{(k,l)\in C} \varphi(k,l)$, est minimum.

Remarque 5.1. *Plusieurs autres pondérations peuvent être utilisées pour la fonction* φ*, nous citons la liste suivante :*

- $F_1 = \max\{0, |\mathcal{R}_k| + |\mathcal{R}_l| - |\mathcal{R}_k \cap \mathcal{R}_l| - Z\}$ *: cette pondération définit une borne inférieure sur le nombre de préchargements entre deux calculs successifs. Elle a été inspirée de celle proposée par Tang et Denardo (1988) [110] dans le contexte du problème ToSP.*
- $F_2 = \max\{0, |\mathcal{R}_k \cup \mathcal{R}_l| - C\}$ *: cette pondération, utilisée par Crama et al. (1994) [29], définit une autre borne inférieure sur le nombre de préchargements entre chaque couple de calculs.*
- $F_3 = \min_{k\in\mathcal{Y}\backslash\mathcal{Y}'} (|\mathcal{R}_k \backslash \bigcup_{y\in\mathcal{Y}'} \mathcal{R}_y|$ *: cette pondération est la première que nous avons introduite. Elle représente une troisième borne inférieure définissant le nombre de tuiles à précharger en plus pour l'enchainement du préchargement de la tuile de*

sortie k à ceux de toutes les tuiles de sortie l qui précèdent la tuile k dans la séquence $(s_j)_{j\in\mathcal{M}}$. Autrement dit, nous cherchons à tenir compte de la durée de vie de chaque tuile d'entrée déjà préchargée.

b) **Étape 2 - KTNS Adapted to DPP (KAD) Algorithm** :

La séquence des calculs étant fixée (déterminée par l'étape 1 ci-avant), cette étape consiste en l'application de l'algorithme KAD qui a été développé pour établir la polynomialité du problème DPP (cf. Section 4.5.1 du Chapitre 4). Rappelons que le problème DPP est une variante mono-objectif du problème PSP, où la séquence des calculs $(s_j)_{j\in\mathcal{M}}$ est donnée en entrée. En outre, l'algorithme KAD consiste en l'adaptation d'un algorithme connu dans le contexte de la résolution optimale de la variante polynomiale TP du problème ToSP, dénommé KTNS [110].

Dans le contexte du problème DPP, la signification de l'algorithme KTNS est donnée par *Keep Tiles Needed Soonest*. En effet, le principe de l'algorithme KTNS consiste à respecter les deux règles suivantes, pour chaque tuile de sortie dans la séquence des calculs,

i) N'insérer une tuile d'entrée que si elle est requise par la tuile de sortie à calculer.

ii) Si les buffers sont tous préchargés, et qu'une tuile doit être écrasée, choisir la tuile qui sera requise de nouveau au plus tard dans la séquence.

Autrement dit, l'algorithme KTNS prend en compte la durée de vie des tuiles d'entrée dans les buffers.

Comme nous l'avons signalé dans la Section 4.5.1 du Chapitre 4, notre démarche pour effectuer l'adaptation KAD peut être exprimée en trois phases principales ainsi :

1 : Déterminer le nombre total de préchargements $\hat{N}$, ainsi que la séquence des préchargements en appliquant l'algorithme KTNS. Ce nombre est optimal pour la séquence des calculs fixée en entrée ;

2 : Déterminer la séquence des destinations correspondante, c'est-à-dire, spécifier dans quel buffer chaque tuile sera préchargée ;

3 : Déterminer les dates de début pour toutes les étapes de préchargement. Idem pour toutes les étapes de calcul.

Le pseudo-code de l'algorithme KAD peut être illustré comme suit :

Algorithm 5.5.1: KAD$(\mathcal{X},\mathcal{Y},(\mathcal{R}_y)_{y\in\mathcal{Y}},Z,(s_j)_{j\in\mathcal{M}},\alpha,\beta)$

comment: KTNS adapted to DPP problem

$r_{xy} \leftarrow$ INCIDENCE_MATRIX$(\mathcal{X},\mathcal{Y},(\mathcal{R}_y)_{y\in\mathcal{Y}})$
P $\leftarrow$ PERMUTE$(r_{xy},(s_j)_{j\in\mathcal{M}})$
P$' \leftarrow$ FLIP_BLOCKS(P,Z)
N$,(d_i)_{i\in\mathcal{N}} \leftarrow$ PREFETCH_IN_TILE(P$'$)
$(b_i)_{i\in\mathcal{N}} \leftarrow$ PREFETCH_BUFFER$((d_i)_{i\in\mathcal{N}}$,P,Z)
$(t_i)_{i\in\mathcal{N}} \leftarrow$ PREFETCH_STARTDATE$((d_i)_{i\in\mathcal{N}},\alpha,\beta)$
$(u_j)_{j\in\mathcal{M}} \leftarrow$ COMPUTATION_STARTDATE$((s_j)_{j\in\mathcal{M}},(t_i)_{i\in\mathcal{N}},\alpha,\beta)$
$\Delta \leftarrow$ COMPLETION_TIME$((u_j)_{j\in\mathcal{M}},\beta)$
return N$,(p_i)_{i\in\mathcal{N}},(u_j)_{j\in\mathcal{M}},\Delta)$

Nous allons maintenant donner quelques explications sur le pseudo-code donné ci-avant. Comme entrées, l'algorithme KTNS prend une séquence de calculs $(s_j)_{j\in\mathcal{M}}$, un nombre de buffers Z, l'ensemble de tuiles d'entrée à précharger $\mathcal{X}$, l'ensemble de tuiles de sortie à calculer $\mathcal{Y}$, l'ensemble $(\mathcal{R}_y)_{y\in\mathcal{Y}}$ et les deux constantes α et β. Comme une étape clé dans le processus de cet algorithme, nous construisons la matrice d'incidence *Tuiles d'entrée/Tuiles de sortie* r_{xy} associée, dans laquelle l'élément r_{xy} vaut 1 si la tuile d'entrée x est requise pour calculer la tuile de sortie y, et 0 sinon (ligne 1 du pseudo-code). Nous déterminons d'abord la nouvelle matrice d'incidence P en permutant les colonnes de r_{xy} dans leurs ordres dans la séquence $(s_j)_{j\in\mathcal{M}}$ (ligne 2 du pseudo-code).

Dans une seconde étape, nous déterminons l'ensemble des blocs de "0" de la matrice P, où un bloc de "0" est défini comme une succession de "0" sur une même ligne encadrée par un "1", à gauche (en colonne $y-1$ par exemple) et à droite (colonne $y+k+1$). Un bloc de "0" est alors défini par un ensemble sous la forme $\{(x,y),(x,y+1),\ldots,(x,y+k)\}$, tels que :

- ⋆ $0 < y \leq y+k < \mathrm{Y}$
- ⋆ $\mathrm{P}_{x,y} = \mathrm{P}_{x,y+1} = \cdots = \mathrm{P}_{x,y+k} = 0$
- ⋆ $\mathrm{P}_{x,y-1} = \mathrm{P}_{x,y+k+1} = 1$

Concrètement, il s'agit d'une tuile d'entrée qui est préchargée dans le buffer comme une tuile requise, puis qui n'a plus été utilisée par les tuiles de sortie suivantes, avant d'être de nouveau nécessaire au calcul d'une tuile de sortie, comme le montre l'exemple de la Figure 5.1.

Pour un bloc de "0" donné, si les colonnes incidentes à ce bloc ne contiennent pas déjà Z "1" — c'est-à-dire, pour chaque colonne y d'entre elles, on calcule le vecteur de buffers résiduels V_y en utilisant la formule $\mathrm{V}_y = \mathrm{Z} - \sum_{x\in\mathcal{X}} r_{xy}$ — l'ensemble de ce bloc peut alors être "basculé à 1" en gardant la solution optimale. Cela signifie que chaque "0" du bloc passe à "1", donc la tuile d'entrée correspondante à la ligne où se situe le bloc est maintenu dans le buffer interne durant le calcul des k tuiles de sortie.

Pour décider du choix de ces blocs, ces derniers doivent être ordonnés par extrémité terminale croissante, c'est-à-dire l'ordre croissant de l'indice de leur dernière colonne dans la matrice P.

À partir de ces données, une nouvelle matrice P′ est obtenue (ligne 3 du pseudo-code). Autrement dit, l'algorithme KTNS choisit quelles tuiles d'entrée à maintenir dans les buffers entre deux phases d'utilisation, en basculant certains blocs de "0" à "1", comme le montre l'exemple donné par la Figure 5.2, où Z = 2 buffers.

$$\begin{pmatrix} 0 & 1 & \boxed{0} & 1 & \boxed{0} & 1 & 1 \\ 1 & \boxed{0} & \boxed{0} & \boxed{0} & \boxed{0} & \boxed{0} & 1 \\ 0 & 0 & 1 & \boxed{0} & 1 & 0 & 0 \end{pmatrix}$$

FIGURE 5.1 – KTNS - Matrice initiale

$$\begin{pmatrix} 0 & 1 & \boxed{1} & 1 & \boxed{1} & 1 & 1 \\ 1 & \boxed{0} & \boxed{0} & \boxed{0} & \boxed{0} & \boxed{0} & 1 \\ 0 & 0 & 1 & \boxed{1} & 1 & 0 & 0 \end{pmatrix}$$

FIGURE 5.2 – KTNS - Matrice Finale

Ensuite, à partir de la matrice P′, l'algorithme KTNS détermine la séquence de préchargements $(d_i)_{i\in\mathcal{N}}$ associée (ligne 4 du pseudo-code). Pour construire cette séquence, nous affectons, pour chaque étape de calcul d'une tuile de sortie (colonne j dans la matrice P′), les tuiles d'entrée correspondantes aux lignes de P′ dans lesquelles un bloc de "1" consécutifs commence à cette colonne.

Nous déterminons par la suite la séquence des destinations $(d_i)_{i\in\mathcal{N}}$ associée, en affectant pour chaque tuile d'entrée préchargée un buffer libre parmi les Z buffers disponibles.

Nous construisons maintenant un ordonnancement sans recouvrement entre les calculs et les préchargements (lignes 5–8 du pseudo-code). Pour chaque tuile de sortie, on ordonnance les préchargements les uns à la suite des autres. Idem pour les calculs. Nous rappelons que chaque étape du calcul ne commence qu'à la date où ses différentes tuiles d'entrée requises sont préchargées. Nous déterminons ainsi les dates de début des préchargements (routine `Prefetche_StartDate()`) et celles des calculs (routine `Computation_StartDate()`). La routine `Completion_Time()` calcule le temps total de traitement Δ, où $\Delta = u_M + \beta$ et u_M est la date de début du dernier calcul dans la séquence.

Le pseudo-code de l'algorithme KAP peut alors être illustré comme suit :

Algorithm 5.5.2: KAP$(\mathcal{X},\mathcal{Y},(\mathcal{R}_y)_{y\in\mathcal{Y}},Z,\alpha,\beta)$

comment: KTNS adapted to PSP problem

1 : Soit $\overrightarrow{G}=(Y,A)$ *un graphe complet orienté* à Y *sommets*
2 : Soit φ : $A \rightarrow \mathbb{N}$, $\overrightarrow{kl} \mapsto |\mathcal{R}_l \backslash \mathcal{R}_k|$
3 : $(s_j)_{j\in\mathcal{M}} \leftarrow$ SHORT_HAMILTONIAN_CYCLE $(\overrightarrow{G},\varphi)$
4 : $N,(p_i)_{i\in\mathcal{N}},(u_j)_{j\in\mathcal{M}},\Delta \leftarrow$ KAD$(\mathcal{X},\mathcal{Y},(\mathcal{R}_y)_{y\in\mathcal{Y}},Z,(s_j)_{j\in\mathcal{M}},\alpha,\beta)$
return $(N,(p_i)_{i\in\mathcal{N}},(c_j)_{j\in\mathcal{M}},\Delta)$

Remarque 5.2. *L'algorithme KAD sera réutilisé comme une seconde étape de l'algorithme SPbP, proposé pour résoudre le sous-problème bi-objectif 2-PSDPP, comme nous le verrons dans la Section 5.5.3.*

Remarque 5.3. *D'un point de vue algorithmique, la complexité en temps de l'algorithme KAP est au total polynomial, en* $O(X^2Y^2)$*, où chacune de ses étapes l'étant. De même, on peut vérifier que la complexité de l'algorithme SPbP est également en* $O(X^2Y^2)$*, car cet algorithme réutilise KAP comme une sous-routine.*

5.5.2.2 Solutions pour minimiser le temps total de traitement Δ

Dans cette partie, nous nous intéressons à la résolution de deux sous-problèmes mono-objectif dérivés du problème d'origine 3-PSDPP : MCT-PSDPP et B-C-MCTP, où seul le temps total de traitement Δ doit être minimisé. Rappelons que le problème B-C-MCTP est une variante mono-objectif du problème MCT-PSDPP, où le nombre de buffers Z est fixé en entrée.

Dans un premier temps, nous allons présenter deux algorithmes pour le problème MCT-PSDPP. Nous donnons ensuite un algorithme pour résoudre le problème B-C-MCTP.

1. **Les algorithmes ECM et CGM pour le problème MCT-PSDPP** :

   Pour résoudre le problème MCT-PSDPP, nous proposons les deux méthodes heuristiques suivantes : Earliest Computations for MCT (ECM) et Computation Grouping for MCT (CGM).

   Pour ces deux algorithmes, le nombre total de préchargements N est égal à sa borne inférieure lb_N, et le nombre de buffers Z est égal à N. Dans ce cas, Z est égal au nombre total de tuiles d'entrée requises X'.

Une description détaillée ainsi que le pseudo-code de chacune des deux méthodes, citées ci-avant, peuvent être résumés de la manière suivante :

a) **L'algorithme ECM** :

Cette méthode consiste à calculer les tuiles de sortie, l'une après l'autre, au plus tôt possible ("at earliest"), c'est-à-dire, à partir du moment où les tuiles requises pour chaque calcul sont préchargées dans les buffers. Elle autorise également un recouvrement entre les étapes de préchargements et celles de calculs, tout en garantissant qu'il n'y aura aucun chevauchement entre les calculs.

L'algorithme ECM consiste en deux étapes comme suit :

i) **Étape 1 - Schedule Prefetches According to Occurrence Number (lignes 1–5 du pseudo-code)** :

Dans cette phase, nous calculons d'abord le nombre d'occurrences, noté $O_c(x)$, pour chaque tuile d'entrée x dans l'ensemble $\mathscr{R}_y, \forall y \in \mathscr{Y}$. Ensuite, les préchargements $d_i, i \in \mathscr{N}$ seront séquencés dans l'ordre décroissant de leur $O_c(x), \forall x \in \mathscr{X}$, où chaque préchargement sera effectué dans son propre buffer. Le nombre de buffers Z utilisé est alors égal à N.

ii) **Étape 2 - Schedule Computations that at Earliest (lignes 6–7 du pseudo-code)** :

Pour chaque étape de calcul $j, j \in \mathscr{M}$, la date de début $u_j, j \in \mathscr{M}$ correspondante est généralement la date de fin de la dernière tuile d'entrée préchargée parmi l'ensemble de ses tuiles requises. Plus formellement, cette date vaut le $\max(t_i) + \alpha, \forall i \in \mathscr{N} / d_i \in \mathscr{R}_y$, où y est la tuile de sortie calculée à une étape de calcul j. Il est aussi nécessaire de s'assurer qu'il n'y a pas de chevauchement entre les différentes étapes de calcul, c'est-à-dire $u_j - u_{j-1} \geq \beta, \forall j \in \mathscr{M} \setminus \{1\}$.

Par conséquent, pour déterminer la séquence des calculs $s_j, j \in \mathscr{M}$ associée, les tuiles de sortie seront calculées dans l'ordre croissant de leur date de début $u_j, j \in \mathscr{M}$.

Le pseudo-code de l'algorithme ECM est décrit comme suit :

Algorithm 5.5.3: ECM$(\mathscr{X}, \mathscr{Y}, (\mathscr{R}_y)_{y\in\mathscr{Y}}, \alpha, \beta)$

comment: Algorithm ECM for MCT-PSDPP problem

Phase 1 - Schedule Prefetches According to Tile's Occurrence Number

1 : **for** $x \leftarrow 1$ **to** $\mathscr{X}$
do $\{O_c(x) \leftarrow$ COMPUTE_NB_OCCUR_TILE$(\mathscr{X}, (\mathscr{R}_y)_{y\in\mathscr{Y}})$
2 : $N, (d_i)_{i\in\mathscr{N}} \leftarrow$ PREFETCH_IN_TILE$(\mathscr{X}, (\mathscr{R}_y)_{y\in\mathscr{Y}})$
3 : $Z \leftarrow N$
4 : $(b_i)_{i\in\mathscr{N}} \leftarrow$ PREFETCH_BUFFER$((d_i)_{i\in\mathscr{N}}, Z)$
5 : $(t_i)_{i\in\mathscr{N}} \leftarrow$ PREFETCH_START_DATE$((d_i)_{i\in\mathscr{N}}, \alpha)$

Phase 2 - Schedule Computations that at Earliest

6 : $(s_j)_{j\in\mathscr{M}}, (u_j)_{j\in\mathscr{M}} \leftarrow$ COMPUTE_OUT_TILE$(\mathscr{Y}, (\mathscr{R}_y)_{y\in\mathscr{Y}}, (d_i)_{i\in\mathscr{N}}, (t_i)_{i\in\mathscr{N}}, \alpha, \beta)$
7 : $\Delta \leftarrow u_M + \beta$
return $(N, Z, (p_i)_{i\in\mathscr{N}}, (c_j)_{j\in\mathscr{M}}, \Delta)$

b) **L'algorithme CGM** :

Cette méthode consiste en la recherche d'une *Partition*, que l'on note $\mathscr{G}$, de l'ensemble des tuiles de sortie $y, \forall y \in \mathscr{R}_y$ en une liste des "Group".

Pour une tuile de sortie y, un "Group", noté G_y, désigne un sous-ensemble des tuiles de sortie $\{g : g \in \mathscr{Y}\}$, tel que $g \neq y$ et chaque tuile g de ce groupe nécessite les mêmes tuiles d'entrée que celles requises par y (la totalité de l'ensemble $\mathscr{R}_y$) ou également un sous-ensemble de cette configuration : $\mathscr{R}_g \subseteq \mathscr{R}_y$. Dans ce cas, les tuiles de sortie dans le groupe G_y seront immédiatement calculées, l'une après l'autre, après la tuile y sans avoir besoin de charger à nouveau ses tuiles d'entrée requises.

Plus formellement, considérons une tuile de sortie $y, y \in \mathscr{Y}$, G_y est dit un "Group" si et seulement si on a $G_y = \{g : g \in \mathscr{Y}, g \neq y,\ et\ \mathscr{R}_g \subseteq \mathscr{R}_y\}$.

Dans le contexte de cet algorithme, un "Group" G pour une tuile de sortie y est codé sous la forme d'un dictionnaire : G = {clé : valeur}, où la "clé" désigne la tuile y et la "valeur" désigne l'ensemble G_y.

L'algorithme CGM consiste en deux étapes comme suit :

i) **Étape 1 - Determine a Computation Sequence Using "Groups" (ligne 1 du pseudo-code)** :

Cette étape consiste en la détermination d'une séquence de calculs initiale, qu'on note s_{j_0}, où $s_{j_0} \subseteq \mathscr{Y}$ (également désigne une *partition* $\mathscr{G}$). En effet, cette séquence regroupe uniquement l'ensemble de *clés* spécifiques aux différents groupes cibles (dictionnaires).

Nous notons d'abord que les tuiles de sortie sont ordonnées dans l'ordre décroissant en fonction du cardinal de leur configuration $\mathscr{R}_y, \forall y \in \mathscr{Y}$. Nous construisons ensuite la séquence s_{j_0} de la manière suivante :
Dans un premier temps, après avoir déterminé un "Group" G associé à une tuile de sortie y, l'ensemble $\mathscr{Y}$ est mis à jour, tout en supprimant les tuiles de sortie appartenant au groupe G (la clé y et la valeur G_y). Nous déterminons ensuite un nouveau groupe de type G en utilisant l'ensemble $\mathscr{Y}$ mis à jour. Cette phase se répète jusqu'à ce que l'ensemble $\mathscr{Y}$ soit vide.

ii) **Étape 2 - Schedule Prefetches and Computations (lignes 2–7 du pseudo-code)** :

Nous notons d'abord que les tuiles de sortie dans la séquence s_{j_0}, déterminée à la première étape, seront calculées dans l'ordre croissant en fonction du cardinal de leur configuration $|\mathscr{R}_y|, \forall y \in s_{j_0}$. Nous déterminons ensuite la séquence des préchargements $(d_i)_{i \in \mathscr{N}}$ associée à la séquence des calculs s_{j_0}, où chaque tuile d'entrée requise est préchargée une et une seule fois dans son propre buffer. Cela signifie que le nombre total de préchargements obtenu vaut X′, ainsi que le nombre de buffers utilisés vaut N. On a alors N = Z = X′.

En outre, pour déterminer la séquence des destinations $(b_i)_{i \in \mathscr{N}}$ associée, il suffit d'affecter chaque buffer à une étape unique de préchargement.

Puis, pour construire la séquence finale $(s_j)_{j \in \mathscr{M}}$, nous calculons, à la suite de chaque tuile de sortie y dans la liste initiale des calculs s_{j_0}, l'ensemble de tuiles de sortie dans G_y.

Enfin, pour déterminer les deux séquences des dates de début $(t_i)_{i \in \mathscr{N}}$ et $(u_j)_{j \in \mathscr{M}}$, les préchargements s'effectuent en parallèle des calculs. Le temps total de traitement Δ est alors égal à la date de fin du dernier calcul : $\Delta = u_M + \beta$.

Le pseudo-code de l'algorithme CGM est donné comme suit :

Algorithm 5.5.4: CGM($\mathscr{X}, \mathscr{Y}, (\mathscr{R}_y)_{y\in\mathscr{Y}}, \alpha, \beta$)

comment: Algorithm CGM for MCT-PSDPP problem

Phase 1 - Determine a Computation Sequence Using "Groups"

1 : **for** $y \leftarrow 1$ **to** $\mathscr{Y}$

do $\begin{cases} G_y \leftarrow \{g : g \in \mathscr{Y}, g \neq y, \ \& \ \mathscr{R}_g \subseteq \mathscr{R}_y\} \\ s_{j_0} \leftarrow y \\ \mathscr{Y} \leftarrow Y \backslash \{y \cup G_y\} \quad \text{\# MAJ de l'ensemble } \mathscr{Y} \end{cases}$

Phase 2 - Schedule Prefetches and Computations

2 : $N, (d_i)_{i\in\mathscr{N}} \leftarrow$ PREFETCH_IN_TILE($\mathscr{X}, (\mathscr{R}_y)_{y\in\mathscr{Y}}$)

3 : $Z \leftarrow N$

4 : $(b_i)_{i\in\mathscr{N}} \leftarrow$ PREFETCH_BUFFER($(d_i)_{i\in\mathscr{N}}, Z$)

5 : $(s_j)_{j\in\mathscr{M}} \leftarrow$ COMPUTE_OUT_TILE($\mathscr{Y}, (\mathscr{R}_y)_{y\in\mathscr{Y}}, (d_i)_{i\in\mathscr{N}}, (t_i)_{i\in\mathscr{N}}, \alpha, \beta, s_{j0}$)

6 : $(t_i)_{i\in\mathscr{N}}, (u_j)_{j\in\mathscr{M}} \leftarrow$ START_DATE_PREFETCH_COMPUTE($(d_i)_{i\in\mathscr{N}}, (s_j)_{j\in\mathscr{M}}, \alpha, \beta$)

7 : $\Delta \leftarrow u_M + \beta$

return $(N, Z, (p_i)_{i\in\mathscr{N}}, (c_j)_{j\in\mathscr{M}}, \Delta)$

2. **l'algorithme CCM pour le problème B-C-MCTP** :

Pour résoudre le problème B-C-MCTP, nous avons proposé la méthode heuristique Computation Classes for MCT (CCM). L'idée principale de cet algorithme est de partitionner l'ensemble des tuiles de sortie $\mathscr{Y}$ en une liste des "Class", que l'on note $\mathscr{C}$. Une "Class", notée C, est un sous-ensemble des tuiles de sortie de l'ensemble principal $\mathscr{Y}$ qui nécessitent au plus Z_0 tuiles d'entrée. Ces tuiles de sortie seront calculées successivement (l'une après l'autre) après avoir préchargé les Z_0 tuiles requises.

Pour déterminer une "Class" C, nous considérons un nombre de buffer Z_0, qui doit être égale ou supérieure à sa borne inférieure lb_Z donnée par $\max_{y\in\mathscr{Y}} |\mathscr{R}_y|$. En effet, cette valeur de Z_0 sera considérée comme donnée d'entrée dans le processus de cet algorithme présenté par l'organigramme de la Figure 5.6 ci-dessous. Plus formellement, C est dit une "Class" si et seulement si $\left|\bigcup_{y\in C} \mathscr{R}_y\right| \leq Z_0$.

L'algorithme CCM consiste en deux étapes comme suit :

a) **Step1 - Determine a Computation Sequence Using "Classes" (lignes 1–2 du pseudo-code)** :

Cette étape consiste en la détermination d'une partition réalisable $\mathscr{C}$ de l'ensemble des tuiles de sortie en une liste de k "Class" $\mathscr{C}$. Plus formellement, la partition $\mathscr{C}$ est définie par : $\mathscr{C} = \{C_k, \forall k \in \mathbb{N}^*\}$. Il est aussi nécessaire de garantir que chaque tuile de sortie appartient à exactement une seule classe C_k.

Supposons maintenant que les tuiles de sortie sont ordonnées dans l'ordre décroissant en fonction du cardinal de leur configuration $\mathscr{R}_y, \forall y \in \mathscr{Y}$. Il s'agit ainsi de choisir quel sous-ensemble peut être considéré comme une "Class" C. Nous utilisons ici comme critère de choix le nombre de tuiles d'entrée partagées entre les tuiles de sortie, c'est-à-dire que pour chaque couple de tuiles de sortie i et j, ce nombre est donné par $|\mathscr{R}_i \cap \mathscr{R}_j|$. Cette phase de regroupement est répétée à chaque fois, sur l'ensemble $\mathscr{Y}$ mis à jour après la détermination d'une "'Class", jusqu'à ce qu'aucun regroupement ne soit possible sans dépasser le nombre de buffers Z_0.

Une fois que la partition $\mathscr{C}$ associée à l'ensemble $\mathscr{Y}$ est déterminée, nous devons maintenant construire la séquence finale des calculs $s_j, j \in \mathscr{M}$ correspondante.

Nous supposons d'abord que les tuiles de sortie appartenant à la même "Class" C_k seront calculées dans l'ordre décroissant en fonction du cardinal de leur configuration, noté par $|\mathscr{R}_y|, \forall y \in C_k, \forall k \in \mathbb{N}^*$ et $\forall C_k \in \mathscr{C}$. Pour déterminer la séquence $s_j, j \in \mathscr{M}$, de nombreuses tentatives ont alors été proposées.

Nous en décrivons ici quelques-unes pour illustrer la variété des stratégies qu'on peut tenter de mettre en œuvre dans le but de trouver un meilleur ordonnancement des calculs. Nous appliquons ainsi l'une de trois stratégies suivantes :

i) **CCM1** :

Nous notons $N_s(C)$ le nombre de tuiles d'entrée partagées ("shared tiles") entre la "Class" C et chaque "Class" $C_k, \forall k \in \mathbb{N}^*$ dans la partition $\mathscr{C} \backslash C$. Nous rappelons également que le nombre de tuiles d'entrée partagées entre deux classes C_1 et C_2, où $C_1, C2 \in \mathscr{C}$, est donné par : $N_s(C_1, C_2) = \left|\mathscr{R}_y(C_1) \cap \mathscr{R}_y(C_2)\right|$, où $\mathscr{R}_y(C_1) = \bigcup_{y \in C_1} \mathscr{R}_y$ et $\mathscr{R}_y(C_2) = \bigcup_{y \in C_2} \mathscr{R}_y$. Le nombre $N_s(C)$ peut ainsi être donné par la formule suivante : $N_s(C) = \sum_{(C_k)_{k \in \mathbb{N}^*} \in \mathscr{C} \backslash C} N_s(C, C_k)$.

Dans cette stratégie, nous calculons d'abord le nombre $N_s((C_k)_{k \in \mathbb{N}^*})$ pour chaque "Class" $C_k, \forall k \in \mathbb{N}^*$ de la partition $\mathscr{C}$. Ensuite, les classes $(C_k)_{k \in \mathbb{N}^*}$ déterminées dans la partition $\mathscr{C}$ seront séquencées dans l'ordre décroissant en fonction de leur $N_s((C_k)_{k \in \mathbb{N}^*})$.

ii) **CCM2** :

Nous notons $N_o((C_k)_{k \in \mathbb{N}^*})$ le nombre total d'occurrence $O_c(x)$ pour chaque "Class" $C_k, \forall k \in \mathbb{N}^*$ de la partition $\mathscr{C}$, où $x \in \mathscr{R}_y(C) = \bigcup_{y \in C} \mathscr{R}_y$ et $O_c(x)$ représente le nombre d'occurrences de la tuile d'entrée x dans $\mathscr{R}_y(C)_{C \in \mathscr{P}}$: $N_o(C) = \sum_{x \in \mathscr{R}_y(C)} O_c(x)$.

Dans cette stratégie, nous calculons d'abord le nombre $N_o(C)$ pour chaque "Class" $C_k, \forall k \in \mathbb{N}^*$ de la partition $\mathscr{C}$. Ensuite, les classes $(C_k)_{k \in \mathbb{N}^*}$ déterminées dans la partition $\mathscr{C}$ seront séquencées dans l'ordre décroissant en fonction de leur $N_o((C_k)_{k \in \mathbb{N}^*})$.

iii) **CCM3** :

Dans cette stratégie, les classes déterminées dans la partition $\mathscr{C}$ seront séquencées dans l'ordre décroissant en fonction du nombre de tuiles de sortie constituant chaque "Class" $C_k, \forall k \in \mathbb{N}^*$: $|C_k|, \forall C_k \in \mathscr{C}$.

b) **Step2 - Schedule Prefetches and Computations (lignes 3–7 du pseudo-code)** :

Une fois la séquence des calculs $(s_j)_{j \in \mathscr{M}}$ est déterminée (cf. Step N° 1), nous cherchons maintenant à déterminer les autres données de sortie, à savoir la séquence des préchargements $(d_i)_{i \in \mathscr{N}}$, la séquence des destinations $(b_i)_{i \in \mathscr{N}}$, les séquences des dates de début associées : $(t_i)_{i \in \mathscr{N}}$et $(u_j)_{j \in \mathscr{M}}$, ainsi que les valeurs des trois critères N, Z et Δ.

Dans cette phase, nous utilisons ainsi le même principe celui de la deuxième étape de l'algorithme CGM, décrit ci-avant (cf. Partie b)- ii)).

Remarque 5.4. *Si la tuile de sortie y nécessite exactement Z_0 tuiles d'entrée, où $|\mathscr{R}_y| = Z_0$, cette tuile construit toute seule une classe de type "Class" C. En revanche, on peut*

uniquement intégrer les tuiles de sortie qui nécessitent un ensemble $(\mathscr{R}_{y'})_{y'\in\mathscr{Y}}$*, tel que* $y' \neq y$ *et* $\mathscr{R}_{y'} \subseteq \mathscr{R}_y$*. Dans ce contexte, cette "Class"* C *peut être considérée comme étant un groupe de type "Group"* G_y*.*

Le pseudo-code de l'algorithme CCM est décrit comme suit :

Algorithm 5.5.5: CCM$(\mathscr{X}, \mathscr{Y}, (\mathscr{R}_y)_{y\in\mathscr{Y}}, Z_0, \alpha, \beta)$

comment: Algorithm CCM for B-C-MCTP problem

Phase 1 - Determine a Computation Sequence Using "Classes"

1 : **for** $y \leftarrow 1$ **to** $\mathscr{Y}$

do
$\quad C \leftarrow \{y : y \in \mathscr{Y} / \left|\bigcup_{y\in C} \mathscr{R}_y\right| \leq Z_0\}$
$\quad s_{j_0} \leftarrow C$
$\quad \mathscr{Y} \leftarrow Y \backslash C$ # MAJ de l'ensemble $\mathscr{Y}$

2 : $(s_j)_{j\in\mathscr{M}} \leftarrow$ CCM1$(\mathscr{Y}, (\mathscr{R}_y)_{y\in\mathscr{Y}}, Z_0)$ #ou CCM2() ou CCM3()

Phase 2 - Schedule Prefetches and Computations

3 : $N, (d_i)_{i\in\mathscr{N}} \leftarrow$ PREFETCH_IN_TILE$(\mathscr{X}, (\mathscr{R}_y)_{y\in\mathscr{Y}})$

4 : $Z \leftarrow N$

5 : $(b_i)_{i\in\mathscr{N}} \leftarrow$ PREFETCH_BUFFER$((d_i)_{i\in\mathscr{N}}, Z)$

6 : $(t_i)_{i\in\mathscr{N}}, (u_j)_{j\in\mathscr{M}} \leftarrow$ START_DATE_PREFETCH_COMPUTE$((d_i)_{i\in\mathscr{N}}, (s_j)_{j\in\mathscr{M}}, \alpha, \beta)$

7 : $\Delta \leftarrow u_M + \beta$

return $(N, Z, (p_i)_{i\in\mathscr{N}}, (c_j)_{j\in\mathscr{M}}, \Delta)$

Remarque 5.5. *La première étape de chacun des deux algorithmes CGM et CCM peut être utilisée comme une première étape pour les deux algorithmes KAP et SPbP afin de déterminer la séquence des calculs.*

Remarque 5.6. *Les trois algorithmes, ECM, CGM et CCM, sont équivalents en terme de complexité, qui est en* O(XY).

5.5.3 Résolution du problème bi-objectif 2-PSDPP : l'algorithme SPbP

Nous considérons dans cette section le problème bi-objectif 2-PSDPP, où le nombre de buffers Z est fixé et les deux critères, le nombre total de préchargements N et le temps total de traitement Δ, doivent être minimisés simultanément.

Pour résoudre cette variante, nous avons développé une approche de type heuristique constructive, dénommée Shifted Prefetches for bi-PSDPP (SPbP). L'idée de cet algorithme consiste à réutiliser les deux étapes de l'algorithme KAP pour optimiser en premier temps le nombre total de préchargement N. Dans un second temps, nous l'enrichissons par une troisième étape afin de minimiser le temps total de traitement Δ.

Cet algorithme comporte trois étapes que nous décrivons ci-dessous :

a) **Étape 1 - Determine a Computation Sequence (ligne 1 du pseudo-code)** :
Comme première étape, nous reprenons celle de l'algorithme KAP (également celle de l'algorithme M_1 et M_2), qui résout une instance d'un ATSP, pour déterminer la séquence de calculs $(s_j)_{j\in\mathscr{M}}$ (cf. Section 4.2.1 du Chapitre 4).

b) **Étape 2 - Apply KAD Algorithm (ligne 2 du pseudo-code)** :
Le nombre de buffers Z étant fixé, nous utilisons l'algorithme KAD pour obtenir un nombre N de préchargements optimal, et un ordonnancement des préchargements et des calculs, associés à la séquence de calculs $(s_j)_{j\in\mathscr{M}}$ déterminée à l'étape 1.

c) **Étape 3 - Shifting Prefetches (lignes 3–4 du pseudo-code)** :

Cette étape autorise un recouvrement entre les étapes de préchargements et celles de calculs. Nous cherchons ainsi à décaler ("Shift") dans le planning résultant de l'algorithme KAD les préchargements qui peuvent être menés en parallèle avec les calculs visant la minimisation du temps total de traitement Δ.

En effet, par exemple pour décaler une étape du préchargement k qui a été effectuée entre les deux calculs i et j (i précède j dans la séquence de calculs $(s_j)_{j\in\mathcal{M}}$) en parallèle avec l'étape du calcul i, il est d'abord nécessaire de faire vérifier si aucune tuile d'entrée requise par la tuile de sortie à l'étape de calcul i, n'a été écrasée jusqu'à sa date de fin w_i. En outre, comme il s'agit d'un décalage dans le temps, certaines étapes de calculs peuvent aussi être effectuées en avance, tout en prenant en compte l'état de chaque buffers (libre, utilisé ou préchargé).

Remarque 5.7. *L'idée de l'étape 3 de l'algorithme SPbP fait référence à une première tentative de résolution. Cette dernière consiste à avancer dans le temps, si c'est possible, toutes les étapes de préchargements effectuées entre deux calculs successifs, par exemple i et j (i précède j dans la séquence de calculs $(s_j)_{j\in\mathcal{M}}$), de manière à les paralléliser avec l'étape du calcul i sans avoir à écraser les tuiles déjà présentes dans les buffers pour ce calcul i. Ceci revient à assurer que toutes les étapes de préchargements pouvant être avancées n'utilisent pas les mêmes buffers que ceux utilisés pour précharger les tuiles requises par le calcul i.*

Ces différentes étapes sont décrites par le pseudo-code ci-dessous, où les deux premières lignes représentent les deux étapes de l'algorithme KAP. En revanche, la troisième ligne décrit le principe de "Shift" en utilisant la stratégie de *Décalage unitaire*.

Algorithm 5.5.6: SPBP$(\mathcal{X},\mathcal{Y},(\mathcal{R}_y)_{y\in\mathcal{Y}},Z,\alpha,\beta)$

comment: Algorithm SPbP for 2-PSDPP

Phase 1 - Determine a Computation Sequence

1 : $(s_j)_{j\in\mathcal{M}} \leftarrow$ COMPUTATION_SEQUENCE$(\mathcal{X},\mathcal{Y},(\mathcal{R}_y)_{y\in\mathcal{Y}})$

Phase 2 - Apply KAD Algorithm

2 : $N,(p_i)_{i\in\mathcal{N}},(u_j)_{j\in\mathcal{M}},\Delta_1 \leftarrow$ KAD$(\mathcal{X},\mathcal{Y},(\mathcal{R}_y)_{y\in\mathcal{Y}},Z,(s_j)_{j\in\mathcal{M}},\alpha,\beta)$

Phase 3 - Shifting Prefetches

3 : **for** $j \leftarrow 2$ **to** $\mathcal{M}$

do
- Prefetch[j] $\leftarrow \{i \in \mathcal{N} / P'[j][d_i] - P'[j-1][d_i] = 1\}$
- Buffer[j] $\leftarrow \{b_i / i \in$ Prefetch[j]$\}$
- Advanced_Prefetch[j] $\leftarrow \{l / l \in$ Prefetch[j] & Buffer[j][l] $\notin b_i(\mathcal{R}_{s_{j-1}})\}$
- Not_Advanced_Prefetch[j] $\leftarrow$ Prefetch[j]\Advanced_Prefetch[j]

4 : $(t_i)_{i\in\mathcal{N}},(u_j)_{j\in\mathcal{M}},\Delta_2 \leftarrow$ SCHEDULE_PREFETCHES_COMPUTATIONS$((s_j)_{j\in\mathcal{M}}$, Advanced_Prefetch[], Not_Advanced_Prefetch[], $\alpha,\beta)$

return $(N,(p_i)_{i\in\mathcal{N}},(c_j)_{j\in\mathcal{M}},\Delta)$

Nous allons maintenant donner quelques explications sur la phase de "Shifting Prefetches" (lignes 3–4 du pseudo-code). Nous donnons d'abord la définition de chacune des différentes variables mentionnées à la ligne 3 de la manière suivante, où :

— Prefetch[j] : désigne la liste des tuiles d'entrée qui seront préchargées avant l'étape du calcul j, c'est-à-dire les tuiles d'entrée i correspondantes aux blocs de "1" commençant à la colonne j dans la matrice P′ (cf. Pseudo-code et explication de l'algorithme *KAD* présentés à la Section 5.5.2.1) ;

— Buffer[j] : désigne la liste des buffers affectés aux préchargements dans Prefetch[j] ;

— Advanced_Prefetch[j] : désigne la liste des préchargements pouvant être avancés ;

— Not_Advanced_Prefetch[j] : désigne la liste des préchargements stables (ne pouvant pas être avancés) ;

Ensuite, pour chaque étape du calcul $j, j \in \mathcal{M}$, nous déterminons d'abord les listes Prefetch[j] et Buffer[j]. Enfin, pour choisir quelle tuile de Prefetch[j] peut être préchargée en parallèle du calcul j, il suffit de vérifier que sa destination est différente de tous les buffers utilisés par l'ensemble des tuiles requises par j et déjà préchargées dans des étapes précédentes.

L'organigramme de la Figure 5.3 présente les différentes étapes des deux algorithmes KAP et SPbP. Comme le montre cette figure, l'algorithme KAP représente les deux premières étapes de l'algorithme SPbP.

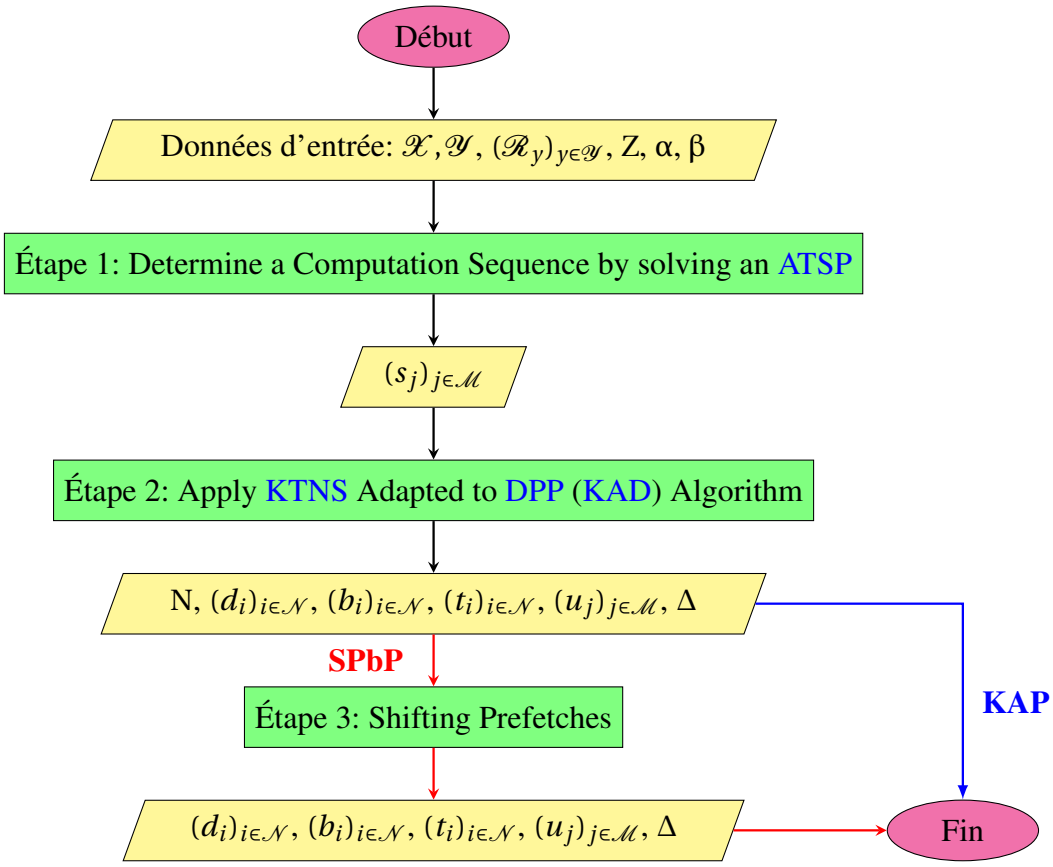


FIGURE 5.3 – Organigramme pour les algorithmes KAP et SPbP

5.5.4 Résolution du problème d'origine 3-PSDPP

Le problème d'origine abordé dans cette thèse étant multi-objectif, avec trois objectifs contradictoires tels que le nombre de buffers Z (le faible coût), le nombre total de préchargements N (la faible consommation d'énergie) et le temps total de traitement Δ (la haute performance). Dans ce contexte, il est impossible de parler de solution optimale ou unique. On

cherche alors à identifier un ensemble de solutions de compromis entre les trois objectifs. Nous proposons alors une liste de quatre méthodes heuristiques. Ces dernières représentent quatre extensions de certaines de nos heuristiques proposées ci-avant, que nous appelons, respectivement, E-SPbP, E-ECM, E-CGM et E-CCM.

L'idée principale de chacune d'entre elles peut être résumée ci-après. Nous distinguons alors les deux familles suivantes :

1. **L'algorithme E-SPbP** :

   Comme nous l'avons vu dans la Section 5.5.3, l'algorithme SPbP fournit des solutions, en termes de N et Δ, pour le problème 2-PSDPP, où Z est fixé en entrée, nous pouvons alors le réutiliser comme une méthode de résolution efficace pour produire également des solutions au problème d'origine 3-PSDPP.

   Chaque exécution de l'algorithme SPbP, avec un nombre de buffers Z variant dans l'intervalle $I = \{lb_Z, \ldots, uv_Z\}$, où lb_Z est la borne inférieure sur le nombre de buffers Z et uv_Z présente une grande constante pour Z ("upper value") qui peut être définie par la constante $\sum_{y \in \mathscr{Y}} |\mathscr{R}_y|$ ou encore par le nombre de tuiles d'entrée utilisées X' ($uv_Z = X'$), donne une solution au problème 3-PSDPP.

   L'utilisateur de l'outil MMOpt peut alors choisir sa solution de compromis préférée, parmi l'ensemble des solutions fournies.

2. **Les algorithmes : E-ECM, E-CGM et E-CCM** :

   Pour chacun des trois algorithmes (ECM, CGM et CCM), qui ont été proposés pour résoudre les deux problèmes MCT-PSDPP et B-C-MCTP (cf. la Section 5.5.2.2), nous considérons trois versions étendues pour résoudre le problème d'origine 3-PSDPP. Ces algorithmes sont : E-ECM, E-CGM et E-CCM. Pour chacun d'entre eux, nous ajoutons une troisième étape, que nous appelons *Reduce Buffers Usage*, afin de minimiser le nombre de buffers Z.

   Le principe de cette étape fait référence à la notion de blocs de "1" (voir Section 5.5.3 : étape 3). En effet, le nombre maximal de blocs de "1" contenus dans la matrice d'incidence Tuiles d'entrée/Tuiles de sortie — la matrice r_{xy} permutée en fonction de la séquence $(s_j)_{j \in \mathscr{M}}$ déterminée à la première étape de chacun des trois algorithmes ECM, CGM et CCM — constitue le nombre de buffers nécessaires pour effectuer tous les préchargements.

   Dans cette étape, nous réutilisons tout simplement certains buffers lorsque la tuile d'entrée préchargée correspondante ne sera plus utilisée après sa dernière étape de calcul associée.

Nous donnons maintenant un ensemble de trois organigrammes — représentés dans les Figures 5.4, 5.5 et 5.6 — qui définit la liste des étapes partagées ou combinées, entre les différents algorithmes proposés (Section 5.5.2.2 et 5.5.4).

L'organigramme de la Figure 5.4 présente les étapes de deux algorithmes ECM et E-ECM. De la même manière, l'organigramme de la Figure 5.5 illustre celles de deux algorithmes CGM et E-CGM. Comme le montrent ces deux figures, l'algorithme ECM représente les deux premières étapes de l'algorithme E-ECM et de même, l'algorithme CGM représente les deux premières étapes de l'algorithme E-CGM.

L'organigramme de la Figure 5.6 illustre les étapes principales de l'algorithme CCM. Comme le montre cette figure, l'algorithme CCM représente les deux premières étapes de l'algorithme E-CCM.

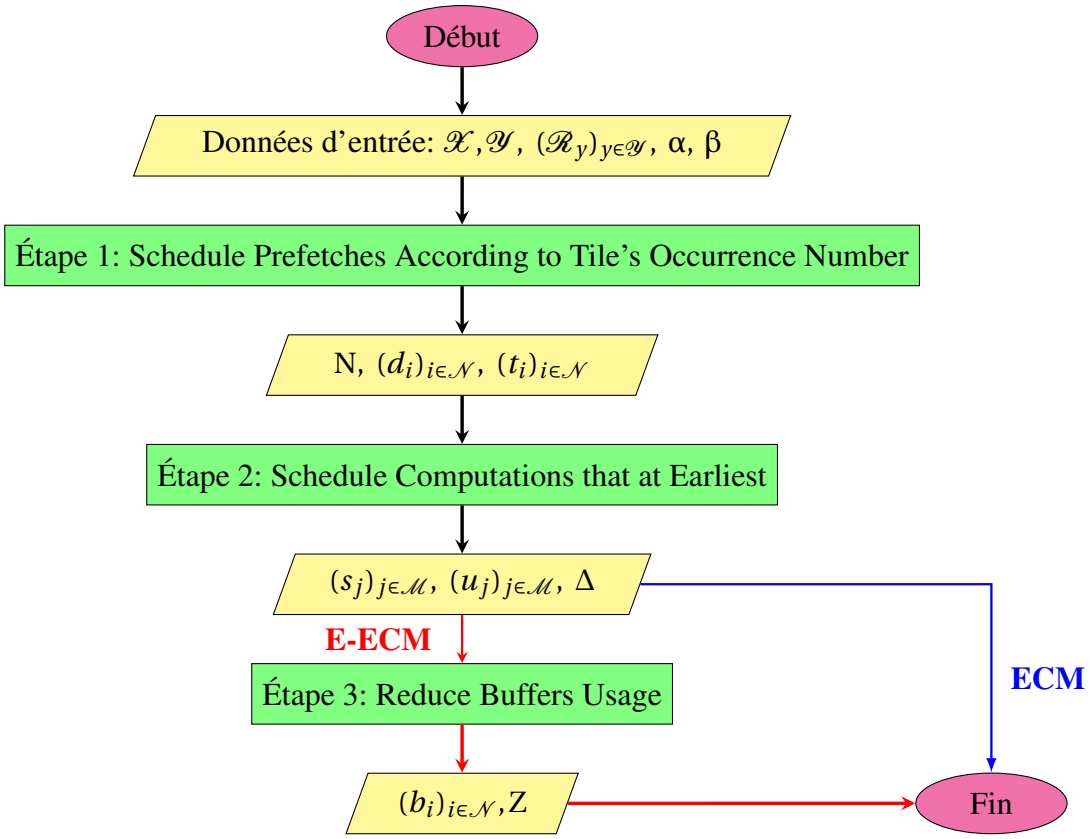


FIGURE 5.4 – Organigramme pour les algorithmes ECM et E-ECM

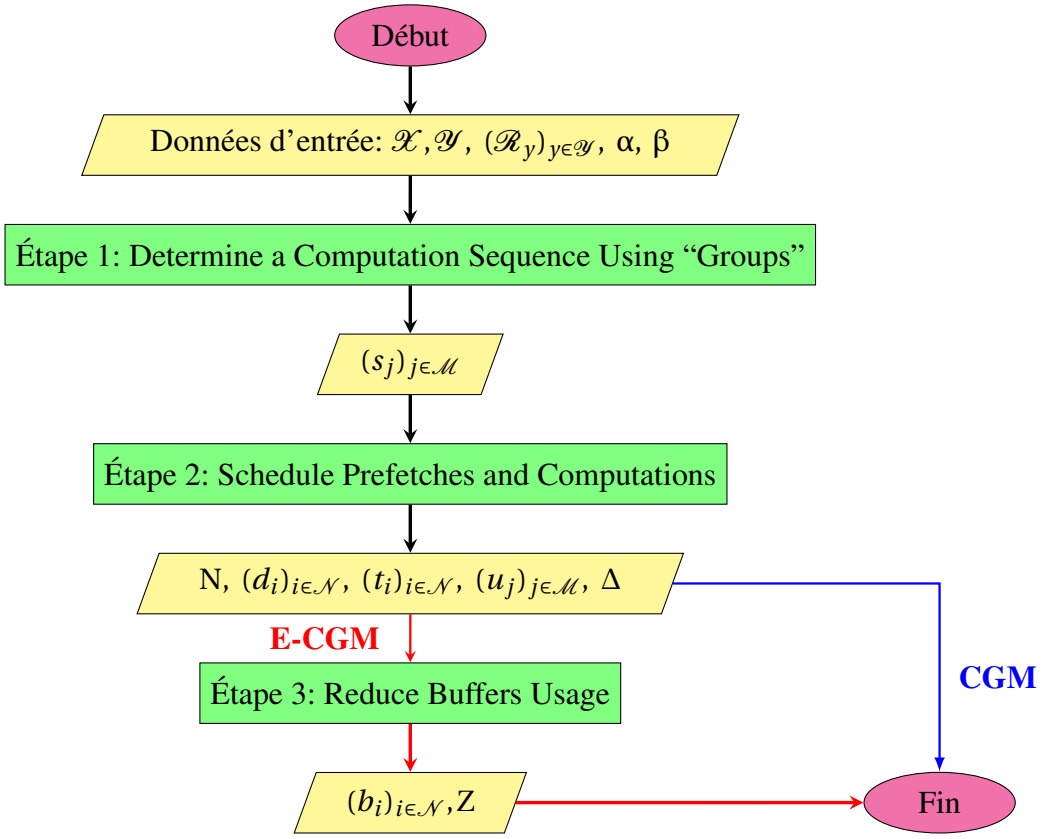


FIGURE 5.5 – Organigrammes pour les algorithmes CGM et E-CGM

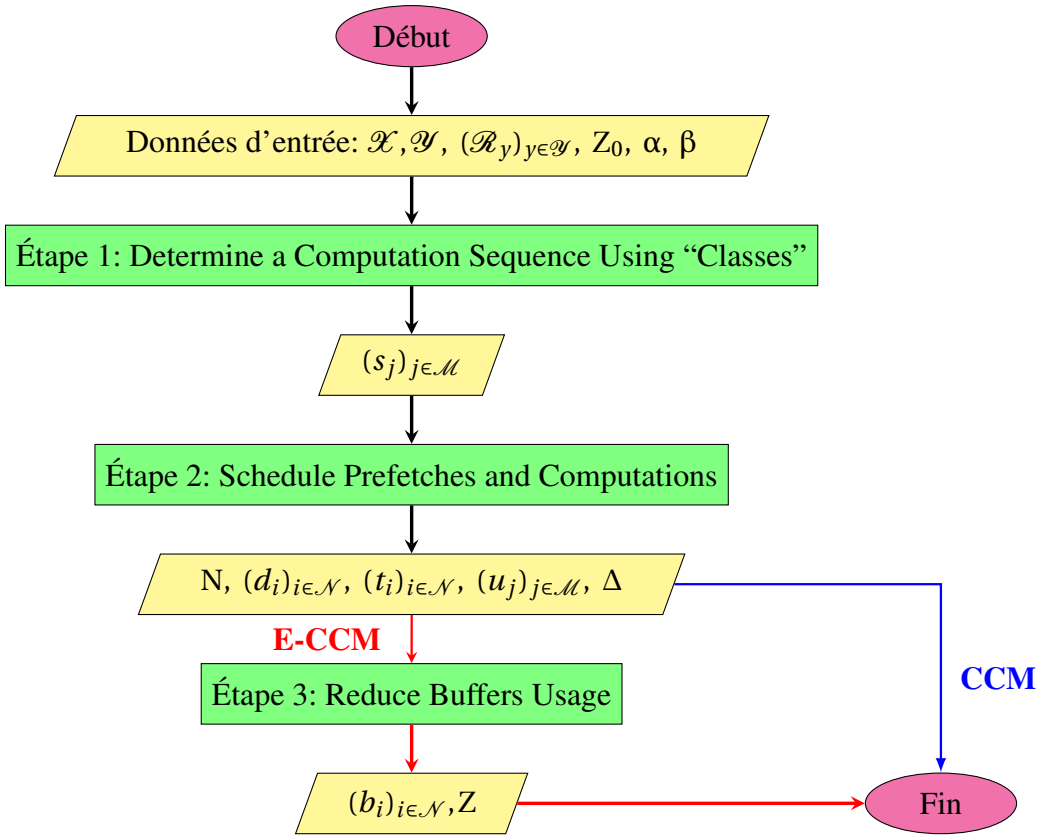


FIGURE 5.6 – Organigarmme pour les algorithmes CCM et E-CCM

5.5.5 Exemple

Dans cette section, nous illustrons le fonctionnement de différentes approches heuristiques décrites ci-avant sur l'exemple suivant. Nous considérons alors l'instance $I_{3-PSDPP}$ de notre problème d'origine 3-PSDPP définie par le quintuplet $(\mathscr{X}, \mathscr{Y}, r_{xy}, \alpha, \beta)$, où :

- ⋆ $\mathscr{X} = \{X_1, X_2, X_3, X_4, X_5, X_6\}$, avec X = 6 tuiles d'entrée à précharger ;
- ⋆ $\mathscr{Y} = \{Y_1, Y_2, Y_3\}$, avec Y = 3 tuiles de sortie à calculer ;
- ⋆ $\mathscr{R}_y, y \in \mathscr{Y}$ est donné par la matrice d'incidence Tuiles d'entrée/Tuiles de sortie r_{xy}, présentée à la Figure 5.7 ;
- ⋆ α = 2 unités de temps (durée pour précharger la Te $x, \forall x \in \mathscr{X}$) ;
- ⋆ β = 3 unités de temps (durée pour calculer la Ts $y, \forall y \in \mathscr{Y}$).

$$r_{xy} = \begin{array}{c} \\ X_1 \\ X_2 \\ X_3 \\ X_4 \\ X_5 \\ X_6 \end{array} \begin{array}{c} \begin{array}{ccc} Y_1 & Y_2 & Y_3 \end{array} \\ \begin{pmatrix} 0 & 1 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 0 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 0 & 1 \end{pmatrix} \end{array}$$

FIGURE 5.7 – Matrice d'incidence X × Y

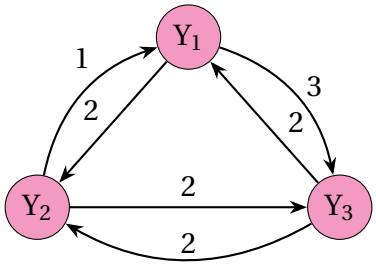


FIGURE 5.8 – Graphe $\overrightarrow{G}_T$

Les valeurs des différentes bornes inférieures, associées à l'instance $I_{3-PSDPP}$, sont résumées dans le Tableau 5.2 ainsi :

Bornes	lb_{N}	lb_{Z}	lb_1	lb_2	lb_{Δ}
$\mathrm{I}_{3-\mathrm{PSFPP}}$	6	4	15	15	15

TABLEAU 5.2 – $\mathrm{I}_{3-\mathrm{PSDPP}}$ - Valeurs des bornes inférieures

Prenons pour la séquence des calculs l'ordre de numérotation croissant. En supposant que le nombre de buffers est 5, l'algorithme KAD trouve une solution avec N = 6 préchargements, où le bloc de "0" dans la 5ème ligne de la matrice r_{xy} est basculé en "1". Les séquencements obtenus sont illustrés par la Figure 5.9.

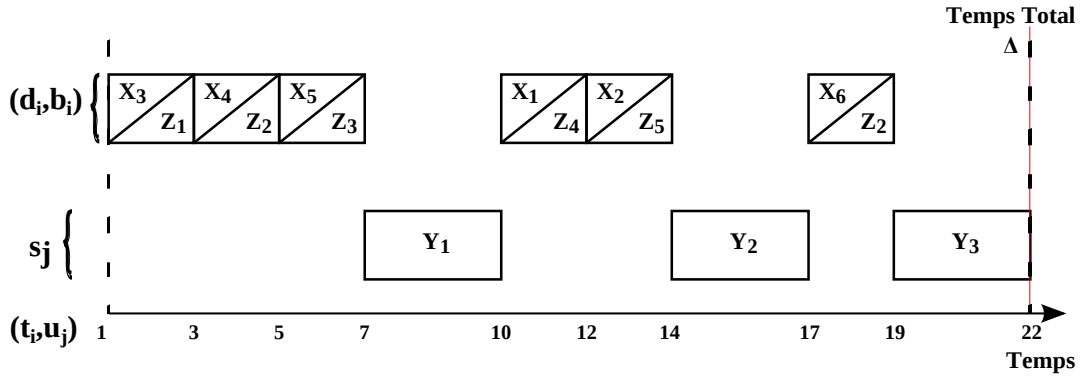


FIGURE 5.9 – $\mathrm{I}_{3-\mathrm{PSDPP}}$ - Solution de l'algorithme KAD

En exécutant l'algorithme KAP sur cet exemple, avec un nombre de buffers Z égal à 5 et une séquence des calculs $(s_j)_{j\in\mathscr{M}}$ donnée par $s_j = \mathrm{Y}_2, \mathrm{Y}_1, \mathrm{Y}_3$, où la séquence s_j est construite en résolvant une instance du problème ATSP, comme le montre le graphe $\overrightarrow{\mathrm{G}}_{\mathrm{T}}$ donné par la Figure 5.8, la Figure 5.10 donne les séquencements des préchargements et des calculs correspondants ainsi que la valeur des variables de sortie.

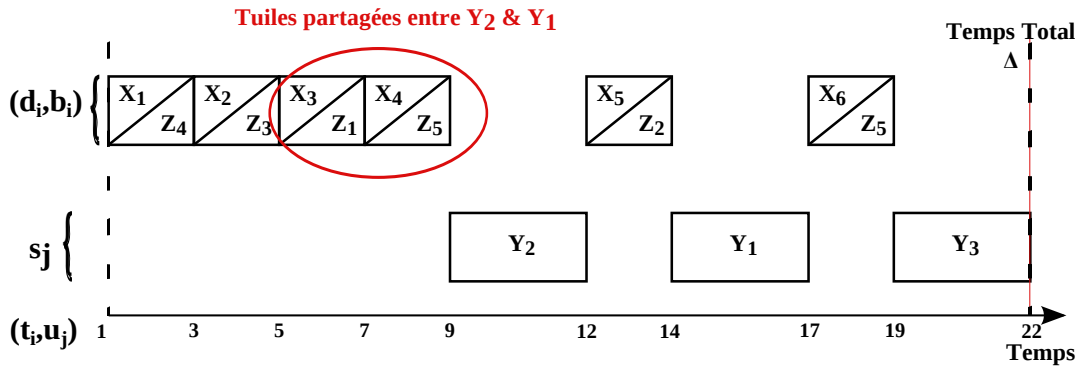


FIGURE 5.10 – $\mathrm{I}_{3-\mathrm{PSDPP}}$ - Solution de l'algorithme KAP

En reprenant les mêmes valeurs de Z et de $(s_j)_{j\in\mathscr{M}}$, la Figure 5.11 donne les séquencements des préchargements et des calculs correspondants avec les différentes valeurs des sorties, obtenus par l'algorithme SPbP.

Pour les trois algorithmes ECM, CGM et CCM, le nombre de préchargements ainsi que le nombre de buffers sont toujours égal 6. Nous donnons ici uniquement la solution obtenue par l'algorithme ECM, comme le montre la Figure 5.12, ci-après :

Finalement, en exécutant l'algorithme E-ECM, le nombre de buffers Z obtenu vaut 5.

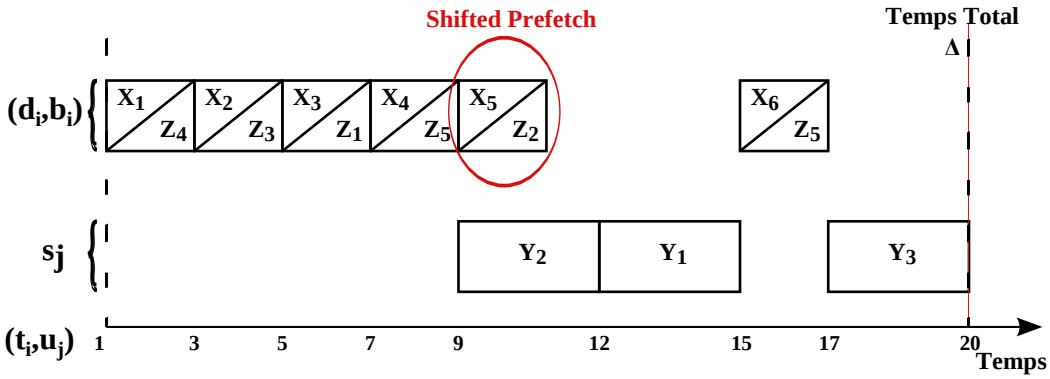


FIGURE 5.11 – $I_{3-PSDPP}$ - Solution de l'algorithme SPbP

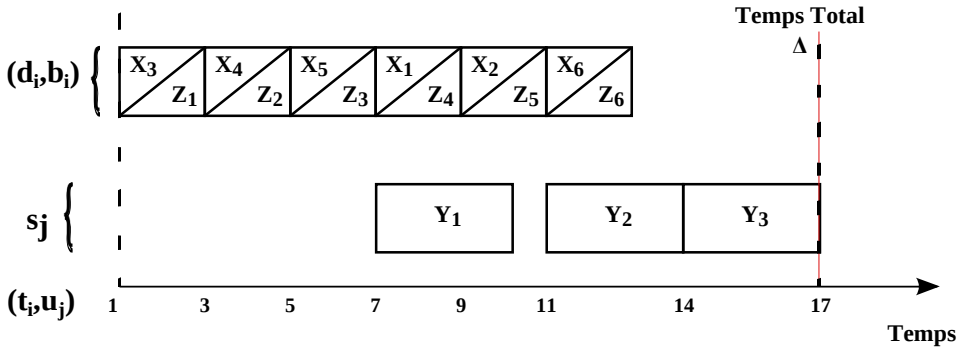


FIGURE 5.12 – $I_{3-PSDPP}$ - Solution de l'algorithme ECM

5.6 Conclusion

Dans ce chapitre, nous avons présenté une liste d'approches de résolution exactes et approchées pour le problème d'origine 3-PSDPP ainsi que certaines de ses variantes mono- et bi-objectif. En effet, nous sommes face à des problèmes combinatoires complexes, à savoir minimiser simultanément les trois enjeux électroniques classiques (coût, consommation d'énergie et performance) ou séparément chacun d'entre eux. Cependant, il n'existe pas de "solution optimale" unique, en particulier dans le cas du problème 3-PSDPP.

Les algorithmes issus de la PLNE proposés pour certaines variantes mono-objectif $\mathcal{NP}$-Complètes ne sont pas suffisamment efficaces pour résoudre les instances de grande taille. Pour cela, nous avons proposé une variété de méthodes de type *heuristique* visant une résolution rapide et efficace pour les différents problèmes (mono-, bi- et tri-objectif) et dans le but de fournir des solutions utiles à l'utilisateur de l'outil MMOpt.

Par conséquent, face à cette diversité, l'utilisateur de l'outil MMOpt peut faire son choix d'algorithmes par rapport aux critères (un ou plusieurs) qu'il cherche à les optimiser. Ceci joue un rôle fondamental dans la phase d'optimisation du fonctionnement des TPUs générés par l'outil MMOpt.

Chapitre 6

Implémentation et Validation Numérique et Pratique

Sommaire

6.1 Introduction

Ce chapitre est destiné à présenter une synthèse des principaux résultats qui ont été établis par notre problème d'origine 3-PSDPP ainsi que certaines de ses variantes mono- et bi-objectif. Il consiste essentiellement en l'évaluation numérique de la qualité des solutions produites par les différentes approches proposées. En effet, à partir de cette étude expérimentale, nous envisageons d'aider l'utilisateur de l'outil MMOpt à choisir sa meilleure approche parmi celles qui ont été présentées dans le Chapitre 5.

Afin d'évaluer la pertinence des résultats obtenus, nous avons effectué une série de tests sur des benchmarks disponibles dans la littérature ainsi que ceux fournis par le concepteur de l'outil MMOpt. Nous allons d'abord présenter le contexte de nos expérimentations, y compris les paramètres choisis ainsi qu'une description détaillée des instances utilisées. Nous présentons ensuite les résultats de différents tests menés sur ces instances, suivis d'une étude comparative dans laquelle nous examinons la performance en termes de qualité de solution ainsi que la vitesse de résolution de chacune de nos approches. Enfin, nous terminons notre étude expérimentale par une discussion qui comprend une brève synthèse sur les différents tests menés suivie d'une analyse critique des résultats obtenus.

6.2 Contexte d'implémentation et validation numérique

Dans cette section, nous décrivons d'abord les paramètres choisis ainsi que le protocole expérimental et ensuite les jeux de données testés.

6.2.1 Choix des paramètres

Nous rappelons que le paramètre α désigne la durée nécessaire pour précharger une tuile d'entrée, où cette durée est uniforme pour tous les préchargements. De même, le paramètre β désigne la durée nécessaire pour calculer une tuile de sortie, où cette durée est uniforme pour tous les calculs.

Dans cette section, nous présentons notre démarche pour établir le choix de ces paramètres. Afin d'éviter le choix aléatoire, nous cherchons à estimer la valeur du rapport $\frac{\beta}{\alpha}$, par exemple, par rapport au nombre total de préchargements N. En effet, plusieurs pistes peuvent être envisagées. Nous nous focaliserons ici en particulier sur celle qui sera décrite par la suite.

Soient I = ($\mathscr{X}$, $\mathscr{Y}$, $\mathscr{R}_y$, α, β) une instance du problème 3-PSDPP et S = (N, Z, $(p_i)_{i\in\mathscr{N}}$, $(c_j)_{j\in\mathscr{M}}$, Δ) la solution à une telle instance I.

Pour chercher un ordonnancement des calculs et celui des préchargements avec un temps total de traitement optimal Δ^*, c'est-à-dire des séquencements idéaux (sans "temps morts"), il est alors nécessaire d'avoir un temps total de traitement Δ égal à sa borne inférieure lb_Δ. En effet, pour déterminer Δ^*, il suffit simplement de calculer le maximum entre le temps total pour effectuer tous les préchargements ($\alpha * N$) et celui pour effectuer tous les calculs ($\beta * Y$).

Plus formellement on a l'équation suivante : $\Delta^* = \max(\alpha * N, \beta * Y)$. Cela signifie l'utilisation optimale des ressources. Autrement dit, on a $\alpha * N = \beta * Y$. Or, on sait que le nombre de préchargements possible ne peut pas être en dessous de sa borne inférieure lb_N, c'est-à-dire $N \geq lb_N = X'$. Nous considérons également la valeur ub_N, comme une borne supérieure sur la valeur optimale en N (au-delà de cette valeur, il est inutile d'effectuer des préchargements), c'est-à-dire, pour toute solution réalisable, on a $N \leq ub_N$. Cela signifie que le nombre de préchargements N peut être généralement défini dans l'intervalle $[lb_N, ub_N]$.

En résumé, la valeur du rapport $\frac{\beta}{\alpha}$ peut ainsi être déterminée dans l'intervalle $[\frac{lb_N}{Y}, \frac{ub_N}{Y}]$,

où $lb_N = X'$ et ub_N peut être défini par le nombre total de tuiles d'entrée requises $\sum_{y \in \mathscr{Y}} |\mathscr{R}_y|$ ou encore par la constante $X' * Y$ (c'est-à-dire chaque tuile de sortie nécessite exactement X' tuiles).

Par exemple, prenons le noyau N° 10 (le plus grand noyau en termes de nombre de tuiles d'entrée à précharger), avec X = 7040, Y = 428, lb_N = 2272 et une valeur supérieure sur le nombre total de préchargements $ub_N = \sum_{y \in \mathscr{Y}} |\mathscr{R}_y| = 26620$, plusieurs valeurs peuvent alors être possibles pour le rapport $\frac{\beta}{\alpha}$ dans l'intervalle $\{5, \ldots, 62\}$. Cela signifie que β vaut au moins 5 fois plus que α. Autrement dit, si l'on considère $\alpha = 1$ unité de temps, β vaut au minimum 5 et au maximum 62 unités de temps. Ceci se traduit par une augmentation du temps accordé pour l'exécution de chacune des approches proposées sur ce noyau.

Par conséquent, pour des raisons de temps de calcul et face à la diversité des noyaux utilisés (cf. Tableau 6.2 donné dans la Section 6.2.3), les valeurs réelles des paramètres α et β ne sont pas données ici (chaque noyau a ses propres valeurs). Par conséquent, pour exécuter les différents tests, nous avons utilisé une durée de préchargement α égale à 2 unités de temps et une durée de calcul β égale à 3 unités de temps, avec un rapport $\frac{\beta}{\alpha}$ égal à 1.5.

6.2.2 Protocole expérimental

Les algorithmes proposés ont été codés en python, sauf la partie ATSP qui est ré-encodée comme un TSP, et résolue au sein de l'outil logiciel Concorde par l'heuristique *Chained Lin-Kernighan* [10]. En outre, nous avons utilisé le solveur *Gurobi* pour résoudre les trois formulations en PLNE données dans le Chapitre 3.

Ces différentes expérimentations ont été menées sur un PC muni d'un processeur Intel Core i5 à 2.60 GHz et 4 Go de RAM.

6.2.3 Description des instances

L'évaluation numérique des différentes méthodes a été effectuée en utilisant les deux familles d'instances suivantes :

i) La première famille est un ensemble de benchmarks utilisés, dans le contexte du problème ToSP, par : Bard (1988) [14], Hertz et al. (1998) [67], Al-Fawzan et al. (2003) [4] et Zhou et al. (2005) [119]. Ces jeux de données sont extraits de la base de données disponible à l'adresse :`http://www.unet.edu.ve/~jedgar/ToSP/ToSP.htm`.

Pour tester les trois formulations en PLNE proposées au Chapitre 5 (cf. Section 3.6), nous utilisons un ensemble de seize instances, comme le montre le Tableau 6.1 ci-après.

Le fait de devoir choisir ces instances peut être justifié par l'absence d'instances standards de petite taille dans le contexte de notre problème d'origine. En outre, l'étude comparative faite afin de positionner notre problème 3-PSDPP par rapport à la littérature prouve l'équivalence entre les deux sous-problèmes, PSP et DPP, et ceux de ToSP et TP (cf. Chapitre 4-Section 4.3)

Le Tableau 6.1 présente les caractéristiques d'instances générées [4, 14, 67, 119]. Nous pouvons alors distinguer, en prenant comme critère de classement des instances le nombre de tuiles de sortie Y à calculer, six sous-familles, où chacune d'entre elles comprend une ou plusieurs instances. Pour chaque instance, notée par $Z\zeta_Y^X$ (colonne **Notation**), nous spécifions la liste des paramètres suivants :

— Y : le nombre de tuiles de sortie à calculer ;

— X : le nombre total de tuiles d'entrée à précharger ;

— **Min.** : le nombre minimum de tuiles d'entrée requises par une des tuiles de sortie ;

— **Max.** : le nombre maximum de tuiles d'entrée requises par une des tuiles de sortie ;

— Z : le nombre de buffers internes ;

— **Source** : la référence de chaque instance.

Instance N°		Y	X	**Min.**	**Max.**	Z	**Notation**	**Source**
1	**1a**	10	9	2	4	4	$4\zeta_{10}^{9}$	[14, 119]
	1b	-	10	2	4	4	$4\zeta_{10}^{10}$	[67, 4]
	1c	-	15	3	6	6	$6\zeta_{10}^{15}$	[119]
2	**2a**	15	12	3	6	6	$6\zeta_{15}^{12}$	[14, 119]
	2b	-	20	3	6	6	$6\zeta_{15}^{20}$	[67]
3	**3a**	20	15	3	8	8	$8\zeta_{20}^{15}$	[4]
	3b	-	16	3	8	8	$8\zeta_{20}^{16}$	[14, 119]
	3c	-	20	4	10	10	$10\zeta_{20}^{20}$	[14, 119]
	3d	-	30	9	24	24	$24\zeta_{20}^{30}$	[14, 119]
	3e	-	36	9	24	24	$24\zeta_{20}^{36}$	[14, 119]
	3f	-	40	11	30	30	$30\zeta_{20}^{40}$	[119]
4	**4a**	30	25	4	10	10	$10\zeta_{30}^{25}$	[4]
	4b	-	40	6	15	15	$15\zeta_{30}^{40}$	[67]
5	**5a**	40	30	6	15	15	$15\zeta_{40}^{30}$	[4]
	5b	-	60	7	20	20	$20\zeta_{40}^{60}$	[67]
6		50	40	9	20	25	$25\zeta_{50}^{40}$	[4]

TABLEAU 6.1 – Caractéristiques d'instances générées [4, 14, 67, 119]

ii) La deuxième famille comprend un ensemble de douze instances réelles fournies par Mancini et al. [89, 90].

Les caractéristiques de ces jeux de données sont décrites dans le Tableau 6.2. En effet, pour chaque noyau non-linéaire, nous spécifions :

— **La structure de données d'entrée** :

Les jeux de données sont des variations de six noyaux non-linéaires typiques des étapes de pré-traitement des applications de vision : la transformée "polar", la transformée "fisheye", "fd (face detection) resize", "fd haar" et "Cameleon". Ces noyaux sont déclinés selon les données d'entrée : "image" ou "(an)isotropic mipmap multi-resolution" ou "pyramidal multi-resolution".

Les neuf premiers noyaux représentent des transformations géométriques non-linéaires (voir Thornton et al. [113] et Bellas et al. [17]).

Le dixième noyau, qui représente un noyau d'une application de détection des visages (fd) en fonction des caractéristiques "Haar", crée une image pyramidale à multi-résolution (voir Viola et al. [117]).

Les deux derniers représentent une transformation de type "system dedection (sd)".

— **Dim.** : la dimension de données (tuile d'entrée ou de sortie) en termes d'organisation de leur stockage (tableaux 2D, 3D, 4D) ;

— X : le nombre de tuiles d'entrée à précharger ;

— Y_0 : le nombre initial de tuiles de sortie ;

— Y : le nombre de tuiles de sortie à calculer, où Y est plus petit que Y_0 ;

— **Min.** : le nombre minimum de tuiles d'entrée requises par une des tuiles de sortie ;

— **Max.** : le nombre maximum de tuiles d'entrée requises par une des tuiles de sortie ;

Comme le montre le Tableau 6.2 ci-dessus, le nombre de tuiles de l'image d'entrée varie entre 60 et 7000 tuiles d'entrée, et celui des tuiles de sortie à calculer varie entre 60 et 3450 tuiles de sortie.

6.2.4 Calcul des bornes

Cette section est consacrée au calcul des différentes bornes inférieures (lb_Z, lb_N, lb_1, lb_2 et lb_Δ) sur les jeux de données décrits dans le Tableau 6.2. En effet, ces bornes permettent d'évaluer la performance des approches décrites dans le Chapitre 5, comme nous le verrons à la Section 6.4. Les valeurs de ces différentes bornes sont données dans le Tableau 6.3. Nous rappelons ici que ces valeurs sont données par les formules suivantes :

— $lb_Z = \max_{y \in \mathscr{Y}} |\mathscr{R}_y|$

— $lb_N = X'$

— $lb_1 = \alpha * X' + \beta$

— $lb_2 = \alpha * \min_{y \in \mathscr{Y}} |\mathscr{R}_y| + \beta * Y$

— $lb_\Delta = \max(lb_1, lb_2)$

N°	Noyau	Structure de données d'entrée	Tuiles d'entrée		Tuiles de sortie			Tuiles d'entrée requises	
			Dim	X	Dim	Y_0	Y	Min.	Max.
1	**Test2D**	image	2D	256	2D	64	64	4	4
2	**Test2D PE**	image	2D	64	2D	256	256	1	1
3	**Fisheye**	image	2D	176	2D	176	158	1	9
4	**Fisheye**	mipmap isotropic	3D	352	2D	176	158	2	13
5	**Fisheye**	mipmap anisotropic	4D	704	2D	176	158	3	21
6	**Polar**	image	2D	169	2D	112	112	2	8
7	**Polar**	mipmap isotropic	3D	845	2D	112	112	2	12
8	**Polar**	mipmap anisotropic	4D	4225	2D	112	112	5	20
9	**Fd Resize**	mipmap isotropic	3D	1280	3D	3520	1186	1	13
10	**Fd Haar**	pyramidal integral image	4D	7040	3D	2112	428	28	96
11	**Cameleon**	image	3D	1200	2D	1350	877	1	9
12	**Cameleon Sd**	image	3D	4800	2D	5400	3353	1	10

TABLEAU 6.2 – Caractéristiques des jeux de données [89, 90]

Noyau N° B.inf	lb_Z	lb_N	Bornes sur Δ		
			lb_1	lb_2	lb_Δ
1	4	256	516	201	516
2	1	64	132	771	771
3	9	176	356	477	477
4	13	224	452	479	479
5	21	360	724	481	724
6	8	146	296	341	341
7	12	80	164	341	341
8	20	244	492	347	492
9	13	429	862	3561	3561
10	96	2272	4548	1341	4548
11	9	784	1572	2634	2634
12	10	2868	5740	10062	10062

TABLEAU 6.3 – Valeurs de bornes inférieures : lb_Z, lb_N, lb_1, lb_2 & lb_Δ

6.3 Évaluation des méthodes exactes

Dans cette section, nous décrivons les résultats des tests effectués pour évaluer la performance des différentes méthodes exactes basées sur les formulations en PLNE : `PSP-PLNE` et `MCT-PLNE` (cf. Chapitre 3).

Les expérimentations sont effectuées sur les différentes instances données au Tableau 6.1 en utilisant le solveur Gurobi. Le temps de calcul nécessaire pour la résolution de chacune des instances est limité à 500 secondes.

Les deux Tableaux 6.4 et 6.5 présentent les différents résultats obtenus. Pour chaque méthode, la colonne “Instance N° ” est le nom du jeu de données résolu. Les deux colonnes “B.inf” et “Obj.val” indiquent respectivement la borne inférieure obtenue par le programme correspondant et la valeur de la meilleure solution trouvée. En outre, la colonne “Temps” représente le temps de calcul. Ce dernier est exprimé en secondes (s). Ensuite, la colonne “Nœuds” représente le nombre de nœuds qui composent l’arbre de recherche dans l’algorithme de branchement et coupes. Enfin, la colonne “Gap” représente l’écart relatif de la borne inférieure avec la meilleure solution trouvée, exprimé en (%).

Le Tableau 6.4 illustre les résultats obtenus dans le contexte des deux méthodes PSP_a et PSP_b, où :

— PSP_a est basée sur la formulation initiale du problème PSP : `PSP-PLNE` (cf. Chapitre 3).

— PSP_b est basée sur la formulation `PSP-PLNE` tout en réduisant le nombre de tuiles de sortie à calculer Y vers une nouvelle valeur Y′ (cf. Section 5.4) du Chapitre 5.

Dans le Tableau 6.4, nous observons que les deux méthodes PSP_a et PSP_b peuvent résoudre à l’optimalité le même nombre d’instances, en particulier les trois premières instances. En outre, PSP_a est plus rapide que PSP_b en termes du temps de calcul. Cependant, pour les instances — 2a, 2b, 4a, 5a et 5b — PSP_b est meilleure que PSP_a en termes d’écart à l’optimalité (Gap). De plus, de nouvelles bonnes bornes inférieures ont été déterminées par PSP_b dans

Instance N°		PSP$_a$					PSP$_b$				
		B.inf	**Obj.val**	**Nœuds**	**Temps (s)**	**Gap (%)**	**B.inf**	**Obj.val**	**Nœuds**	**Temps (s)**	**Gap (%)**
1	**1a**	12	12	73369	77	0.0	12	12	76735	92	0.0
	1b	11	11	1368	2	0.0	11	11	1415	3	0.0
	1c	18	18	112733	233	0.0	18	18	139560	262	0.0
2	**2a**	13	24	153690	500	45.83	13	22	146401	500	40.90
	2b	20	31	74915	500	35.48	20	30	54967	500	33.33
3	**3a**	15	30	51150	500	50.00	15	32	52092	500	53.12
	3b	16	39	37474	500	58.97	16	42	46573	500	61.90
	3c	20	38	35285	500	47.36	19	38	45233	500	50.00
	3d	29	58	20585	500	50.00	22	60	21392	500	63.33
	3e	20	70	20827	500	71.42	23	68	20603	500	66.17
	3f	23	71	20777	500	67.60	23	73	21262	500	68.49
4	**4a**	25	96	1591	500	73.95	25	95	1881	500	73.68
	4b	16	153	1680	500	89.54	15	168	1497	500	91.07
5	**5a**	30	150	576	500	80.00	30	134	840	500	77.61
	5b	15	324	605	500	95.37	15	313	331	500	95.20
6		40	230	77	500.01	82.60	40	253	64	500	84.18

TABLEAU 6.4 – Comparaison entre les résultats de PSP$_a$ et ceux de PSP$_b$

le contexte de trois instances 3c, 3d et 4b contre une mauvaise borne pour l'instance 3e. En revanche, pour les autres instances (3a, 3b, 3f et 6), PSP$_a$ reste meilleure que PSP$_b$ en termes de valeur objective et d'écart à l'optimalité (Gap).

De la même manière, le Tableau 6.5 donne les résultats des deux méthodes MCT-PSDPP$_a$ et MCT-PSDPP$_b$ basées sur la formulation `MCT-PLNE`, où :

— MCT-PSDPP$_a$ est la formulation initiale `MCT-PLNE` (cf. Chapitre 3).

— MCT-PSDPP$_b$ est basée sur la formulation `MCT-PLNE` en réduisant le nombre de tuiles de sortie à calculer Y vers une nouvelle valeur Y$'$ (cf. Section 5.4) du Chapitre 5.

Comme on peut le remarquer dans le Tableau 6.5, seules les huit premières instances ont été résolues à l'optimalité en utilisant les deux méthodes MCT-PSDPP$_a$ et MCT-PSDPP$_b$ avec un temps maximal de 471 secondes pour l'instance N^o 3c. Cependant, pour les autres instances, nous observons que le pourcentage d'écart par rapport à la borne inférieure est à moins de 50 %. Une autre observation intéressante est que la performance de la formulation `MCT-PLNE` se dégrade car il ne peut résoudre que les instances avec au plus Y = 20 tuiles de sortie à calculer et jusqu'à X = 20 tuiles d'entrée à précharger. En revanche, pour nos instances réelles [89, 90], le nombre minimal de tuiles d'entrée ainsi que de tuiles de sortie est de 64 tuiles. Il est alors impossible de les résoudre en moins de 500 secondes. C'est pourquoi, nous avons utilisé des instances de petite taille pour évaluer la performance de ces deux formulations.

Dans un dernier temps, il nous a été impossible de résoudre, en moins de 500 secondes, les instances données au Tableau 6.1 en utilisant les deux méthodes `BCMCTP-PLNE1` et `BCMCTP-PLNE2` (cf. Chapitre 3). Par exemple, pour l'instance N^o 1a, la `BCMCTP-PLNE1` ainsi que la `BCMCTP-PLNE2` donnent un écart à l'optimalité (Gap) de 10% pour une durée d'environ 1500 secondes.

Cela peut être justifié par le fait que les deux formulations, `BCMCTP-PLNE1` et `BCMCTP-PLNE2`, ont un grand nombre de variables et de contraintes. Par exemple, pour l'instance N^o 1a, où Y = 10 tuiles de sortie à calculer et X = 9 tuiles d'entrée à précharger, la `BCMCTP-PLNE1` a 2465 variables et 7699 contraintes et la `BCMCTP-PLNE1` a 2552 variables et 8795 contraintes.

En revanche, la résolution de la relaxation linéaire de ces deux formulations nous a également permis d'obtenir des nouvelles bornes sur Δ qui sont supérieures à $\beta * Y$ (dans le cas de `BCMCTP-PLNE1`) et inférieure à $\beta * Y + \alpha$ (dans le cas de `BCMCTP-PLNE2`).

D'après les résultats présentés ci-dessus et les comparaisons effectuées, nous constatons que les trois méthodes : `PSP-PLNE` et `MCT-PLNE` fournissent des solutions optimales pour les problèmes de petite taille (jusqu'à 20 tuiles de sortie à calculer et jusqu'à 20 tuiles d'entrée à précharger) en un temps d'exécution raisonnable, mais, malheureusement, elles sont totalement inefficaces pour les problèmes de grande taille. En revanche, les deux méthodes `BCMCTP-PLNE1` et `BCMCTP-PLNE2` ne résolvent aucune instance.

Instance N°		MCT-PSDPP$_a$					MCT-PSDPP$_b$				
		B.inf	**Obj.val**	**Nœuds**	**Temps (s)**	**Gap (%)**	**B.inf**	**Obj.val**	**Nœuds**	**Temps (s)**	**Gap (%)**
1	**1a**	34	34	0	≤1	0.0	34	34	0	≤1	0.0
	1b	35	35	456	≤1	0.0	35	35	5076	1	0.0
	1c	38	38	2626	2	0.0	38	38	5091	6	0.0
2	**2a**	53	53	2170	2	0.0	53	53	207	1	0.0
	2b	53	53	9139	213	0	53	53	14846	391	0.0
3	**3a**	69	69	3090	16	0.0	69	69	3678	15	0.0
	3b	71	71	3591	87	0.0	71	71	7477	113	0.0
	3c	70	71	17414	471	0.0	68	74	8301	309	0.0
	3d	60	97	732	500	38.14	61	97	821	500	37.11
	3e	60	105	744	500	42.85	60	105	730	500	42.85
	3f	60	116	591	500	48.27	60	115	511	500	47.82
4	**4a**	90	106	757	500	15.09	90	104	751	500	13.46
	4b	90	123	734	500	26.82	90	122	585	500	26.22
5	**5a**	120	146	745	500	17.80	120	145	732	500	17.24
	5b	120	197	0	500	39.08	120	195	0	501	38.46
6		150	201	42	500	25.37	150	202	0	500	25.74

TABLEAU 6.5 – Comparaison entre les résultats de MCT-PSDPP$_a$ et ceux de MCT-PSDPP$_b$

6.4 Évaluation des méthodes heuristiques

Dans cette section, nous présentons les résultats numériques obtenus en exécutant les différentes heuristiques sur les différents noyaux donnés au Tableau 6.2 suivis d'une étude comparative afin d'évaluer leurs performances.

6.4.1 Axes d'évaluation

Dans le contexte de notre problème d'origine 3-PSDPP et de ses diverses variantes (mono- et bi-objectif), nous nous intéressons à évaluer la performance de nos approches en s'appuyant particulièrement sur les trois axes principaux, tels que décrits ci-après. L'objectif derrière cette diversité des axes d'évaluation consiste en une garantie de performance pour nos différentes heuristiques proposées.

i) **Évaluation par rapport à la solution optimale** :
D'une manière générale, pour toute instance I, la performance d'une heuristique, notée par $\Theta_H(I)$, est calculée par le rapport entre la valeur de la solution retournée par cette heuristique H(I) et la valeur de la solution optimale OPT(I). Ce rapport est donné par la formule suivante :

$$\Theta_H(I) = H(I)/OPT(I) \tag{6.1}$$

En revanche, lorsque la solution optimale n'est pas calculable, il est également possible d'étudier expérimentalement le comportement de l'heuristique par exemple en comparant ses performances pour plusieurs instances (cf. Section 6.2.3) avec les performances d'autres heuristiques, soit avec des bornes inférieures de la solution optimale.

ii) **Évaluation par rapport aux méthodes actuellement utilisées dans l'outil MMOpt** :
Pour effectuer cette évaluation, nous utilisons les deux formules décrites par les équations données ci-après, et on les note respectivement $\Phi_H(I)$ et $\Psi_H(I)$. En effet, la première formule $\Phi_H(I)$ donne le gain absolu, exprimé en (%), c'est-à-dire le gain obtenu par rapport à l'une des deux méthodes M_1 et M_2 de l'outil MMOpt. En revanche, la deuxième $\Psi_H(I)$ représente le gain relatif, exprimé en (%), c'est-à-dire le gain obtenu par rapport aux M_1 et M_2 et par rapport à la borne inférieure.
Ces deux formules seront appliquées pour chacun des critères d'optimisation : Z, N et Δ.

$$\Phi_H(I) = \left(\frac{MMOpt(I) - H_C(I)}{MMOpt(I)}\right) * 100 \tag{6.2}$$

$$\Psi_H(I) = \left(\frac{MMOpt(I) - H_C(I)}{MMOpt(I) - LB_C(I)}\right) * 100 \tag{6.3}$$

iii) **Évaluation par rapport aux bornes inférieures** :
Pour évaluer la performance de chacune de nos approches par rapport aux différentes bornes inférieures développées à la Section 6.2.4, nous utilisons la formule $\Upsilon_H(I)$ donnée ci-après. En effet, $\Upsilon_H(I)$ est donnée par le rapport entre la valeur du critère d'optimisation (à minimiser) H_C obtenue par l'heuristique H et la valeur de la borne inférieure sur ce critère LB_C.

$$\Upsilon_H(I) = H_C(I)/LB_C(I) \tag{6.4}$$

6.4.2 Comparaison d'algorithmes KAP et SPbP avec M_1 et M_2

Nous analysons d'abord les performances de deux algorithmes KAP et SPbP (cf. Section 5.5.3 du Chapitre 5). En effet, nous donnons dans un premier temps les résultats numériques obtenus par ces deux algorithmes ainsi qu'une comparaison par rapport aux travaux antérieurs. Dans un deuxième temps, nous évaluons leurs performances par rapport aux bornes inférieures.

Nous rappelons aussi que le nombre de buffers Z est considéré comme une donnée d'entrée. Afin de comparer ces deux méthodes à celles utilisées dans l'outil MMOpt (M_1 et M_2), nous définissons deux valeurs différentes pour Z, qu'on note Z_1 et Z_2, comme suit :

— La valeur de Z_1 est donnée par l'algorithme M_1 de MMOpt lorsque le nombre de buffers obtenu est maximum et le temps total de traitement est minimisé.

— La valeur de Z_2 donne le nombre minimum de buffers. Autrement dit, Z_2 est égal à la borne inférieure lb_Z.

Les résultats numériques obtenus par l'exécution de deux algorithmes KAP et SPbP dans ces deux cas sont décrits dans les deux Tableaux 6.6 et 6.7. La ligne 1 donne le numéro du noyau. La ligne 2 donne, respectivement, le nombre de buffers Z, le nombre de préchargements N et le temps total de traitement Δ fournis par les deux algorithmes M_1 (Tableau 6.6) et M_2 (Tableau 6.7) de l'outil MMOpt (ligne 2) qui est notre référence.

Dans le Tableau 6.6 (resp. Tableau 6.7), pour le premier cas Z_1 (resp. le second cas Z_2) spécifié à la ligne 3, les valeurs de N et Δ obtenus par l'algorithme KAP sont ensuite données dans la ligne 4. La cinquième ligne montre les gains absolus de l'algorithme KAP (resp. perte dans certains cas), par rapport à M_1 (resp. à M_2), qui sont mesurés par la formule Φ_H donnée par l'équation 6.2. En revanche, les gains relatifs de cet algorithme, mesurés par la formule Ψ_H donnée par l'équation 6.3, sont représentés par la ligne 6.

De même, les valeurs de N et Δ obtenues par l'algorithme SPbP sont ensuite données à la ligne 7. Les lignes 8 et 9 donnent, respectivement, les gains absolus et relatifs (resp. perte absolue et relative dans certains cas) de cet algorithme en utilisant les mêmes formules précédentes : Φ_H et Ψ_H.

Enfin, la ligne CPU (s) donne les temps d'exécution de chacun de deux algorithmes KAP et SPbP sur les différents noyaux.

Comme illustré dans les deux Tableaux 6.6 et 6.7, la réduction moyenne du nombre de préchargements N, entre l'algorithme KAP et ceux de MMOpt, est de 16.8 % (absolue) et de 37.7 % (relative) dans le cas de Z_1, respectivement, de 12.5 % (absolue) et de 27.4 % (relative) dans le cas de Z_2. Cependant, le temps total de traitement Δ fourni par l'algorithme KAP est plus grand que celui de M_1 et M_2, avec un pourcentage d'augmentation de 21.6 % (absolu) et de 14.9 % (relatif) pour Z_1, respectivement, de 15.8 % (absolu) et de 51.1 % (relatif) pour Z_2. Ceci est dû à l'absence de recouvrement entre les préchargements et les calculs dans les deux séquencements (préchargements et calculs) fournis par l'algorithme KAP.

De même, en raison de la réutilisation de l'algorithme KAP comme une sous-routine de l'algorithme SPbP, le trafic vers la mémoire externe (nombre de préchargements) est réduit avec le même pourcentage dans les deux cas celui de Z_1 et de Z_2. Par ailleurs, contrairement à l'algorithme KAP, minimiser le temps total de traitement Δ par l'algorithme SPbP entraîne une réduction moyenne de 5.9 % (absolu) et de 13.6 % (relatif) pour Z_1, respectivement, de 6.1 % et de 13 % pour Z_2.

En outre, les expériences menées montrent qu'en particulier, pour le plus grand noyau N° 10 en termes de nombre de tuiles d'entrée à précharger (X = 7040), les deux algorithmes KAP et SPbP donnent de meilleurs résultats que ceux de M_1 et M_2 pour l'outil MMOpt.

Par contre, pour les deux derniers noyaux (N° 11 et N° 12), les deux algorithmes M_1 et M_2 donnent de meilleurs résultats en termes de temps total de traitement Δ que ceux fournis par

Algo.Noyau N°		**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**Av. Gain (%)**
M_1 **(référence)**	Z_1	8	2	12	20	29	11	-	28	18	139	12	14	
	N	256	64	289	395	640	243	-	478	1710	3640	1390	5269	
	Δ	516	772	711	907	1388	549	-	1021	5075	7899	3552	13444	
KAP (Z_1 **fixé**)	N	256	64	232	298	457	192	-	353	1283	2560	1293	5053	
	Δ	705	897	939	1071	1389	721	-	1043	6125	6405	5218	20166	
	CPU (s)	0.109	0.125	0.312	0.608	0.624	0.265	-	3.604	14.456	22.829	12.513	>5 min	
Gain (absolu)	N	0	0	19.7	24.5	28.5	20.9	-	26.1	24.9	**29.6**	6.9	4.1	**16.8**
	Δ	-36.6	-16.1	-32.0	-18.0	-0.1	-31.3	-	-2.1	-20.6	**18.9**	-46.9	-50	-21.6
Gain (relatif)	N	0	0	50.4	56.7	65.3	52.5	-	53.4	33.3	**78.9**	16.0	8.9	**37.7**
	Δ	0	-12.5	-97.4	-38.31	-7.7	-82.69	-	-4.1	-69.3	**44.5**	-181.4	-198.7	-14.9
SPbP (Z_1 **fixé**)	N	256	64	232	298	457	192	-	353	1283	2560	1293	5053	
	Δ	531	833	708	795	1113	545	-	851	4629	5849	3658	14137	
	CPU (s)	0.109	0.125	0.312	0.608	0.624	0.265	-	3.604	14.456	22.829	12.513	>5 min	
Gain (absolu)	N	0	0	19.7	24.5	28.5	20.9	-	26.1	24.9	**29.6**	6.9	4.1	**16.8**
	Δ	-2.9	-7.9	0.4	12.3	19.8	0.7	-	16.6	08.7	**25.9**	-2.9	-5.1	**5.9**
Gain (relatif)	N	0	0	50.4	56.7	65.3	52.5	-	53.4	33.3	**78.9**	16.0	8.9	**37.7**
	Δ	0	-6.1	1.2	26.16	36.9	1.92	-	32.1	29.4	**61.1**	-11.5	-20.4	**13.6**

TABLEAU 6.6 – Évaluation d'algorithmes KAP (Z_1) et SPbP (Z_1) Vs. M_1

Algo.Noyau N°		**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**Av. Gain (%)**
M_2(**référence**)	Z_2	4	1	9	13	21	8	12	20	13	96	9	10	
	N	256	64	289	395	640	243	169	478	1710	3640	1390	5269	
	Δ	705	961	762	976	1457	651	469	1081	5129	8070	3690	13723	
KAP (Z_2 **fixé**)	N	256	64	245	322	517	213	119	405	1458	2997	1310	5124	
	Δ	705	897	965	1119	1509	763	575	1147	6475	7279	5252	20308	
	CPU (s)	0.124	0.125	0.28	0.64	0.55	0.28	0.514	3.176	14.557	24.152	11.796	>4 min	
Gain (absolu)	N	0	0	15.2	18.4	19.2	12.3	29.5	15.2	14.7	**17.6**	5.7	2.7	**12.5**
	Δ	0	6.6	-26.6	-14.6	-3.5	-17.2	-22.6	-6.1	-26.2	**9.8**	-42.3	-47.8	-15.8
Gain (relatif)	N	0	0	38.9	42.6	43.9	30.9	56.1	31.1	19.6	**47.0**	13.2	6.0	**27.4**
	Δ	0	14.3	-71.2	-28.77	-7.0	-36.12	-82.81	-11.2	-85.8	**22.4**	-147.9	-179.8	-51.1
SPbP (Z_2 **fixé**)	N	256	64	245	322	517	213	119	405	1458	2997	1310	5124	
	Δ	705	897	745	871	1226	601	440	947	4916	6789	3751	14520	
	CPU (s)	0.124	0.125	0.28	0.64	0.655	0.28	0.514	3.176	14.557	24.152	11.796	>4 min	
Gain (absolu)	N	0	0	15.2	18.4	19.2	12.3	29.5	15.2	14.7	**17.6**	5.7	2.7	**12.5**
	Δ	0	6.6	2.2	10.7	15.8	7.6	6.1	12.3	4.1	**15.8**	-1.6	-5.8	**6.1**
Gain (relatif)	N	0	0	38.9	42.6	43.9	30.9	56.1	31.1	19.6	**47.0**	13.2	6.0	**27.4**
	Δ	0	14.3	5.9	21.12	31.5	16.12	22.65	22.7	13.5	**36.3**	-5.7	-21.7	**13.0**

TABLEAU 6.7 – Évaluation d'algorithmes KAP (Z_2) et SPbP (Z_2) Vs. M_2

l'algorithme SPbP. En effet, par exemple pour le noyau N° 12, le pourcentage de perte absolue est de moins de 6 % avec un temps de calcul beaucoup plus élevé (> 5 minutes).

Pour les onze premiers noyaux, le temps d'exécution du module KAP ainsi que celui de l'algorithme SPbP pour l'optimisation est de moins de 25 secondes, tandis que pour le dernier noyau (N° 12), le temps passé par chacun de ces deux algorithmes est supérieur à 4 minutes. Globalement, ces deux algorithmes sont équivalents en termes de temps d'exécution.

Nous allons maintenant évaluer la performance de chacune des heuristiques considérées précédemment par rapport aux bornes inférieures données au Tableau 6.3. Le nombre de pré-chargements N fourni par les deux algorithmes KAP et SPbP étant optimal pour la séquence des calculs $(s_j)_{j\in\mathcal{M}}$ déterminée à la première étape de leur processus, le Tableau 6.8 permet de comparer leurs résultats ainsi que ceux obtenus par les deux algorithmes M_1 et M_2, en termes de temps total de traitement Δ, par rapport à la borne inférieure lb_Δ. Ce rapport est calculé en utilisant la formule Υ_H donnée par l'équation 6.4. La colonne 1 donne le numéro du noyau. Les colonnes 2, 3 et 4 donnent les ratios Υ_{M_1}, Υ_{KAP} et Υ_{SPbP}, où la valeur de Z vaut Z_1. Les colonnes 5, 6 et 7 donnent les ratios Υ_{M_2}, Υ_{KAP} et Υ_{SPbP}, où la valeur de Z vaut Z_2.

Noyau N° Υ_H	Z_1			Z_2		
	Υ_{M_1}	Υ_{KAP}	Υ_{SPbP}	Υ_{M_2}	Υ_{KAP}	Υ_{SPbP}
1	**1.00**	1.36	1.02	1.36	1.36	1.36
2	**1.00**	1.16	1.08	1.24	1.16	1.16
3	1.49	1.96	1.48	1.59	2.02	1.56
4	1.89	2.23	1.65	2.03	2.33	1.81
5	1.91	1.91	1.53	2.01	2.08	1.69
6	1.60	2.11	1.59	1.90	2.23	1.76
7	-	-	-	1.37	1.68	1.29
8	2.07	2.11	1.72	2.19	2.33	1.92
9	1.42	1.72	1.29	1.44	1.81	1.38
10	1.73	1.40	1.28	1.77	1.60	1.49
11	1.34	1.98	1.38	1.40	1.99	1.42
12	1.33	2.00	1.40	1.36	2.01	1.44
Moyenne	**1.52**	**1.81**	**1.40**	**1.63**	**1.88**	**1.52**

TABLEAU 6.8 – Évaluation d'algorithmes M_1, M_2, KAP et SPbP Vs. lb_Δ

Comme le montre le tableau 6.8, dans les deux cas Z_1 et Z_2, le temps total de traitement Δ fourni par l'algorithme SPbP est en moyenne plus proche de la valeur de sa borne inférieure lb_Δ que les différentes valeurs données par chacun des algorithmes M_1, M_2 et KAP. Il est alors au plus deux fois la valeur de lb_Δ : $\Delta_{SPbP} < 2 * lb_\Delta$.

En revanche, l'algorithme SPbP donne un meilleur temps total de traitement Δ, avec une moyenne égale à $1.4 * lb_\Delta$, que KAP et M_1 dans le cas où Z vaut Z_1. Idem, dans le cas où Z vaut Z_2, avec une moyenne égale à $1.52 * lb_\Delta$.

Par exemple pour le noyau N° 10 l'heuristique SPbP donne de meilleurs résultats avec un temps Δ égal à $1.28 * lb_\Delta$ dans le cas où Z vaut Z_1, respectivement, à $1.49 * lb_\Delta$ dans le cas où Z vaut Z_2. Idem pour l'heuristique KAP, le temps Δ obtenu est au plus $1.6 * lb_\Delta$ contre $1.73 * lb_\Delta$ pour M_1 et $1.77 * lb_\Delta$ pour M_2. Cependant, pour les deux noyaux N° 11 et N° 12, les deux algorithmes M_1 et M_2 semblent plus proches de la borne inférieure sur Δ. En effet, l'algorithme M_1 est au plus $1.34 * lb_\Delta$ contre $1.4 * lb_\Delta$ pour M_2.

Remarque 6.1. *Sur les différents noyaux donnés au Tableau 6.2, les deux heuristiques KAP et SPbP ont des temps d'exécution (sans y avoir intégré la partie ATSP) très courts et très proches, jusqu'à quelques secondes pour les onze premiers noyaux. Cependant, dans le cas du noyau N° 12, elles ont pris plus de* 5 *minutes.*

6.4.3 Résultats numériques d'algorithmes ECM, CGM et CCM et leurs versions étendues

Dans cette partie, nous nous focalisons sur l'analyse des résultats obtenus en exécutant les algorithmes ECM, CGM et CCM (pour les 3 stratégies : CCM1, CCM2 et CCM3), ainsi que leurs versions étendues E-ECM, E-CGM et E-CCM (pour les 3 stratégies : E-CCM1, E-CCM2 et E-CCM3) sur les différents noyaux donnés au Tableau 6.2.

Nous rappelons que le nombre de préchargements fourni par chacune des heuristiques ECM, CGM, CCM, E-ECM, E-CGM et E-CCM, noté N^*, vaut sa borne inférieure lb_N. Plus formellement, on a : $N^*_{ECM} = N^*_{E-ECM} = N^*_{CGM} = N^*_{E-CGM} = N^*_{CCM} + N^*_{E-CCM} = lb_N$. En outre, dans le contexte des trois heuristiques ECM, CGM et CCM, le nombre de buffers utilisé vaut le nombre de tuiles préchargées (borne inférieure sur N) : $Z_{ECM} = Z_{CGM} = Z_{CCM} = X' = lb_N$. En effet, ce nombre de buffers est très supérieur à sa borne inférieure lb_Z : $Z_{ECM} = Z_{CGM} = Z_{CCM} \gg lb_Z$. En plus, dans ce cas, seul le temps total de traitement Δ doit être minimisé.

En revanche, dans le contexte des trois heuristiques E-ECM, E-CGM et E-CCM, seul le nombre de buffers Z doit être minimisé, alors que le temps total de traitement Δ a les mêmes valeurs que celles fournies par ECM, CGM et CCM qui sont utilisés comme trois sous-routines : $\Delta_{ECM} = \Delta_{E-ECM}$, $\Delta_{CGM} = \Delta_{E-CGM}$ et $\Delta_{CCM} = \Delta_{E-CCM}$.

Les Figures 6.1 et 6.2 donnent les valeurs de Δ, respectivement, celles de Z fournies par chacune des différentes heuristiques citées ci-avant.

Comme illustré dans la Figure 6.1, les résultats des deux heuristiques, ECM et CGM, sont équivalents en termes de qualité de solution. En effet, les valeurs du temps de traitement Δ fournies par les deux heuristiques semblent identiques pour la plupart des noyaux. Par contre, pour le noyau N° 10, l'heuristique ECM est meilleure que la CGM car le temps total de traitement obtenu est optimal puisqu'il est égal à sa borne inférieure lb_Δ. Idem pour les deux heuristiques E-ECM et E-CGM, les valeurs Z (Figure 6.2) semblent identiques pour la plupart des noyaux. En effet, par exemple pour le noyau N° 10, avec un nombre de tuiles d'entrée à précharger : X = 7040, l'heuristique ECM donne un temps total de traitement égal à 4548, qui est également égal à sa borne inférieure, alors que la CGM trouve 5009.

De même, l'heuristique E-ECM donne un meilleur résultat que la E-CGM avec un nombre de buffers égal à 1151, alors que la E-CGM trouve 1557. Cependant, pour les deux noyaux (N° 11 et N° 12), la E-CGM donne un meilleur résultat que la E-ECM. En effet, le nombre de buffers fourni par la E-CGM est égal à la moitié de celui obtenu par la E-ECM : $Z_{E-CGM} \simeq \frac{1}{2} Z_{E-ECM}$.

De la même manière, les deux Figures 6.3 et 6.4 donnent les valeurs de Δ fournies par l'heuristique CCM (pour les 3 stratégies CCM1, CCM2 et CCM3), respectivement, celles de Z fournies par l'heuristique E-CCM.

Comme illustré dans la Figure 6.3, pour les trois stratégies CCM1, CCM2 et CCM3, les résultats obtenus sont équivalents en termes de qualité de la solution. Pour les deux premières stratégies E-CCM1 et E-CCM2, les tests menés montrent que les différentes valeurs sont sensiblement identiques alors que la troisième stratégie E-CCM3 donne un nombre de buffers beaucoup plus élevé (Figure 6.4). Par exemple pour le noyau N° 10 les deux stratégies CCM1 et CCM2 trouvent un temps total de traitement Δ égal à 4554 contre 4662 pour la troisième stratégie CCM3. De plus, la stratégie E-CCM2 donne un meilleur nombre de buffers Z égal à 1105

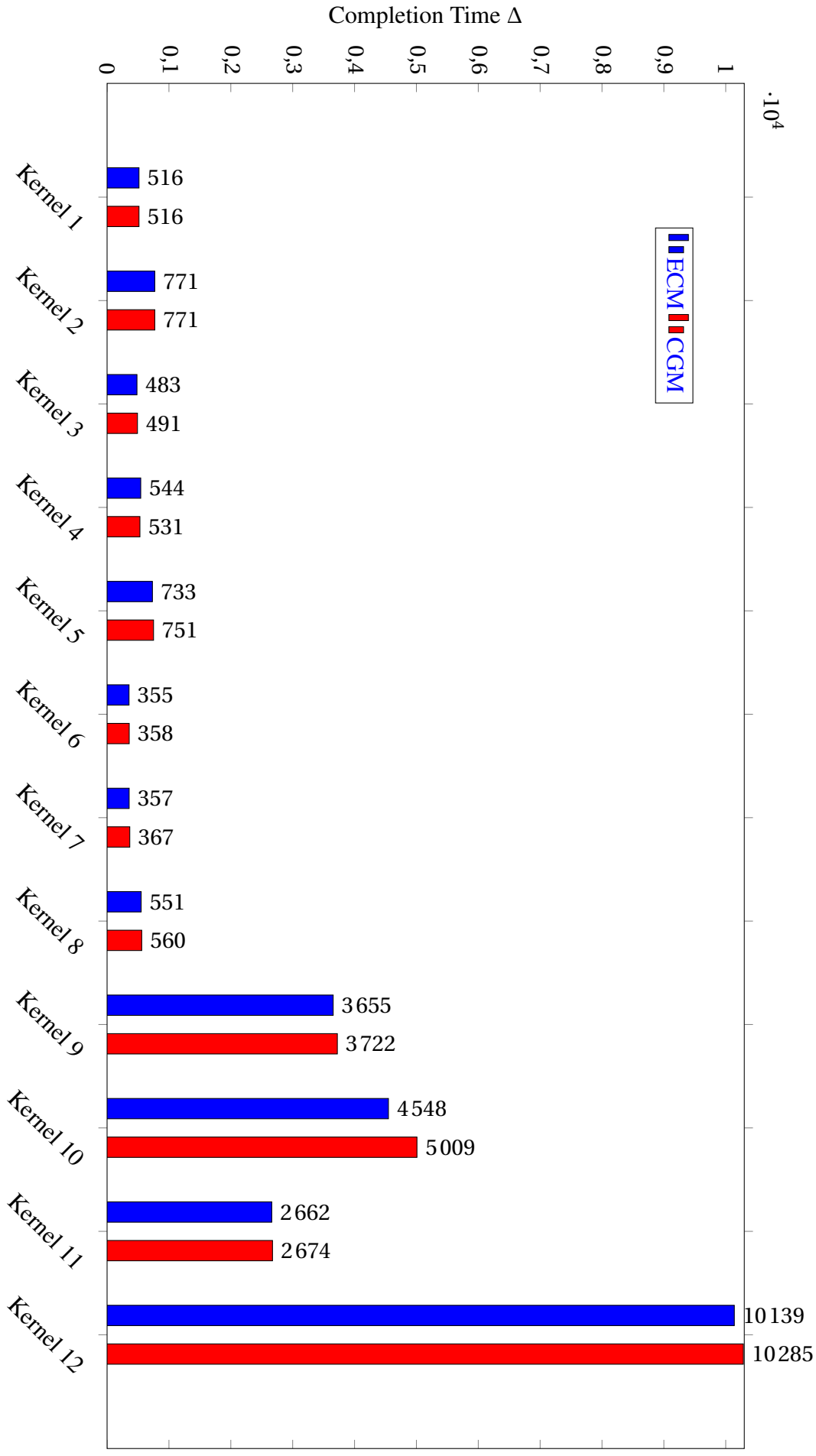


FIGURE 6.1 – Évaluation d'algorithmes ECM et CGM : résultats de Δ

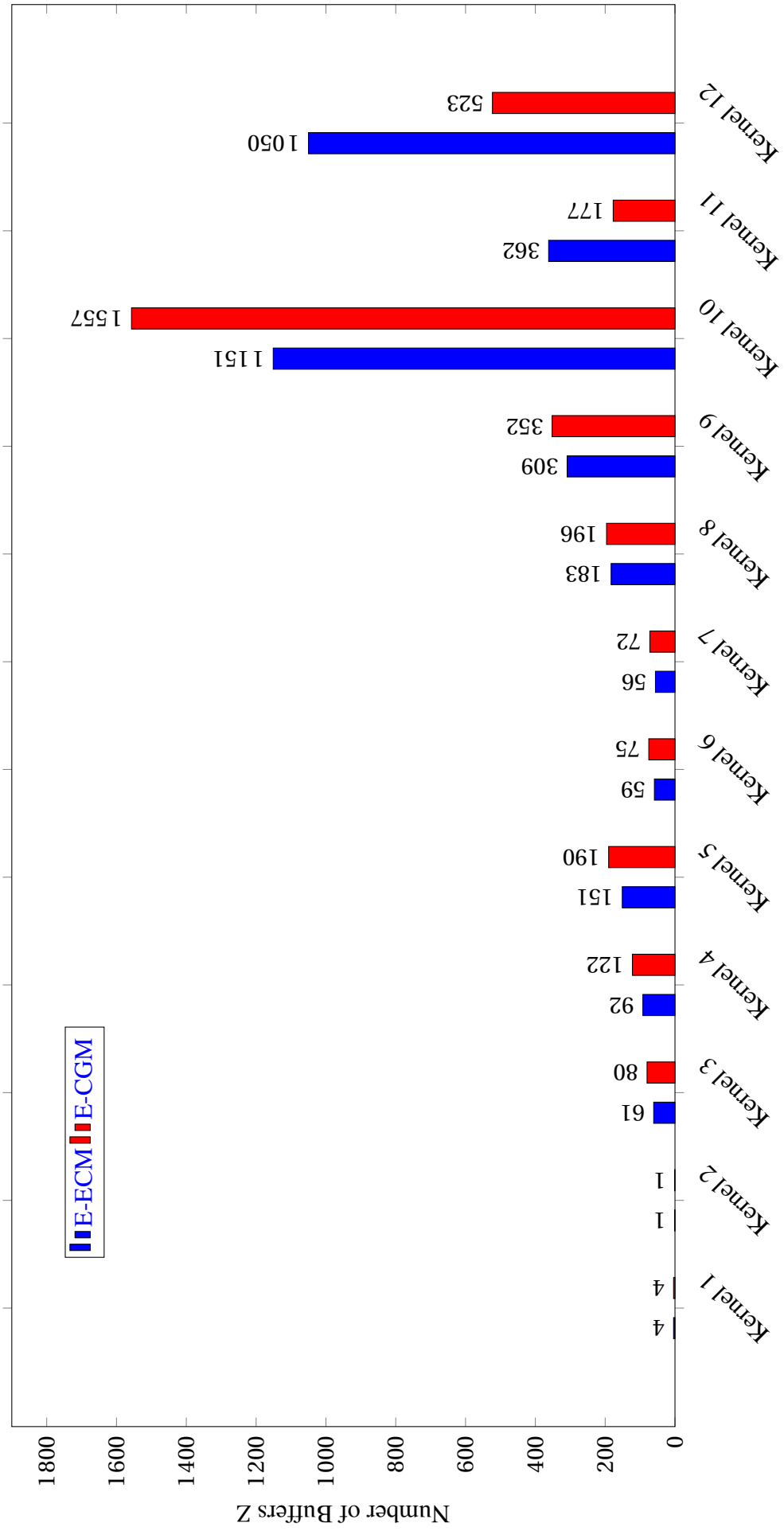


FIGURE 6.2 – Évaluation d'algorithmes E-ECM et E-CGM : résultats de Z

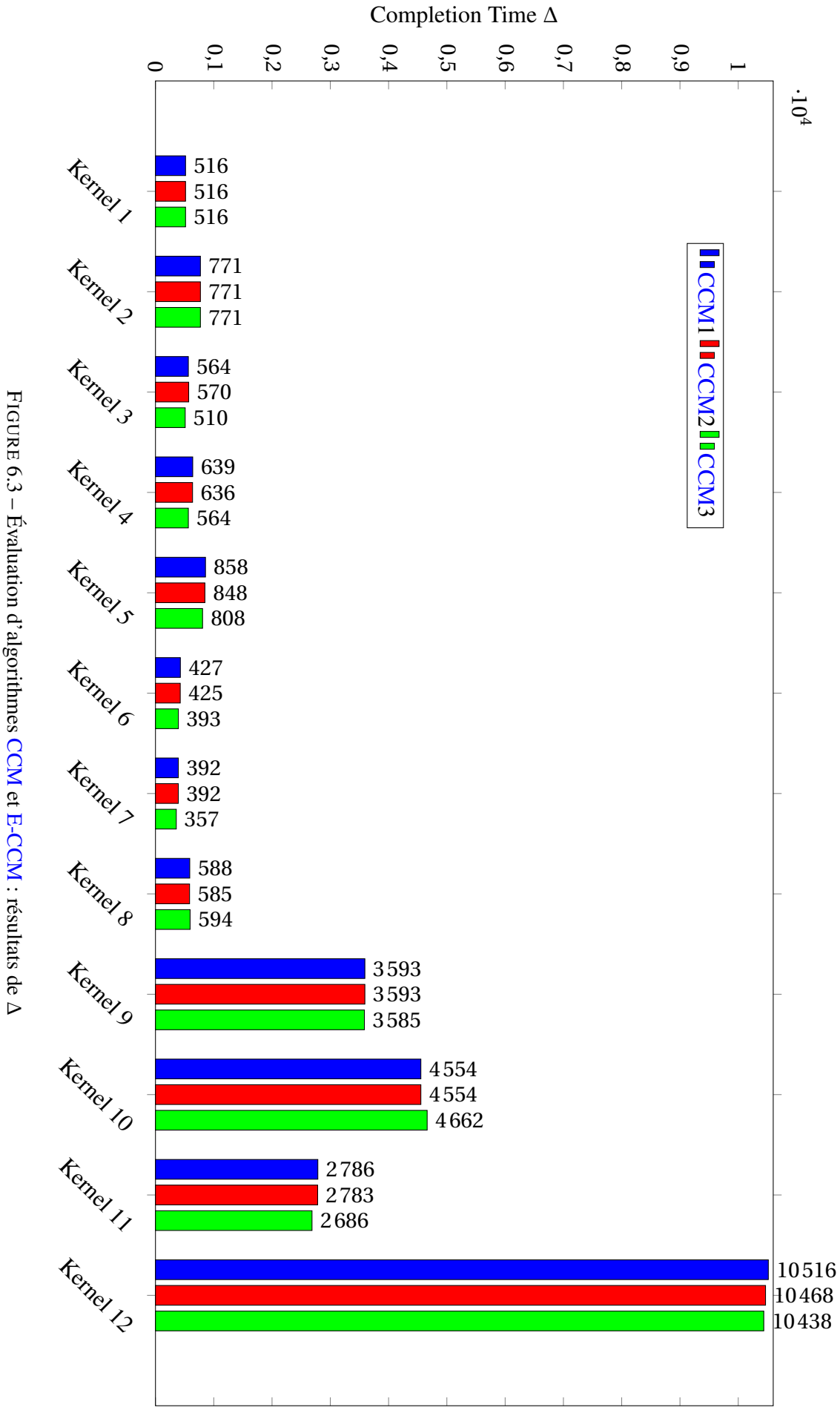


FIGURE 6.3 – Évaluation d'algorithmes CCM et E-CCM : résultats de Δ

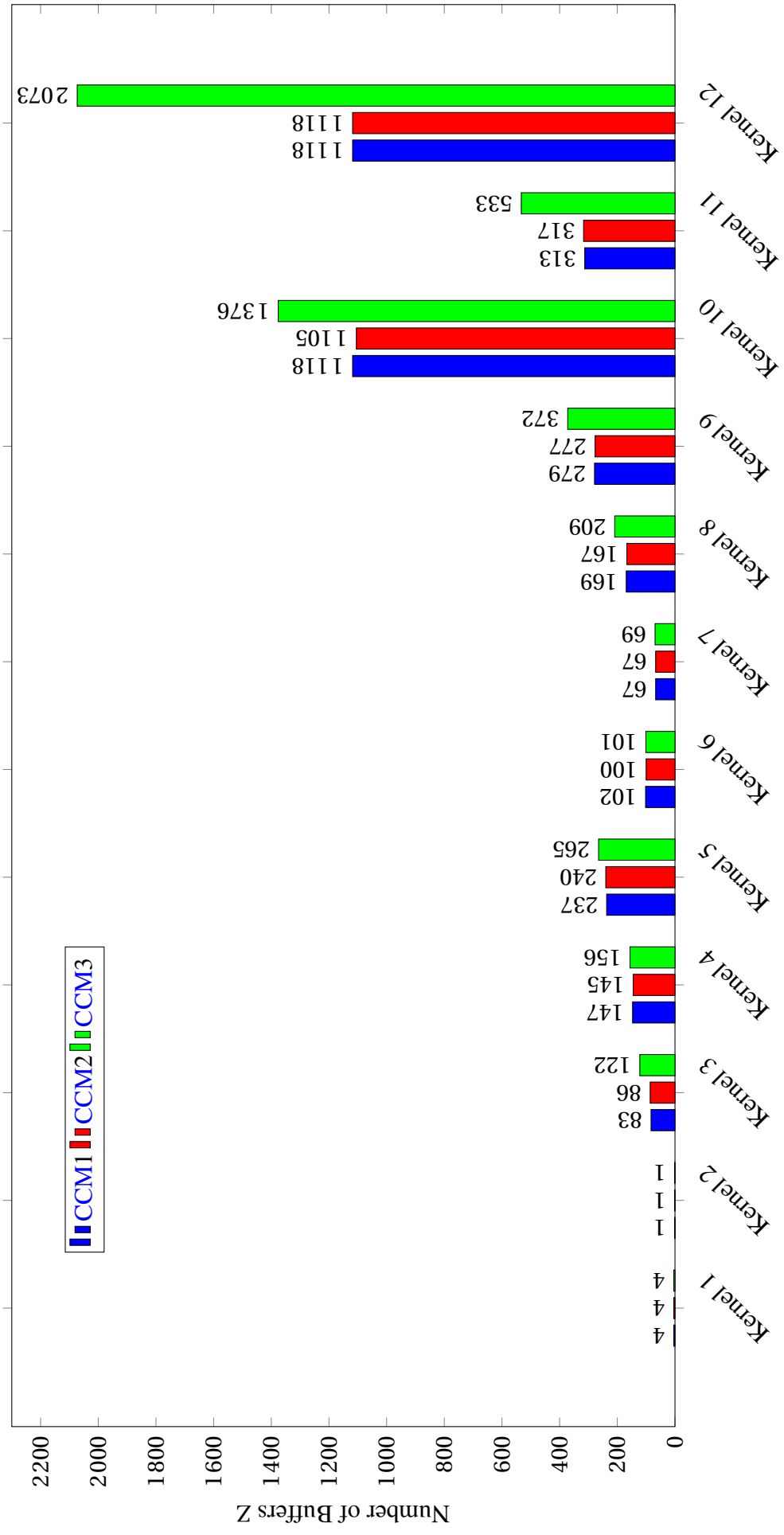


FIGURE 6.4 – Évaluation d'algorithme E-CCM : résultats de Z

contre 1118 pour la E-CCM1 et 1376 pour la E-CCM3.

Remarque 6.2. *Pour les onze premiers noyaux, les différents tests menés montrent que les trois heuristiques (ECM, CGM et CCM) sont équivalentes en termes de temps d'exécution. En effet, pour chacune de ces méthodes, ce dernier est de moins de* 10 *secondes. Cependant, dans le cas du noyau N° 12, seules les deux méthodes ECM et CGM ont pris plus de* 5 *minutes.*

Les résultats de l'évaluation des différentes heuristiques considérées dans cette partie par rapport aux bornes inférieures lb_Z et lb_Δ sont ensuite décrits dans le Tableau 6.9. Cette comparaison est établie en calculant le rapport entre les valeurs obtenues et celles de lb_Δ, respectivement, de lb_Z correspondantes. Ce rapport est calculé en utilisant la formule Υ_H donnée par l'équation 6.4. La colonne 1 donne le numéro du noyau. Les cinq premières colonnes donnent les ratios Υ_H, en termes de temps total de traitement Δ, pour les algorithmes ECM, CGM, CCM1, CCM2 et CCM3. Les cinq dernières colonnes donnent les ratios Υ_H, en termes du nombre de buffers Z, pour les algorithmes E-ECM, E-CGM, E-CCM1, E-CCM2 et E-CCM3.

Les résultats des différentes heuristiques semblent équivalents en termes de qualité de solution. En revanche, en termes de temps total de traitement Δ, la CCM3 donne de meilleurs résultats avec une moyenne égale à $1.07 * lb_\Delta$. Par contre, en termes de nombre de buffers Z, la E-CGM est meilleure avec une moyenne qui vaut $14.14 * lb_Z$. En outre, pour les noyaux de petite taille, en particulier N° 1 et N° 2, les différentes heuristiques donnent la solution optimale (pour Δ et pour Z) qui vaut sa borne inférieure.

Cependant, pour le noyau N° 10 de grande taille (en termes de nombre de tuiles d'entrée à précharger), l'heuristique ECM et la stratégie CCM1 sont les meilleures. En effet, elles donnent aussi l'optimalité en termes de temps total de traitement Δ contre un nombre de buffers Z égal à $11.98 * lb_Z$ pour la E-ECM et à $11.64 * lb_Z$ pour la E-CCM1.

6.4.4 Évaluation de l'algorithme E-SPbP sur le noyau 10

Dans cette partie, nous présentons les résultats obtenus en exécutant l'algorithme E-SPbP sur le noyau 10 qui représente la plus grande instance, avec plus de 7000 tuiles d'entrée à précharger.

Les trois Figures 6.5, 6.6 et 6.7 présentent, pour chaque paire de critères — (N,Z), (Δ,Z) et (Δ,N) — dix solutions différentes en variant le nombre de buffers dans l'intervalle $I_Z = \{Z_2, \ldots, Z_1\}$, où $Z_2 = 96$ et $Z_1 = 139$ (notez que l'exploration de l'intervalle I_Z n'a pas été entièrement exécutée). En d'autres termes, ces figures donnent trois points de vue différents sur l'ensemble des solutions afin de permettre à l'utilisateur de l'outil MMOpt de prendre la décision qui lui convient le mieux.

Comme le montrent les deux figures 6.5 et 6.6, lorsque nous augmentons le nombre de buffers Z, les valeurs de N et Δ fournies par l'algorithme E-SPbP diminuent. Par conséquent, la consommation d'énergie et le temps diminuent de la même manière par rapport à l'augmentation du paramètre de la surface du TPU produit.

Cependant, la Figure 6.7 montre l'augmentation simultanée de deux paramètres N et Δ. Comme le montre cette figure, lorsque le nombre de buffers est petit, l'algorithme E-SPbP donne un grand nombre de préchargements N ainsi qu'un large temps total de traitement Δ. En revanche, pour trouver une bonne valeur pour N et Δ, le nombre de buffers Z a la valeur la plus grande. Cela signifie qu'en faisant varier la quantité de mémoire interne (Z), de nouveaux compromis entre la consommation d'énergie (N) et le temps de calcul (Δ) peuvent être atteints.

Noyau N° Υ_H	Δ					Z				
	Υ_{ECM}	Υ_{CGM}	Υ_{CCM1}	Υ_{CCM2}	Υ_{CCM3}	$\Upsilon_{E\text{-}ECM}$	$\Upsilon_{E\text{-}CGM}$	$\Upsilon_{E\text{-}CCM1}$	$\Upsilon_{E\text{-}CCM1}$	$\Upsilon_{E\text{-}CCM3}$
1	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**
2	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**
3	1.01	1.02	1.18	1.19	1.14	6.77	8.88	9.22	9.55	13.55
4	1.13	1.10	1.33	1.32	1.17	7.07	9.38	11.30	11.15	12.00
5	1.01	1.08	1.18	1.17	1.11	7.19	9.04	11.28	11.42	12.61
6	1.04	1.04	1.25	1.24	1.15	7.37	9.37	12.75	12.5	12.62
7	1.04	1.07	1.14	1.14	1.04	4.66	6.00	5.58	5.58	5.75
8	1.11	1.13	1.19	1.18	1.20	9.15	9.80	8.45	8.35	10.45
9	1.02	1.04	1.01	1.01	1.01	23.76	27.07	21.46	21.30	28.61
10	**1.00**	1.10	**1.001**	**1.001**	1.02	11.98	16.21	11.64	11.51	14.33
11	3.84	3.90	1.05	1.05	1.01	40.22	19.66	34.77	35.22	59.22
12	1.007	1.02	1.04	1.04	1.03	105	52.3	111.8	111.8	207.3
Moyenne	**1.25**	**1.28**	**1.10**	**1.10**	**1.07**	**18.76**	**14.14**	**20.02**	**20.03**	**31.53**

TABLEAU 6.9 – Évaluation d'algorithmes ECM, CGM, CCM, E-ECM, E-CGM et E-CCM Vs. lb_Δ et lb_Z

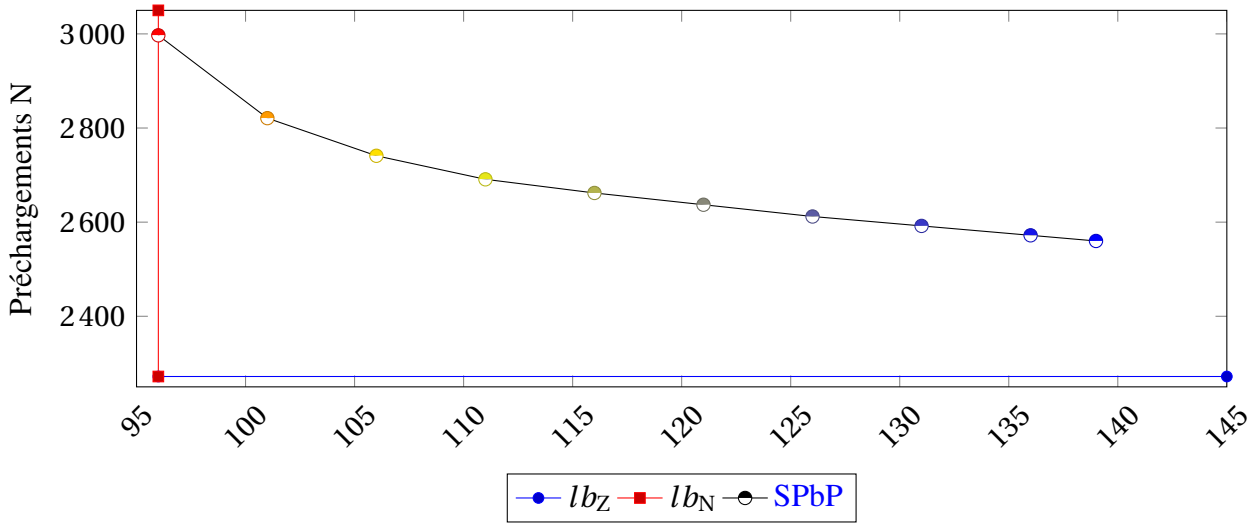


FIGURE 6.5 – Solutions de l'algorithme E-SPbP pour le noyau 10 : Énergie Vs. Surface

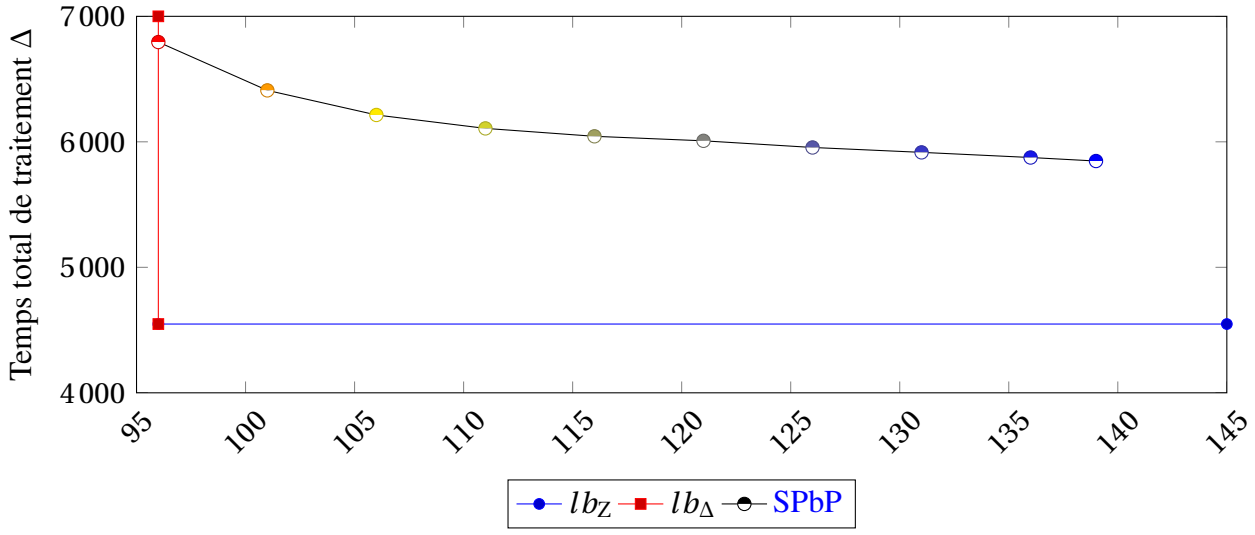


FIGURE 6.6 – Solutions de l'algorithme E-SPbP pour le noyau 10 : Temps Vs. Surface

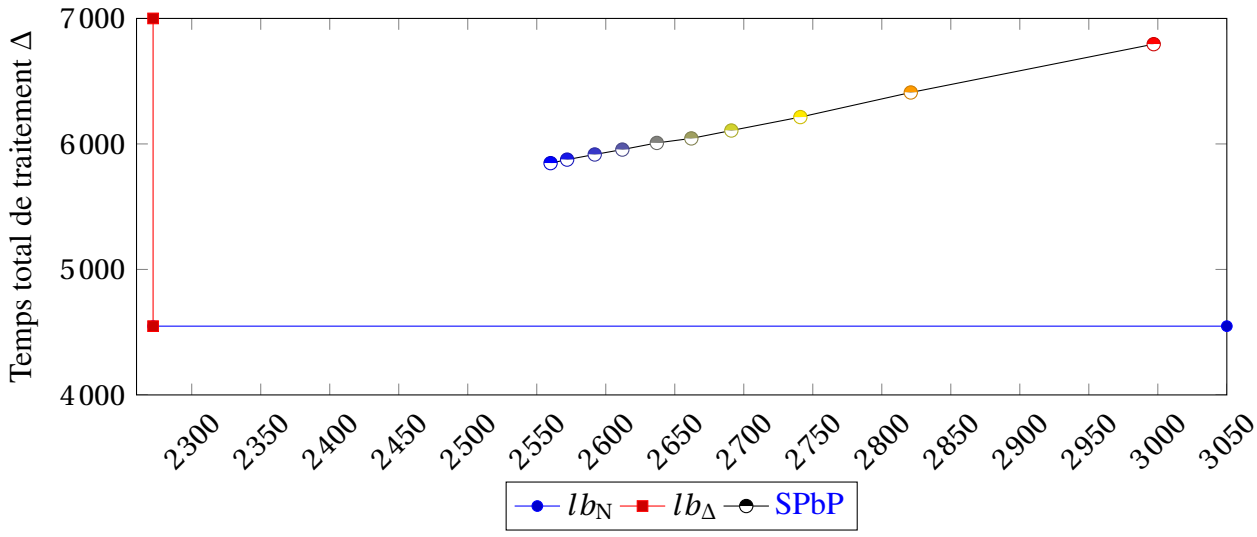


FIGURE 6.7 – Solutions de l'algorithme E-SPbP pour le noyau 10 : Temps Vs. Énergie

6.4.5 Évaluation des différentes heuristiques Vs. solutions optimales

Dans cette partie, nous présentons les résultats d'analyse afin d'évaluer la performance de nos approches par rapport aux solutions optimales fournies par la résolution des différentes formulations en PLNE (cf. Section 6.3) en utilisant les instances données au Tableau 6.1. Pour mesurer cette performance, nous utilisons la formule Θ_H donnée par l'équation 6.1.

Les résultats de cette évaluation sont décrits dans le Tableau 6.10. La colonne 1 donne le numéro du noyau. Les colonnes 3,4 et 5 donnent les valeurs de Δ obtenues par ECM, CGM et CCM (3 stratégies) ainsi que les ratios Θ_{ECM}, Θ_{CGM} et Θ_{CCM} par rapport à la solution fournie par `MCT-PLNE`.

Instance N°		**ECM**		**CGM**		**CCM1,CCM2,CCM3**	
		Δ	Θ_{ECM}	Δ	Θ_{CGM}	Δ	Θ_{CCM}
1	**1a**	37	1.09	36	1.06	(42 ;42 ;39)	(1.24 ;1.24 ;1.15)
	1b	37	1.06	37	1.06	(39 ;39 ;39)	(1.11 ;1.11 ;1.11)
	1c	47	1.24	42	1.11	(46 ;46 ;45)	(1.21 ;1.21 ;1.18)
2	**2a**	60	1.13	55	1.04	(60 ;60 ;58)	(1.13 ;1.13 ;1.09)
	2b	64	1.21	64	1.21	(65 ;65 ;64)	(1.23 ;1.23 ;1.21)
3	**3a**	79	1.14	74	1.07	(80 ;80 ;77)	(1.16 ;1.16 ;1.12)
	3b	81	1.14	75	1.06	(79 ;79 ;81)	(1.11 ;1.11 ;1.14)
	3c	83	1.19	80	1.14	(82 ;82 ;82)	(1.17 ;1.17 ;1.17)
	3d	110	1.83	101	1.68	(116 ;116 ;109)	(1.93 ;1.93 ;1.82)
	3e	116	1.93	111	1.85	(117 ;117 ;123)	(1.95 ;1.95 ;2.05)
	3f	123	2.05	121	2.02	(133 ;133 ;127)	(2.22 ;2.22 ;2.12)
4	**4a**	120	1.33	114	1.27	(122 ;122 ;117)	(1.36 ;1.36 ;1.30)
	4b	141	1.57	130	1.44	(154 ;151 ;139)	(1.71 ;1.68 ;1.54)
5	**5a**	158	1.32	153	1.27	(166 ;166 ;163)	(1.38 ;1.38 ;1.36)
	5b	207	1.73	195	1.62	(208 ;208 ;201)	(1.73 ;1.73 ;1.68)
6		210	1.40	206	1.37	(215 ;215 ;216)	(1.43 ;1.43 ;1.44)
Moyenne		-	**1.39**	-	**1.32**	-	(**1.44** ;**1.44** ;**1.40**)

TABLEAU 6.10 – Évaluation des heuristiques Vs. solutions optimales

Le Tableau 6.10 permet de comparer les résultats des N et Δ obtenus avec les différentes heuristiques utilisées par rapport aux solutions optimales obtenues par les deux méthodes exactes PSP_a et MCT_a. En moyenne, le ratio Θ_H est de moins de 1.5 par rapport à la solution optimale fournie par la MCT_a. En outre, la méthode CGM donne de meilleurs résultats pour tous les noyaux, avec une moyenne égale à $1.32 * OPT$. Par contre, la troisième stratégie CCM3 est meilleure que la CCM1 et CCM2. En effet, la performance de CCM3 est en moyenne égale à $1.4 * OPT$ contre $1.44 * OPT$ pour celui de CCM1 et CCM2.

6.5 Synthèse des résultats expérimentaux des méthodes

Dans cette section, nous présentons d'abord une synthèse sur l'étude empirique établie et ensuite un bilan d'analyse critique qui guidera l'utilisateur de l'outil MMOpt dans le choix de sa meilleure méthode de résolution.

6.5.1 Brève synthèse sur les tests menés

L'analyse des résultats numériques que nous avons obtenus pour les deux familles d'instances utilisées (Tableaux 6.1 et 6.2) nous a permis de mieux cerner l'impact positif ou négatif des différentes approches proposées en fonction de l'instance à résoudre.

D'une manière générale, les méthodes exactes garantissent l'obtention d'une solution optimale au problème. Dans le contexte des différents problèmes abordés dans cette thèse, les méthodes exactes que nous avons proposé en termes de PLNE ne peuvent pas résoudre les différentes instances utilisées. En effet, elles sont capables de résoudre en temps raisonnable et de manière efficace les instances de petite taille, c'est-à-dire jusqu'à 20 tuiles de sortie à calculer et jusqu'à 20 tuiles d'entrée à précharger. Cependant, leur performance reste encore limitée pour les instances de grande et moyenne taille comme celles fournies par Mancini et al. [89, 90] (décrits dans le Tableau 6.2).

Par ailleurs, les tests numériques menés sur les différents noyaux, comprenant jusqu'à 7040 tuiles d'entrée en tout et un maximum de 3353 tuiles de sortie à calculer, montrent clairement l'efficacité des différentes approches heuristiques proposées. En effet, les deux méthodes KAP et SPbP apportent en moyenne, dans l'objectif de minimiser la consommation d'énergie (le trafic depuis la mémoire externe ou encore le nombre de préchargements), des gains absolus de performance supérieurs à 15 % comparativement aux M_1 et M_2 (en utilisant les mêmes ressources mémoires locales : Z_1 et Z_2 buffers). De même, dans l'objectif d'avoir la meilleure performance des TPUs produits, l'algorithme SPbP permet de réduire le temps total de traitement d'une réduction absolue supérieure à 5 %.

En revanche, dans le contexte des heuristiques ECM, CGM et CCM ainsi que leurs versions étendues E-ECM, E-CGM et E-CCM, les résultats obtenus sont généralement équivalents en termes de qualité de solution et de temps d'exécution. De plus, les tests montrent que les valeurs obtenues pour les différentes critères d'optimisation (N, Z, Δ) sont sensiblement identiques.

6.5.2 Discussion et analyse critique des méthodes proposées

Compte tenu du nombre important d'approches proposées pour résoudre le problème d'origine 3-PSDPP ainsi que certaines de ses variantes, nous nous intéressons dans cette partie à la phase d'analyse critique de ces méthodes dans le but qu'elle facilite la prise de décision.

La première question qui se pose ici est : *est-il techniquement faisable d'intégrer les méthodes exactes en termes de (PLNE) dans l'outil MMOpt ?*

On sait qu'il est souvent impossible de proposer des algorithmes exacts suffisamment efficaces pour résoudre les instances de grande taille. Or les jeux de données fournies par Mancini et al. [89, 90] (cf. Tableau 6.2) le sont. Sur la base de différents tests menés, nous pouvons dire que les trois PLNEs proposées, dans leurs formulations actuelles, sont réputés inefficaces et ne peuvent pas être utilisées en tant que méthodes de résolution exacte pour optimiser le fonctionnement de MMOpt. En revanche, de nombreuses pistes (les contraintes de rupture de symétrie, les coupes, etc.) pourraient également être envisagées pour améliorer leurs performances.

La deuxième question qui se pose ici est : *parmi les nombreuses heuristiques proposées ici pour résoudre les différents problèmes d'optimisation abordés, laquelle est la meilleure, en*

termes de qualité de solution, et laquelle est la plus performante pour l'utilisateur de MMOpt ?

Face aux souhaits divers de l'utilisateur de l'outil MMOpt, il est impossible de parler de solution unique. En effet, s'il désire optimiser un seul critère à la fois, il peut avoir recours :

- Aux algorithmes polynomiaux et optimaux décrits à la Section 4.5.1 du Chapitre 4. En effet, pour avoir un nombre de préchargements optimal N^*, il peut simplement choisir l'algorithme décrit à la Preuve 5.2 (solution du problème MP-PSDPP) ou utiliser l'algorithme KAD, dans le cas où le nombre de buffers est fixé en entrée (solution du problème DPP). De même, pour avoir un nombre de buffers optimal Z^*, il peut simplement utiliser l'algorithme décrit à la Preuve 5.1 (solution du problème MB-PSDPP).
- Aux algorithmes polynomiaux décrits à la Section 5.5.2 du Chapitre 5, qui produisent généralement de bons résultats en termes de qualité de solution et de temps d'exécution. En effet, s'il cherche la faible énergie (nombre de préchargement N), il peut utiliser l'algorithme KAP (solution du problème PSP présentée à la Section 5.5.2.1). Par contre, s'il cherche la haute performance du circuit produit, il peut choisir l'algorithme qui le satisfait le plus parmi les cinq suivants : ECM, CGM, CCM1, CCM2 et CCM3 (solutions, pour les problèmes MCT-PSDPP et B-C-MCTP, présentées à la Section 5.5.2.2).

Toutefois, s'il veut optimiser deux critères simultanément, c'est-à-dire résoudre le problème bi-objectif 2-PSDPP, il aura recours à SPbP, notre unique algorithme proposé dans ce contexte (cf. Section 5.5.3 du Chapitre 5).

Finalement, s'il veut optimiser les trois critères simultanément, c'est-à-dire résoudre le problème d'origine 3-PSDPP, il aura plutôt recours aux algorithmes E-ECM, E-CGM, E-CCM et E-SPbP (cf. Section 5.5.4 du Chapitre 5), où de nouveaux compromis entre la zone de mémoire interne (buffers Z), le temps de calcul (Δ) et la consommation d'énergie (N) peuvent être atteints.

En ce qui concerne la validation de nos approches dans un environnement Field Programmable Gate Array (FPGA), nous n'avons pas pu réaliser cette phase. Elle reste ainsi la perspective applicative de cette thèse.

Pour conclure cette partie, nous pouvons dire que l'évaluation numérique menée et l'analyse critique qui suit présentent une phase fondamentale pour aider l'utilisateur de l'outil MMOpt afin de choisir la solution qui répond le mieux à ses souhaits.

6.6 Conclusion

Dans ce dernier chapitre, la performance de nos approches exactes et approchées (présentées au Chapitre 5) a été évaluée dans une série d'expériences. En effet, pour effectuer les différentes expérimentations, nous avons utilisé des jeux de données composés de deux familles : des instances connues dans la littérature du problème de ToSP ainsi que des jeux de données réelles fournis par Mancini et al. [89, 90].

Les résultats obtenus sur cet ensemble d'instances ont été présentés et analysés selon trois axes d'évaluation : solutions optimales, bornes inférieures et méthodes existantes. Ils montrent clairement la performance de nos algorithmes, en particulier les méthodes approchées dans le contexte d'instances de grande taille. Les solutions obtenues sont alors encourageantes et présentent une piste prometteuse pour optimiser les performances des hiérarchies mémoires produites par le générateur.

En revanche, les méthodes exactes basées sur la formulation en PLNE restent inefficaces dans le contexte de l'outil MMOpt en tant qu'application concrète pour les systèmes de vision embarquée. Un certain nombre d'améliorations (comme par exemple en ajoutant des coupes

efficientes), pourraient alors être apportées aux différentes formulations. Il semble encore possible d'accélérer le temps de calcul pour résoudre les grandes instances. Par la suite, une étude pour affirmer l'efficacité de nos méthodes sera évidente.

Conclusion et Perspectives

Conclusion

Les travaux que nous avons présentés dans ce manuscrit traitent des problèmes d'optimisation qui découlent de la vision embarquée avec le point de vue de la recherche opérationnelle. Ils sont portés particulièrement sur la mise en œuvre des méthodes issues de la recherche opérationnelle pour la conception de circuits numériques dans le domaine du traitement du signal et de l'image. Ce travail de thèse s'est intéressé à la phase d'optimisation visant le fonctionnement efficace des TPUs générés par l'outil MMOpt.

Le mariage entre ces deux domaines donne lieu à une problématique d'optimisation riche. Il s'agit d'un problème d'ordonnancement multi-objectif, dénommé 3-PSDPP. Cette problématique peut se décrire comme suit : étant donné une image d'entrée découpée en un ensemble de tuiles de taille donnée, une image de sortie formée d'un ensemble de tuiles de sortie à calculer, où le calcul d'une tuile de sortie nécessite en général plusieurs tuiles de l'image d'entrée. Ces tuiles devant être accessibles dans les buffers au moment de ce calcul. Le problème 3-PSDPP consiste à déterminer un séquencement des préchargements de tuiles de l'image d'entrée et leurs emplacements dans les mémoires locales (buffers), et un séquencement des calculs des tuiles de sortie afin d'optimiser simultanément les trois enjeux électroniques classiques : performance, consommation d'énergie et coût de production (surface du circuit).

Après une brève introduction, nous avons donné, dans la première partie (chapitres 1 et 2), un aperçu sur les concepts nécessaires à la compréhension des chapitres suivants concernant le domaine de la vision embarquée, celui de la conception des circuits intégrés et celui de la recherche opérationnelle. Nous avons également décrit l'outil MMOpt qui a fait l'objet de cette thèse en mettant l'accent sur la problématique d'optimisation engendrée.

Les parties 2 et 3 sont consacrées à la description de notre contribution qui porte sur la modélisation et la résolution des différents problèmes d'optimisation qui se présentent au sein de cette problématique. À cet égard, notre contribution a été essentiellement axée sur les quatre volets suivants : Modéliser, Comparer, Résoudre et Valider.

Nous nous sommes intéressés dans le troisième chapitre à la phase modélisation afin de caractériser formellement les problèmes d'optimisation abordés. Cette phase a été l'étape la plus cruciale dans cette étude. Ainsi, l'effort que nous avons fourni, en particulier pour la formalisation de notre problème 3-PSDPP, a été important dans le sens où il nous a permis de traduire la description naturelle du problème réel en un modèle mathématique avec des données clairement définies. Cette phase a également été complexe dans le sens où beaucoup d'échanges ont été nécessaires avec le concepteur de l'outil MMOpt afin de comprendre le problème correctement et de fixer les objectifs à atteindre ainsi que les contraintes à respecter. Après plusieurs tentatives de modélisation, nous avons formulé le problème abordé en utilisant un modèle mathématique avec des données d'entrée, des variables à déterminer, des contraintes et des objectifs à minimiser pour le problème d'origine 3-PSDPP ainsi qu'une formulation en terme de programmes linéaires en nombres entiers pour chacun de ces sous-problèmes mono-objectif dérivés.

Notons que ce problème n'a fait l'objet d'aucune étude préalable dans la littérature de la recherche opérationnelle. Le quatrième chapitre décrit une revue de l'état de l'art et tente de couvrir aussi largement que possible les divers problèmes voisins trouvés dans la littérature ainsi que les travaux antérieurs proposés par Mancini et al. [89]. Cette phase a permis de positionner notre problème d'origine 3-PSDPP par rapport à ceux qui existent dans la littérature. Elle a également facilité la phase d'étude de complexité de certaines variantes dérivées qui nous a également conduit à choisir les méthodes de résolution les mieux adaptées à chacun de ces problèmes.

Arrivé au terme du cinquième chapitre, le lecteur serait face à un large ensemble d'approches de résolution. Comme il a été difficile de résoudre le problème 3-PSDPP, nous nous sommes intéressés à la résolution de chacun des sous-problèmes abordés. Notons que la majorité de ces problèmes sont $\mathscr{NP}$-Complets. Il est donc souvent impossible de proposer des algorithmes exacts suffisamment efficaces pour résoudre les instances de grande taille. Par conséquent, il a alors été nécessaire de s'intéresser à des méthodes heuristiques. La variété des approches proposées nous a ainsi permis de fournir des solutions rapides et de très bonnes qualités à l'utilisateur de l'outil MMOpt.

Dans le chapitre 6, les expérimentations numériques menées sur des benchmarks et des données réelles ont montré que les différentes approches proposées, en particuliers les heuristiques constructives, fournissent de bons résultats par rapport à ceux obtenus par l'algorithme actuellement utilisé dans l'outil MMOpt. En effet, les solutions retournées sont de très bonne qualité, même pour des instances de grande taille. Arrivé au terme de ce chapitre, notre objectif est de fournir un ensemble de bonnes solutions de compromis possibles entre les trois critères : consommation d'énergie, performance et surface.

Les différents travaux et résultats menés dans notre thèse ont donné lieu à des publications dans des conférences internationales ([62], [78], [58], [59] et [60]) et nationales ([63, 63, 65] et [64]) ainsi que dans le livre "Recent Advances in Computational Optimization" ([61]).

Un atout majeur de notre travail de recherche est que face aux différents besoins des concepteurs des circuits intégrés pour la vision embarquée, la recherche opérationnelle présente la bonne démarche et la méthodologie la plus adaptée à utiliser pour apporter une aide à la prise de décision afin d'optimiser le fonctionnement des TPUs générés par l'outil MMOpt. Cette démarche contribue également à donner aux concepteurs de l'outil MMOpt (Mancini et al.) une vision différente en termes de méthodologie et d'outils utilisés.

Perspectives

Le travail réalisé dans cette thèse permet d'envisager de nombreuses perspectives. Nous les présentons en cinq points :

— Améliorer la résolution du problème d'origine 3-PSDPP ainsi que ses variantes dérivées (mono- et bi-objectif) :

  - Utiliser des méthodes méta-heuristiques et/ou des algorithmes d'approximation. Dans la continuité directe de ce travail de thèse, il nous semble intéressant de développer un algorithme génétique pour résoudre le problème d'origine 3-PSDPP.
  - Tenter d'améliorer la résolution de différentes formulations en terme de PLNE en utilisant d'autres solveurs tels que LocalSolver, un solveur méta-heuristique basé sur la recherche locale.
  - Tenter d'améliorer les heuristiques proposées dans le chapitre 5.

— Relever le défi d'analyser la complexité de certaines variantes de problème d'origine 3-PSDPP, qui ne sont pas encore abordées telles que P-C-MBP et P-C-MCTP.

— Tenter d'aborder les problèmes considérés dans ce travail, d'un point de vue différent en termes de notation, modélisation et résolution, tout en le considérant peut-être comme des problèmes d'ordonnancement classiques, à savoir le Flow-shop, le Job-shop, l'Open-shop ou encore le problème de gestion de projet à contraintes de ressources Resource Constrained Project Scheduling Problem (RCPSP).

— Adapter les méthodes proposées dans le contexte du problème ToSP.

— Valider dans un environnement FPGA l'un des algorithmes proposés. Cette phase reste la plus importante pour garantir la performance réelle des méthodes proposées sur des traitements typiques des systèmes de vision (autofocus, fisheye, appariement, stéréo-vision, reconstruction 3D, etc).

Acronymes

2-PSDPP 2-objective Process Scheduling and Data Prefetching Problem. 51, 86, 93, 98, 99, 101, 131

3-PSDPP 3-objective Process Scheduling and Data Prefetching Problem. xii, xiii, 14, 15, 43, 45–51, 60, 62, 64, 65, 70–73, 75, 78, 80, 81, 85, 86, 89, 90, 93, 100, 101, 103, 105, 107, 108, 116, 130, 131, 133–135

AD aide à la décision. 18

ATSP Asymetric Traveling Salesman Problem. 62, 98, 100, 104, 108, 121

B&B Branch and Bound. 33, 34, 86

B-C-MCTP Buffer-Constrained Minimum Completion Time Problem. 51, 53, 56, 58, 60, 74, 78–80, 85, 86, 90, 93, 96, 98, 101, 131

CCI conception des circuits intégrés. 18

CCM Computation Classes for MCT. 89, 96–98, 101, 103, 104, 121, 124–127, 129–131

CGM Computation Grouping for MCT. 89, 93–98, 101, 102, 104, 121, 122, 126, 127, 129–131

CPU Central Processins Unit. 6–8, 117

CTSYS Conception et Test de Systèmes embarqués. xi

DPP Data Prefetching Problem. 50, 70, 71, 73–75, 91, 100, 108, 131

DRAM Dynamic Random Access Memory. 7, 9

E-CCM Extended CCM for 3-PSDPP. 101, 103, 121, 124–127, 130, 131

E-CGM Extended CGM for 3-PSDPP. 101, 102, 121, 123, 126, 127, 130, 131

ECM Earliest Computations for MCT. 89, 93, 94, 98, 101, 102, 104, 105, 121, 122, 126, 127, 129–131

E-ECM Extended ECM for 3-PSDPP. 101, 102, 104, 121, 123, 126, 127, 130, 131

EHP Edge Hamiltonien Path. 78–80

ES Embedded System. 4

E-SPbP Extended SPbP for 3-PSDPP. 101, 126, 128, 131

FC Forward Checking. 23

FMS Flexible Manufacturing System. 65, 70

FPGA Field Programmable Gate Array. 131, 135

GAT Generate-And-Test. 23

HLS High Level Synthesis. 11, 12, 14

IPMTC Identical Parallel Machines Problem with Tooling Constraints. 70

ITB Input Tile Buffers. 43

KAD KTNS Adapted to DPP. 74, 91, 93, 98–100, 104, 131

KAP KTNS Adapted to PSP. 89, 90, 93, 98–100, 104, 117–121, 130, 131

KTNS Keep Tool Needed Soonest. 67, 68, 70, 74, 91–93, 100

LB Lower Bound. 72

LCIS Laboratoire de Conception et d'Intégration des Systèmes. xi

MAC Maintaining Arc Consistency. 23

MB-PSDPP Minimum Buffers of 3-PSDPP Problem. 50, 73, 131

MCT-PSDPP Minimum Completion Time of 3-PSDPP Problem. 50, 53, 55, 56, 74–77, 85, 86, 90, 93, 94, 96, 101, 114, 115, 131

MMOpt Memory Management Optimization. xi–xiii, 4, 9, 11–14, 18, 39, 43, 45, 50, 60, 62, 64, 70, 81, 85, 86, 101, 105, 107, 116, 117, 126, 130, 131, 133, 134

MO-TSP Multi-Objective Tool Selection Problem. 67

MP-PSDPP Minimum Prefetches of 3-PSDPP Problem. 50, 73, 131

MTSIP Minimum Tool Switching Instants Problem. 67

MTSP The Modular Tool Switching Problem. 67

MW Memory Wall. xii, 4, 50

OC Optimisation Combinatoire. 17

OCM Optimisation Combinatoire Multi-Objectif. 17

OMO Optimisation Multi-Objectif. 20, 36–39

OR Operationnal Research. 17

PC Personnal Computer. 4, 108

P-C-MBP Prefetch-Constrained Minimum Buffers Problem. 51, 135

P-C-MCTP Prefetch-Constrained Minimum Completion Time Problem. 51, 135

PE Processing Engine. 12, 13

PL programmation linéaire. 18, 19

PLM Programmation Linéaire en Variables Mixtes. 20

PLNE Programmation Linéaire en Nombre Entiers. 20, 34, 43, 45, 46, 53, 60, 68, 69, 85–89, 105, 108, 112, 129–131, 134

PMO problème d'optimisation multi-objectif. 36, 37

PO Pareto Optimal. 37

PPC programmation par contraintes. 18, 23

PSC problème de satisfaction de contraintes. 23

PSP Prefetching and Scheduling Problem. 50, 51, 53, 60, 70, 71, 74, 75, 85, 86, 88, 90, 91, 93, 108, 112–114, 131

PU Prefetching Unit. 12, 13

PVC Problème du Voyageur de Commerce. 62

PVCA Problème du Voyageur de Commerce Asymétrique. 62

RAM Random Access Memory. 7, 8, 108

RCPSP Resource Constrained Project Scheduling Problem. 135

RO Recherche Opérationnelle. 14, 15, 17–19, 21, 23, 29, 32, 36, 39, 43, 45, 60, 62, 64–66, 70, 76, 81

ROM Read Only Memory. 4

RTL Register Transfer Level. 12

SE Système Embarqué. 4, 5

SPbP Shifted Prefetches for bi-PSDPP. 89, 93, 98–101, 104, 105, 117–121, 128, 130, 131

SRA Simple Retour Arrière. 23

TC théorie de complexité. 18, 24, 25

TG théorie des graphes. 18, 20, 21

TIMA Techniques de l'Informatique et de la Micro-électronique pour l'Architecture des systèmes intégrés. xi

ToSP Tool Switching Problem. 66–71, 74, 75, 91, 108, 131, 135

TP Tooling Problem. 67, 70, 71, 74, 75, 91, 108

TPU Tile Processing Unit. xii, 11–14, 43, 49, 50, 60, 62, 105, 126, 130, 133, 134

TSP Traveling Salesman Problem. 62, 68, 69, 108

VE vision embarquée. 18

Bibliographie

[1] M.S. Akturk, J.B. Ghosh, and E.D. Gunes. Scheduling with tool changes to minimize total completion time : a study of heuristics and their performance. *Naval Research Logistics (NRL)*, 50(1) :15–30, 2003. 70

[2] M.S. Akturk, J.B. Ghosh, and E.D. Gunes. Scheduling with tool changes to minimize total completion time : Basic results and spt performance. *European Journal of Operational Research*, 157(3) :784–790, 2004. 70

[3] M.S. Akturk, J.B. Ghosh, and R.K. Kayan. Scheduling with tool changes to minimize total completion time under controllable machining conditions. *Computers & operations research*, 34(7) :2130–2146, 2007. 70

[4] M.A. Al-Fawzan and Al-Sultan K.S. A tabu search based algorithm for minimizing the number of tool switches on a flexible machine. *Computers & Industrial Engineering*, 44(1) :35–47, 2003. x, 108, 109

[5] S. Albers. New results on web caching with request reordering. In *Proceedings of the sixteenth annual ACM symposium on Parallelism in algorithms and architectures*, pages 84–92. ACM, 2004. 70

[6] J.M. Aldous and R.J. Wilson. *Graphs and applications : an introductory approach*, volume 1. Springer Science & Business Media, 2003. 21

[7] A. Alj and R. Faure. *Guide de la recherche opérationnelle*. Masson, 1990. 39

[8] J.E. Amaya, C. Cotta, and A.J. Fernández. A memetic algorithm for the tool switching problem. In *Hybrid metaheuristics*, pages 190–202. Springer, 2008. 69

[9] J.E. Amaya, C. Cotta, and A.J. Fernández. Hybrid cooperation models for the tool switching problem. In *Nature Inspired Cooperative Strategies for Optimization (NICSO 2010)*, pages 39–52. Springer, 2010. 69

[10] D. Applegate, R. Bixby, W. Cook, and V. Chvátal. *On the solution of traveling salesman problems*. Rheinische Friedrich-Wilhelms-Universität Bonn, 1998. 108

[11] D.L. Applegate, R.E. Bixby, V. Chvátal, and W.J. Cook. *The Traveling Salesman Problem : a computational study (Princeton Series in Applied Mathematics)*. Princeton university press, 2007. 63

[12] K. Apt. *Principles of constraint programming*. Cambridge University Press, 2003. 23

[13] C. Arbib, M. Flammini, and E. Nardelli. How to survive while visiting a graph. *Discrete applied mathematics*, 99(1-3) :279–293, 2000. 77

[14] J.F. Bard. A heuristic for minimizing the number of tool switches on a flexible machine. *IIE Transactions*, 20(4) :382–391, 1988. x, 68, 108, 109

[15] A.C. Beezão, J.F. Cordeau, G. Laporte, and H.H. Yanasse. Scheduling identical parallel machines with tooling constraints. *European Journal of Operational Research*, 257(3) :834–844, 2017. 70

[16] L.A. Belady. A study of replacement algorithms for a virtual-storage computer. *IBM Systems journal*, 5(2) :78–101, 1966. 69

[17] N. Bellas, S.M. Chai, M. Dwyer, and D. Linzmeier. Real-time fisheye lens distortion correction using automatically generated streaming accelerators. In *17th IEEE Symposium on Field Programmable Custom Computing Machines, FCCM'09.*, pages 149–156, 2009. 109

[18] H.P. Benson. Existence of efficient solutions for vector maximization problems. *Journal of Optimization Theory and Applications*, 26(4) :569–580, 1978. 38

[19] C. Berge. *Théorie des graphes et ses applications*. Dunod-Paris, 1958. 21

[20] A.A. Bertossi. The edge hamiltonian path problem is NP-complete. *Information Processing Letters*, 13(4) :157–159, 1981. 68, 79

[21] D. Catanzaro, L. Gouveia, and M. Labbé. Improved integer linear programming formulations for the job sequencing and tool switching problem. *European Journal of Operational Research*, 244(3) :766–777, 2015. 69

[22] J.S. Chen. Optimization models for the tool change scheduling problem. *Omega*, 36(5) :888–894, 2008. 70

[23] K. Chitnis, R. Staszewski, and G. Agarwal. Ti vision sdk, optimized vision libraries for adas systems (white paper). In *Texas Instruments*, 2014. 6

[24] M. Chrobak, H. Karloof, T. Payne, and S. Vishwnathan. New results on server problems. *SIAM Journal on Discrete Mathematics*, 4(2) :172–181, 1991. 69

[25] J. Clausen. Branch and bound algorithms-principles and examples. *Department of Computer Science, University of Copenhagen*, pages 1–30, 1999. 33

[26] Y. Collette and P. Siarry. *Optimisation multiobjectif : Algorithmes*. Editions Eyrolles, 2011. 39

[27] A. Colorni, M. Dorigo, V. Maniezzo, et al. Distributed optimization by ant colonies. In *Proceedings of the first European conference on artificial life*, volume 142, pages 134–142. Paris, France, 1991. 36

[28] S.A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the third annual ACM symposium on Theory of computing*, pages 151–158. ACM, 1971. 30

[29] Y. Crama, A.W.J. Kolen, A.G. Oerlemans, and F.C.R. Spieksma. Minimizing the number of tool switches on a flexible machine. *International Journal of Flexible Manufacturing Systems*, 6(1) :33–54, 1994. 68, 90

[30] Y. Crama, L.S. Moonen, F.C.R. Spieksma, and E. Talloen. The tool switching problem revisited. *European Journal of Operational Research*, 182(2) :952–957, 2007. 67, 70

[31] G.B. Dantzig and P. Wolfe. Decomposition principle for linear programs. *Operations research*, 8(1) :101–111, 1960. 33

[32] K. Deb. *Multi-objective Optimization Using Evolutionary Algorithms*, volume 16. John Wiley & Sons, 2001. 38

[33] M. Denizel. Minimization of the number of tool magazine setups on automated machines : A lagrangean decomposition approach. *Operations Research*, 51(2) :309–320, 2003. 69

[34] G. Desaulniers, J. Desrosiers, and Ma.M. Solomon. *Column generation*, volume 5. Springer Science & Business Media, 2006. 33

[35] E.W. Dijkstra. *A short introduction to the art of programming*, volume 4. Technische Hogeschool Eindhoven Eindhoven, 1971. 29

[36] H. Djellab, K. Djellab, and M. Gourgand. A new heuristic based on a hypergraph representation for the tool switching problem. *International Journal of Production Economics*, 64(1—3) :165 – 176, 2000. 69

[37] J. Dréo, A. Pétrowski, P. Siarry, and E. Taillard. *Métaheuristiques pour l'optimisation difficile*. Eyrolles, 2003. 36

[38] S. Dreyfus. Richard bellman on the birth of dynamic programming. *Operations Research*, 50(1) :48–51, 2002. 34

[39] F.Y. Edgeworth. *Mathematical psychics : An essay on the application of mathematics to the moral sciences*, volume 10. Kegan Paul, 1881. 36

[40] J. Edmonds. Paths, trees, and flowers. *Canadian Journal of mathematics*, 17(3) :449–467, 1965. 30

[41] M. Ehrgott. Multicriteria optimization. vol. 491 of lecture notes in economics and mathematical systems, 2005. 39

[42] F. Fages. *Programmation logique par contraintes*. Ellipses Paris, France, 1996. 23

[43] P. Feautrier. Parametric integer programming. *Revue française d'automatique, d'informatique et de recherche opérationnelle*, 22(3) :243–268, 1988. 10, 11

[44] T. Feder, R. Motwani, R. Panigrahy, S. Seiden, R. Van Stee, and A. Zhu. Combining request scheduling with web caching. *Theoretical Computer Science*, 324(2-3) :201–218, 2004. 70

[45] C.M. Fonseca and P.J. Fleming. Multiobjective optimization and multiple constraint handling with evolutionary algorithms. i. a unified formulation. *IEEE Transactions on Systems, Man, and Cybernetics-Part A : Systems and Humans*, 28(1) :26–37, 1998. 38

[46] J.C Fournier. *Théorie des graphes et applications : Avec exercices et problèmes*. Lavoisier, 2011. 17, 21, 29, 39

[47] C. Gagné. Hiérarchie de mémoire. Cours Universitaire, 2011. 4

[48] M.R. Garey and D.S. Johnson. *Computers and Intractability : A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1979. 17, 31, 32, 39, 62, 68, 74

[49] Gianpaolo Ghiani, Antonio Grieco, and Emanuela Guerriero. Solving the job sequencing and tool switching problem as a nonlinear least cost hamiltonian cycle problem. *Networks*, 55(4) :379–385, 2010. 69

[50] F.W Glover. Tabu search-part i. *ORSA Journal on computing*, 1(3) :190–206, 1989. 35

[51] F.W Glover. Tabu search : A tutorial. *Interfaces*, 20(4) :74–94, 1990. 35

[52] D.E Golberg. Genetic algorithms in search, optimization, and machine learning. *Addion wesley*, 1989 :102, 1989. 36

[53] D.E Golberg. *Genetic algorithms*. Pearson Education India, 2006. 36

[54] D. Goldberg. Genetic algorithms in optimization, search and machine learning. *Addison Wesley*, 905 :205–211, 1989. 38

[55] O. Goldreich. *Computational complexity : a conceptual perspective*. Cambridge University Press, 2008. 39

[56] R. Grisel and N. Abouchi. Les systèmes embarqués : Introduction. Cours Universitaire, 2005. 4

[57] C. Guéret, C.and Prins and M. Sevaux. Programmation linéaire. 2000. 19

[58] K. Hadj Salem, Y. Kieffer, and M. Mancini. Efficient algorithms for memory management in embedded vision systems. In 11^{th} *IEEE Symposium on Industrial Embedded Systems, SIES 2016, Krakow, Poland, May 23-25, 2016*, pages 177–182, 2016. 134

[59] K. Hadj Salem, Y. Kieffer, and M. Mancini. Formulation and practical solution for the optimization of memory accesses in embedded vision systems. In *Proceedings of the 2016 Federated Conference on Computer Science and Information Systems, FedCSIS 2016, Gdańsk, Poland, September 11-14, 2016*, pages 609–617, 2016. 134

[60] K. Hadj Salem, Y. Kieffer, and M. Mancini. Memory management in embedded vision systems : Optimization problems and solution methods. In *Conference on Design & Architectures for Signal & Image Processing, DASIP 2016, Rennes, France, October 12-14, 2016*, pages 200–207, 2016. 134

[61] K. Hadj Salem, Y. Kieffer, and M. Mancini. *Meeting the Challenges of Optimized Memory Management in Embedded Vision Systems Using Operations Research*, pages 177–205. Springer International Publishing, Cham, 2018. 134

[62] K. Hadj Salem, Y. Kieffer, and S. Mancini. Optimizing memory hierarchy in the embedded vision systems. In 27^{th} *European Conference on Operational Research, EURO 2015, Glasgow, UK, July 12-15, 2015*, 2015. 134

[63] K. Hadj Salem, Y. Kieffer, and S. Mancini. Minimisation des accès mémoires dans un cache intelligent pour les systèmes de vision embarquée. In $17^{ème}$ *Congrès Annuel de la Société Française de Recherche Opérationnelle et Aide à la Décision, ROADEF 2016, Compiègne, France, 10-12 Février, 2016*, 2016. 134

[64] K. Hadj Salem, Y. Kieffer, and S. Mancini. Optimisation du fonctionnement d'un contrôleur de buffers pour les systèmes de vision embarquée. In *Conférence d'informatique en Parallélisme, Architecture et Système, ComPAS 2016, Lorient, France, 5-8 Juillet, 2016*, 2016. 134

[65] K. Hadj Salem, Y. Kieffer, and S. Mancini. Minimisation du temps total de traitement pour optimiser le fonctionnement d'un contrôleur de buffers pour les systèmes de vision embarquée. In $18^{ème}$ *Congrès Annuel de la Société Française de Recherche Opérationnelle et Aide à la Décision, ROADEF 2017, Metz, France, 22-24 Février, 2017*, 2017. 134

[66] Y.Y. Haimes, L.S. Ladson, and D.A. Wismer. Bicriterion formulation of problems of integrated system identification and system optimization, 1971. 38

[67] A. Hertz, G. Laporte, M. Mittaz, and K.E. Stecke. Heuristics for minimizing tool switches when scheduling part types on a flexible machine. *IIE transactions*, 30(8) :689–694, 1998. x, 108, 109

[68] A. Hertz and M. Widmer. An improved tabu search approach for solving the job shop scheduling problem with tooling constraints. *Discrete Applied Mathematics*, 65(1-3) :319–345, 1996. 70

[69] F.S. Hiller and G.J. Lieberman. Introduction to operations research, 1995. 39

[70] J.H. Holland. *Adaptation in natural and artificial systems : an introductory analysis with applications to biology, control, and artificial intelligence.* U Michigan Press, 1975. 36

[71] S. Irani. Page replacement with multi-size pages and applications to web caching. In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, pages 701–710. ACM, 1997. 70

[72] E.L. Johnson, G.L. Nemhauser, and M.WP. Savelsbergh. Progress in linear programming-based algorithms for integer programming : An exposition. *Informs journal on computing*, 12(1) :2–23, 2000. 36

[73] P. Kadionik. Les systèmes embarqués : une introduction. Cours Universitaire, 2005. 4

[74] N. Karmarkar. A new polynomial-time algorithm for linear programming. In *Proceedings of the sixteenth annual ACM symposium on Theory of computing*, pages 302–311. ACM, 1984. 34

[75] R.M. Karp. Reducibility among combinatorial problems. In *Complexity of computer computations*, pages 85–103. Springer, 1972. viii, 31, 32

[76] KW Keung, WH Ip, and TC Lee. The solution of a multi-objective tool selection model using the ga approach. *The International Journal of Advanced Manufacturing Technology*, 18(11) :771–777, 2001. 67

[77] L.G. Khachiian. Polynomial algorithm in linear programming. In *Akademiia Nauk SSSR, Doklady*, volume 244, pages 1093–1096, 1979. 34

[78] Y. Kieffer, K. Hadj Salem, and S. Mancini. Multi-objective optimization for the scheduling of embedded vision accelerators. In *The* 29^{th} *Conference of the European Chapter on Combinatorial Optimization, ECCO 2016, Hungary, Budapest, May 26-28, 2016*, 2016. 134

[79] S. Kirkpatrick, M.P. Vecchi, and Others. Optimization by simmulated annealing. *science*, 220(4598) :671–680, 1983. 35

[80] J.D. Knowles. *Local-search and hybrid evolutionary algorithms for Pareto optimization*. PhD thesis, Department of Computer Science Local-Search and Hybrid Evolutionary Algorithms for Pareto OptimiZation Joshua D. Knowles Submitted in partial fulfilment of the requirements for the degree of Doctor of Philosophy in Computer Science. Department of Computer Science, University of Reading, 2002. 38

[81] A. Konak and S. Kulturel-Konak. An ant colony optimization approach to the minimum tool switching instant problem in flexible manufacturing system. In *2007 IEEE Symposium on Computational Intelligence in Scheduling*, 2007. 67

[82] G. Laporte, J.J. Salazar-Gonzalez, and F. Semet. Exact algorithms for the job sequencing and tool switching problem. *IIE Transactions*, 36(1) :37–45, 2004. 69

[83] R. Larbi. *Optimisation des séquences d'opérations dans une plateforme de crossdocking*. PhD thesis, Grenoble, INPG, 2008. 70

[84] S. Lin and B.W. Kernighan. An effective heuristic algorithm for the traveling-salesman problem. *Operations research*, 21(2) :498–516, 1973. 63

[85] C. Low, M. Ji, C.J. Hsu, and C.T. Su. Minimizing the makespan in a single machine scheduling problems with flexible and periodic maintenance. *Applied Mathematical Modelling*, 34(2) :334–342, 2010. 70

[86] Z. Mammeri. Chapitre 1 : Introduction aux systèmes embarqués et temps réel. Cours Universitaire, 2000. 4

[87] M.S. Manasse, L.A. McGeoch, and D.D. Sleator. Competitive algorithms for server problems. *Journal of Algorithms*, 11(2) :208 – 230, 1990. 69

[88] S. Mancini. *Gestion des données dans les systèmes numériques intégrés*. Habilitation à diriger des recherches en informatique, Université de Grenoble : laboratoires GIPSA-Lab et TIMA - École doctorale MST2II, Février 2013. 93 pages. 9

[89] S. Mancini and F. Rousseau. Enhancing non-linear kernels by an optimized memory hierarchy in a high level synthesis flow. In *Proceedings of the Conference on Design, Automation and Test in Europe*, pages 1130–1133. EDA Consortium, 2012. viii, x, xiv, xvi, 6, 9, 11, 12, 14, 18, 39, 45, 62, 86, 109, 111, 114, 130, 131, 134

[90] S. Mancini and F. Rousseau. Optimisation d'accélérateurs matériels de traitement par incorporation d'un gestionnaire de données et de contrôle dans un flot de HLS. In *Conférence en Parallélisme, Architecture et Système*, 2013. x, xiv, 9, 14, 62, 63, 81, 109, 111, 114, 130, 131

[91] K. Marriott and P.J.J Stuckey. *Programming with constraints : an introduction*. MIT press, 1998. 23

[92] B. Matzliach. The online tool switching problem with non-uniform tool size. *International Journal of Production Research*, 36(12) :3407–3420, 1998. 67

[93] B. Matzliach and M. Tzur. Storage management of items in two levels of availability. *European Journal of Operational Research*, 121(2) :363–379, 2000. 67

[94] K.M. Miettinen. Non-linear multi-objective optimization, 1999. 39

[95] C.H. Papadimitriou. *Computational complexity*. John Wiley and Sons Ltd., 2003. 39

[96] V. Pareto. Cours d'économie politique. *F. Rouge, Lausanne, Suisse*, 1896. 36

[97] M.L. Pinedo. *Scheduling : Theory, Algorithms, and Systems*. Springer-Verlag New York, New York, NY, USA, 2012. 39

[98] C. Privault. *Mathematical models for the off-line and on-line management of tool switches on a flexible machine*. Theses, Université Joseph-Fourier - Grenoble I, 1994. 67, 68, 69

[99] C. Privault. *Mathematical models for the off-line and on-line management of tool switches on a flexible machine (Chapitre1 : La gestion des outils dans un système flexible de production)*. Theses, Université Joseph-Fourier - Grenoble I, January 1994. 65

[100] C. Privault and G. Finke. Modelling a tool switching problem on a single nc-machine. *Journal of Intelligent Manufacturing*, 6(2) :87–94, 1995. 68

[101] C. Raduly-Baka and O.S. Nevalainen. The modular tool switching problem. *European Journal of Operational Research*, 242(1) :100–106, 2015. 67

[102] ROADEF. *La Recherche Opérationnelle en France*. www.roadef.org, 2011. 17

[103] M. Sakarovitch. *Optimisation combinatoire, graphes et programmation linéaire*, volume 1. Hermann, 1984. 17, 21, 39

[104] M. Sakarovitch. *Optimisation combinatoire, programmation discrète*, volume 2. Hermann, 1984. 17, 39

[105] M. Sakarovitch. *Linear programming*. Springer Science and Business Media, 2013. 39

[106] J. Sifakis. Logiciels et systèmes embarqués : Enjeux economiques, défis scientifiques et directions de travail. Cours Universitaire, 2002. 4

[107] W. Stallings. *Organisation et architecture de l'ordinateur*. Pearson Education, 2003. viii, 4, 8

[108] S.M. Sulaiman, S.and Shamsuddin, F. Forkan, and A. Abraham. Intelligent web caching using neurocomputing and particle swarm optimization algorithm. In *Modeling & Simulation, 2008. AICMS 08. Second Asia International Conference on*, pages 642–647. IEEE, 2008. 70

[109] E.G Talbi. *Metaheuristics : from design to implementation*, volume 74. John Wiley & Sons, 2009. 36

[110] C.S. Tang and E.V. Denardo. Models arising from a flexible manufacturing machine, part i : Minimization of the number of tool switches. *Operations Research*, 36(5) :767–777, 1988. 53, 67, 68, 69, 74, 90, 91

[111] C.S. Tang and E.V. Denardo. Models arising from a flexible manufacturing machine, part ii : Minimization of the number of switching instants. *Operations Research*, 36(5) :778–784, 1988. 67

[112] H.C. Thomas, C.E. Leiserson, R.L. Rivest, and C. Stein. *Introduction to algorithms*, volume 6. MIT press Cambridge, 2001. 29

[113] A.L. Thornton and S.J. Sangwine. Log-polar sampling incorporating a novel spatially variant filter to improve object recognition. In *Sixth International Conference on Image Processing and Its Applications*, volume 2, pages 776–779, 1997. 109

[114] M. Tzur and A. Altman. Minimization of tool switches for a flexible manufacturing machine with slot assignment of different tool sizes. *IIE Transactions*, 36(2) :95–110, 2004. 67

[115] E.L. Ulungu and J. Teghem. Multi-objective combinatorial optimization problems : A survey. *Journal of Multi-Criteria Decision Analysis*, 3(2) :83–104, 1994. 39

[116] D.A. Van Veldhuizen. Multiobjective evolutionary algorithms : classifications, analyses, and new innovations. Technical report, DTIC Document, 1999. 38

[117] P. Viola and M.J. Jones. Robust real-time face detection. *International journal of computer vision*, 57(2) :137–154, 2004. 109

[118] C. Vira and Y.Y. Haimes. *Multiobjective Decision Making : Theory and Methodology*. North-Holland, 1983. 38

[119] B.H. Zhou, L.F. Xi, and Y.S. Cao. A beam-search-based algorithm for the tool switching problem on a flexible machine. *The International Journal of Advanced Manufacturing Technology*, 25(9–10) :876–882, 2005. x, 69, 108, 109

[120] E. Zitzler and S. Künzli. Indicator-based selection in multiobjective search. In *International Conference on Parallel Problem Solving from Nature*, pages 832–842. Springer, 2004. 38